# Preventing integrated circuit piracy using reconfigurable logic barriers

by

Alex Clark Baumgarten

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Joseph Zambreno, Major Professor
Akhilesh Tyagi, Major Professor
Phillip Jones
Thomas Daniels

Iowa State University

Ames, Iowa

2009

# DEDICATION

Soli Deo Gloria

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

## ACKNOWLEDGEMENTS

I would like to thank my parents, family, and friends who have been instrumental in laying a solid foundation for my life. You instilled in me a set of values, helped me with all of life's challenges and shaped me into who I am today.

A very special thank you to my wife Elizabeth for her relentless encouragement offered with a bright smile. You always believed in me and were my biggest supporter.

I would like to offer my appreciation to Joseph Zambreno and Akhilesh Tyagi, my major professors, for their assistance and support in completing this thesis and guidance throughout graduate school. I am also grateful for the effort of the other committee members, Phillip Jones and Thomas Daniels, in helping me to improve this research.

## ABSTRACT

With each new feature size, integrated circuit (IC) manufacturing costs increase. Rising expenses cause the once vertical IC supply chain to flatten out. Companies are increasing their reliance on contractors, often foreign, to supplement their supply chain deficiencies as they no longer can provide all of the services themselves. This shift has brought with it several security concerns classified under three categories: (1) *Metering* - controlling the number of ICs created and for whom. (2) *Theft* - controlling the dissemination of intellectual property (IP). (3) *Trust* - controlling the confidence in the IC post-fabrication. Our research focuses on providing a solution to the metering problem by restricting an attacker's access to the IC design. Our solution modifies the CAD tool flow in order to identify locations in the circuit which can be protected with reconfigurable logic barriers. These barriers require the correct key to be present for information to flow through. Incorrect key values render the IC useless as the flow of information is blocked. Our selection heuristics utilize observability and controllability don't care sets along with a node's location in the network to maximize an attacker's burden while keeping in mind the associated overhead. We implement our approach in an open-source logic synthesis tool, compare it against previous solutions and evaluate its effectiveness against a knowledgeable attacker.

# CHAPTER 1. INTRODUCTION

A significant security threat has emerged as the once vertical Integrated Circuit (IC) supply chain has flattened into a horizontal model. In the past, IC design and fabrication were mostly done by the same entity since the cost to build a foundry, although expensive, was a reasonable investment. But, as the feature size and time-to-market shrink coupled with the demands for lower-power, high performance ICs, the cost required to establish a full-scale foundry becomes prohibitive. In 2005, a full-scale 300mm wafer 65nm process foundry cost $3 Billion to build with mask creation costing upwards of $2 million [10]. In such a market, very few companies can afford to be experts in all areas driving the market to specialize. Intellectual Property (IP) vendors have emerged who specialize in a certain functional unit and license it to others for use in their Application Specific Integrated Circuit (ASIC) designs. The IC design companies integrate third-party IP along with their own to create an IC design.

Finally, contract foundries have begun to harness economies of scale as they spread the large capital required to build the foundry among their clients. These contract foundries, originally driven by cheap labor, established themselves throughout Asia. Now, as labor prices and automation have risen, the motivating factor for location has diminished. But, the concentration of expertise is well established in these areas as can be seen in recent trends. According to the Semiconductor Industry Association, the share of dollars of semiconductors shipped from the United States was 66.7% of the global market in 1976 while Asia, including Japan only made up 11.4%. Since then, there has been a steady shift in these trends. As of the end of the year 2008, the United States dropped to 15.5% of the global market while Asia grew to 69.2%. These trends can be seen in Figure 1.1 which shows a sharp transfer of the market in the mid-1980s and then again over the last 10 years with a 30% swing [55]. Not only have the

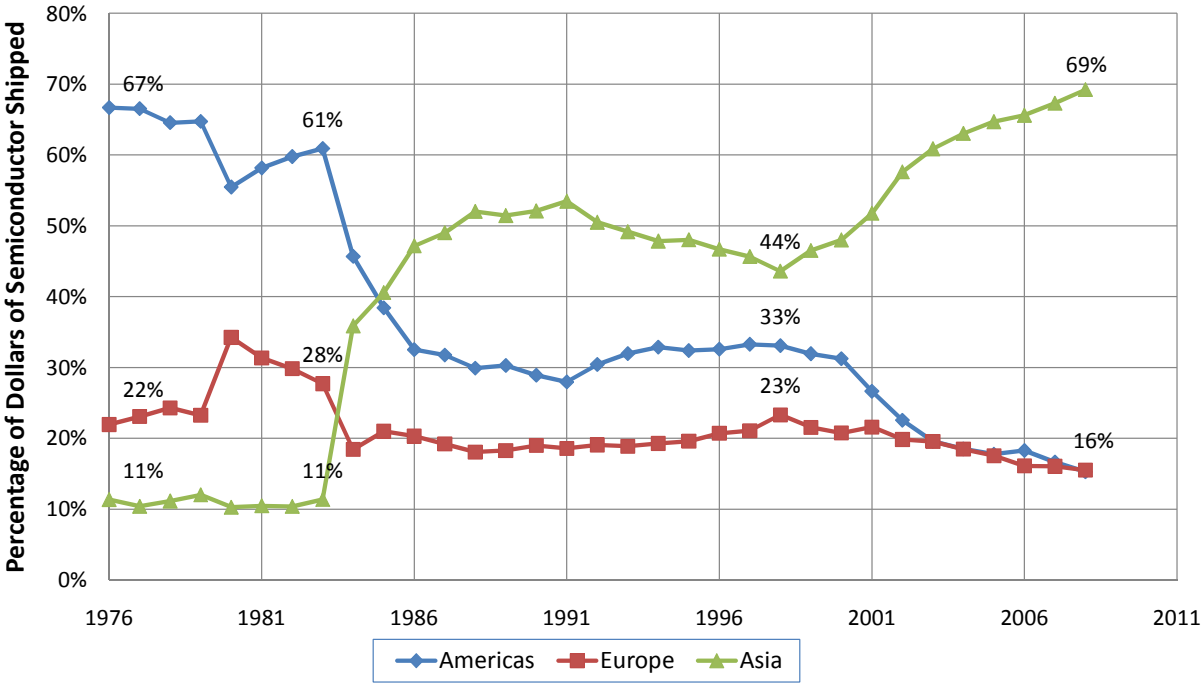



Figure 1.1: Percentage of dollars of semiconductors shipped over time for America, Europe, and Asia

dollars shifted power, but the number of foundries along with it. In 2005, 16 out of the 59 300mm foundries worldwide were in the United States, but a majority of the 16 were special purpose fabrication facilities not tooled for general purpose ASIC design. These changes have not gone unnoticed as even the United States senate has suggested plans of action [56].

Although the fabrication of ICs is dominated by the foreign market, the IC industry in the United States is still alive, but its business model has changed in recent years. Large companies such as Texas Instruments and AMD that once fabricated their own ICs have opted for the fabless model, contract out their IC fabrication. Others such as Qualcomm, Broadcom, and Nvidia who all recently broke into the top 20 semiconductor sales leaders are showing that fabless companies can survive. The paradigm shift has created a serious security threat as control of a process that once could be monitored closely, is being outsourced. A recent report by SEMI [2] surveyed companies working in the semiconductor industry representing more than 50% of the annual semiconductor revenue. Of the respondents, 90% reported the existence of IP infringement in their company and 54% reported it as a serious or extremely

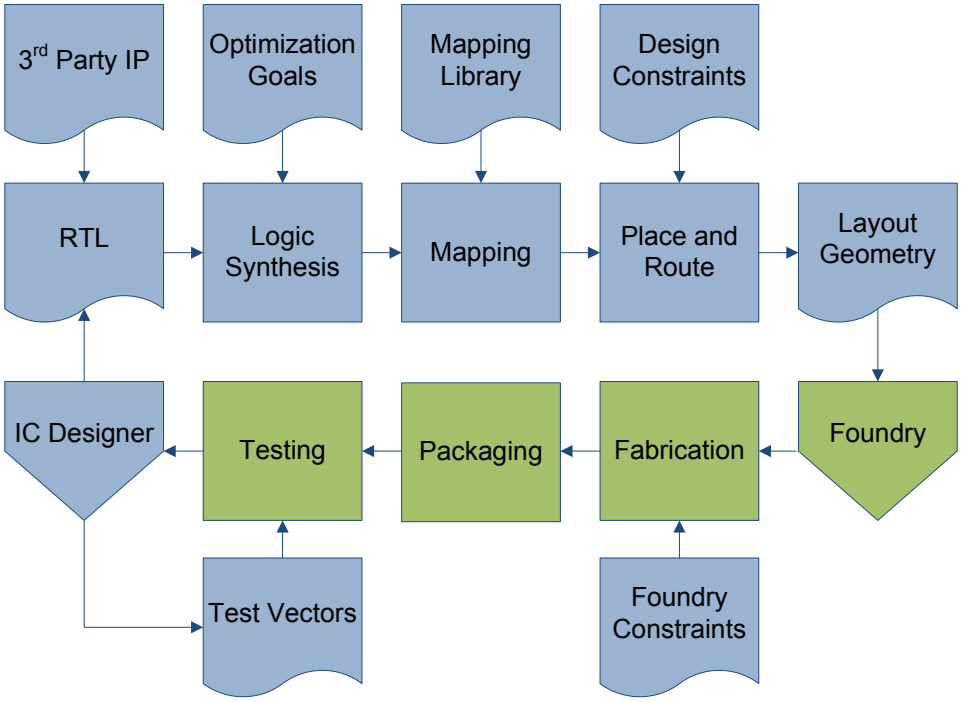



Figure 1.2: Process flow for IC fabrication from RTL to IC

serious threat. Without adequate protections schemes, the $1 Billion lost to IP piracy each day [25] will continue to be fueled by cases such as the fake NEC company [15] and those reported by [2]. Adaptions to the horizontal model need to be made in order to protect the IP that is the sole product of some companies and precious to all companies.

The process flow for taking a napkin design and turning it into a completed IC can be seen in Figure 1.2. The process starts with the IC Designer who creates the Register Transfer Level (RTL) description of the IC according to the specifications of the project. This description, typically in a form of a hardware description language (HDL), is the IP of the designer. The RTL then begins a series of electronic design automation (EDA) steps using software tools by companies such as Cadence, Mentor Graphics, and Synopsys in order to produce a finalized netlist. These steps begin with logic synthesis where the RTL design is turned into a directed acyclic graph (DAG) representation, $\mathcal{N} = (\mathcal{V}, \mathcal{E})$. The nodes in $\mathcal{V}$ each represent a Boolean logic function and the partial ordering of the nodes cause it to be acyclic. The interconnections between the nodes represent the flow of information as the output of one Boolean function is

fed into another node. The inputs to the graph represent the primary inputs to the circuit and the outputs, correspondingly, the primary outputs.

It is in this stage that logic optimization also occurs. Complex heuristics search for graph transformations that optimize the design goals of the system in order to increase circuit quality. These transformations leave the circuit functionally equivalent although, it is structurally optimized. The optimized design is sent to the mapping stage where the Boolean functionality of the DAG is converted into primitive gates representing the same functionality. The next stage takes the mapping library which describes the primitive gates and their functionality as its input. The mapping phase can be seen as a covering problem where the Boolean logic making up the DAG must be covered by primitive gates specified in the mapping library. Once mapping is complete, placing and routing occur. This stage determines the best locations and best connections between each component of the design. By taking into account the design constraints of area, delay, and power, optimizations can be made for the location and connections. At this point, all of the original functionality specified in the RTL stage is present and there are no violations in design constraints. The design is exported to a layout level geometry format such as GDSII or more recently, OASIS which are large plaintext descriptions of the polygons representing the physical descriptions of how the IC should be fabricated.

This exact blueprint, the layout geometry, is sent to the foundry for processing. The foundry reads the layout geometry and legitimately makes modifications to the polygons in order to support the design. This could include increasing the width of some polygons in order to conform to the minimum granularity of etching that the foundry can reliably manufacture. Once the layout geometry has been modified to conform to all of the fabrication rules, masks are created which are needed for etching each layer of the wafer. The fabrication process continues through many stages of manufacturing concluding with the separation of individual ICs from the larger wafers. Basic tests are performed to bin and sort the ICs before they are packaged. In packaging, the die is covered with plastic or ceramic and connections are made from the wafers I/O ports to the metal leads. The final step, IC testing, utilizes the provided test vectors to ensure quality and consistency throughout the process.

The concerns that arise from IC fabrication can be split into three basic categories: Metering, Theft, and Trust. Each of these categories broadly addresses whether extra ICs beyond the purchase order have been produced, whether information including IP has been gained by unauthorized individuals, and whether any tampering has occurred in the ICs received from the foundry. Although solutions in each category are needed and warranted, we chose to focus on a solution to Metering. Significant research effort has been spent to reduce the growing IC piracy. Various schemes to uniquely identify ICs, even those produced from the same wafer, coupled with a method for registering the unique ID of each IC have been proposed. Building upon the passive metering approaches, more active ones add an additional FSM with replicated states which requires a unique key pattern to unlock the IC before becoming functional. Other approaches focus on combinational locking mechanisms by disseminating the FSM throughout the IC. In each of the approaches utilizing a key, a secure communication framework needs to be established, usually based on well-known public-key encryption methods. Our approach provides an improvement to the combinational locking approaches by recognizing several shortcomings of previous schemes.

Where the previous methods did not allow synthesis tools much flexibility and relied on random selection criteria, our approach raises the level of abstraction to increase flexibility and opacity, and establishes node selection heuristics that maximize security while minimizing overhead. Our results show that using a cut-based heuristic selection approach is more efficient than random node selection and that the selection of additional nodes beyond a single cut, does not lead to significant gains in security. We have also shown that our approach is resilient to an intelligent attacker and that our selection heuristic increases this resiliency as its parameters are tuned.

At a high level, our approach adds reconfigurable logic barriers to the IC before it is fabricated. These barriers completely separate the inputs from the outputs such that every path from a primary input to a primary output must pass through one of the barriers. This is analogous to security checks at the airport. The barriers separate all of the terminals from the outside world. If a passenger wants to enter the concourse, they must pass through security.

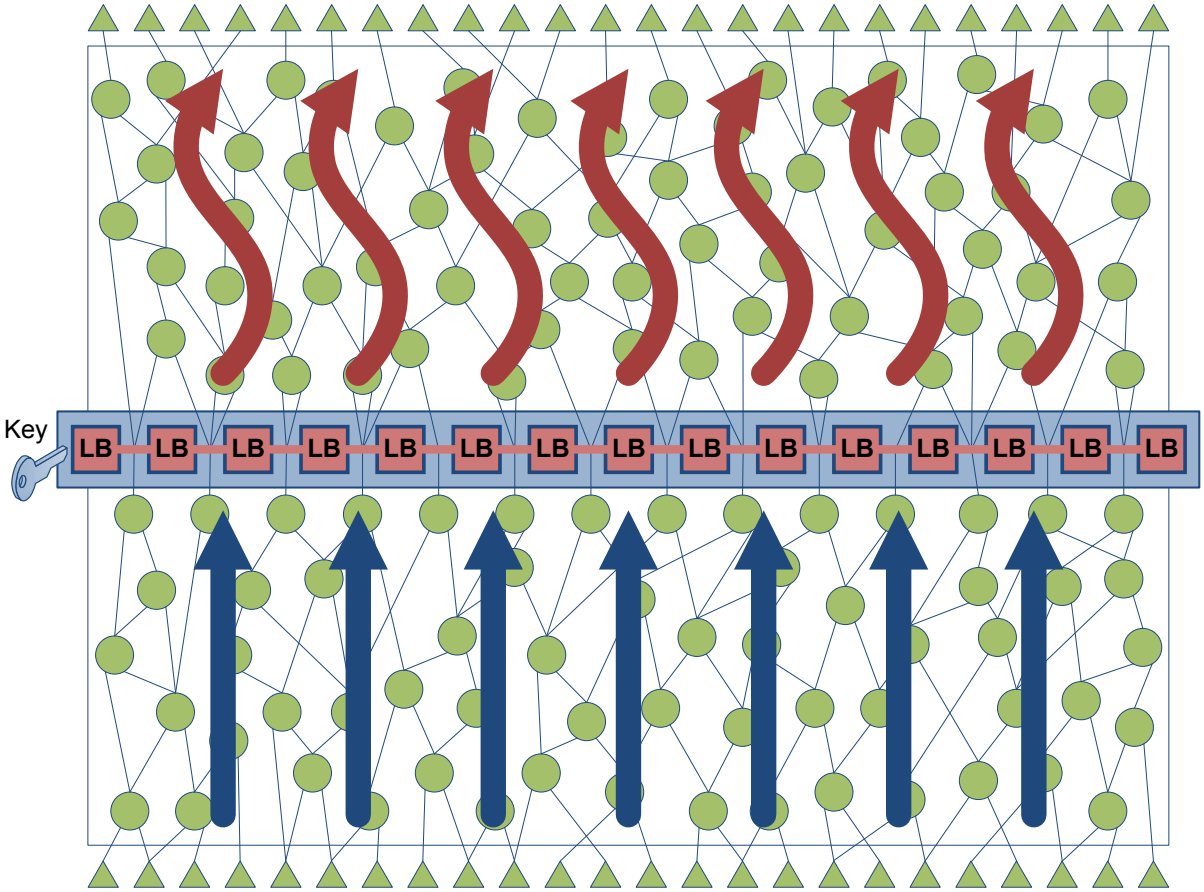



Figure 1.3: High level approach for IC piracy protection. Logic Barriers (LB) block the information flow (straight arrows) such that only unintelligble data (curved arrows) continues to flow towards the outputs when the key is incorrect.

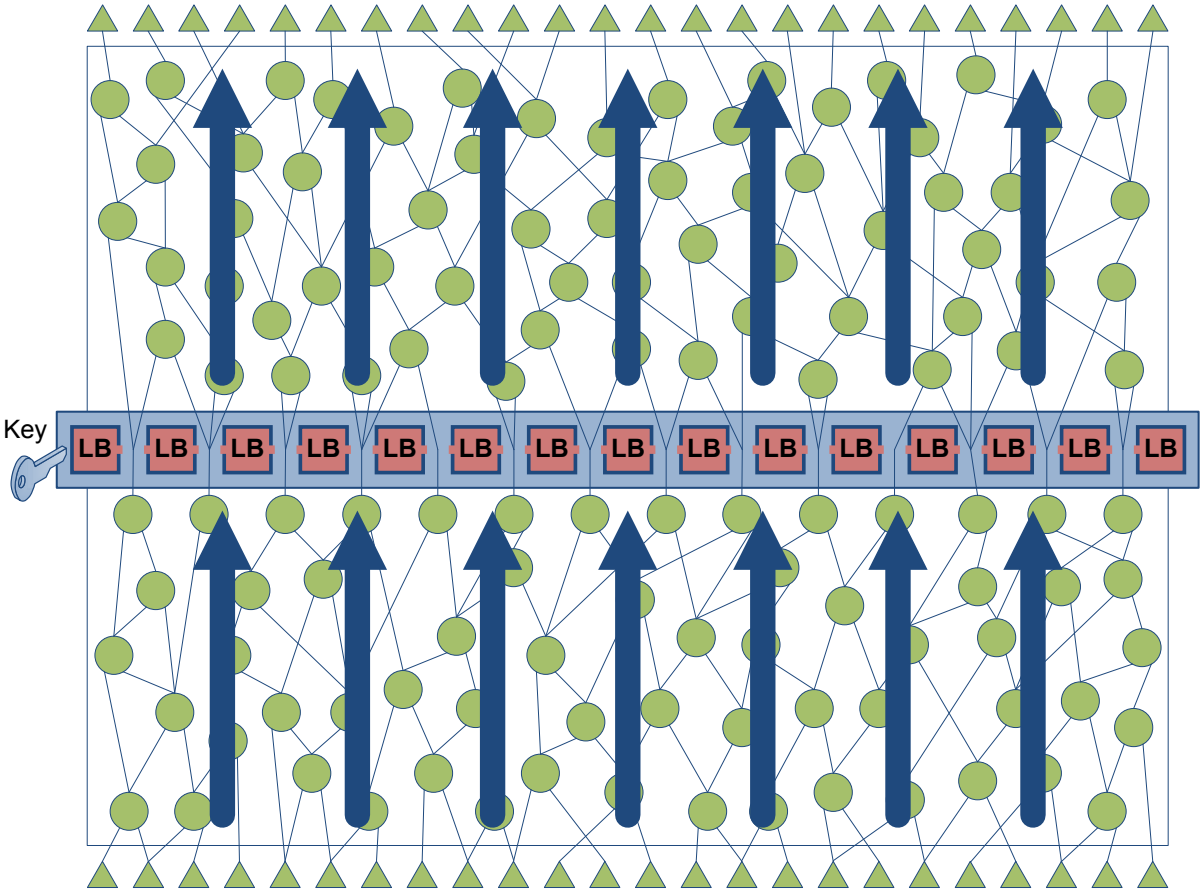



Figure 1.4: High level approach for IC piracy protection. Logic Barriers (LB) allow the information flow (straight arrows) to continue through when the key is correct.

There is no legal path that they can take that would allow them to bypass security. Figure 1.3 and Figure 1.4 visually depict our approach. In both figures, the logic network for the IC is represented as the graph in green (light gray). The inputs to the circuit are the triangles at the bottom of the figure and the outputs, the triangles at the top. Each of the circles represents a piece of the circuit's functionality with the edges of the graph indicating the connections between the logic elements. The thick blue (medium gray) arrows represent the general trends of the information flow as it moves through the circuit. Large straight arrows show data flowing as it would in the original circuit not augmented with our approach and the curved arrows show data that has been altered to be unintelligible. In the middle of the figures, red (dark gray) checkpoint boxes have been placed with red (dark gray) interconnections between the barriers. When the checkpoint is closed as in Figure 1.3, the data becomes skewed and no longer represents the expected data. When they are open, however, the data is able to flow through them without being affected. The difference between having the checkpoint gates opened and closed is the presence of the correct key. Without a valid key, the gate cannot be opened, but once the correct key has been inserted, the checkpoint allows the data to flow through and the circuit functions identical to the original unprotected circuit.

Our work addresses: The need for providing barriers to information flow. How barriers can be efficiently implemented. The best locations for the barriers taking into consideration the effectiveness and overhead required to obscure the data. Analysis of our barrier implementation including comparisons with other approaches.

Our contributions include: Proof of concept implementations of hardware Trojans. A combinational locking scheme integrated into a standard CAD tool flow to prevent IC piracy. The first metering scheme that does not disclose the entire schematic to the foundry. Efficient node selection heuristics for maximizing the security while minimizing the associated overhead. Analysis of the improvement over previous schemes and its resiliency to attack.

The rest of the paper is organized as follows: Chapter 2 discusses related works relevant to ours. Chapter 3 provides a threat model for the IC supply chain describing where the attacker can enter and what they have to gain from the attack throughout each of the stages. Chapter 4

focuses on the background material and a discussion of our proposed approach. Chapter 5 analyzes the experiments performed and provides quantitative performance measurements. Chapter 6 wraps up the work with a summary of our approach. Also, the Appendix details our experience creating proof of concept Hardware Trojans.

# CHAPTER 2. RELATED WORK

Extensive research has been done in the field of IC supply chain security and has been classified here into three broad categories: Metering, Theft, and Trust. The first category deals with controlling the number of ICs produced and also limiting access to those ICs. The second category addresses unauthorized access to information. This may take the form of IP core theft or access to hardcoded secrets and algorithms, etc. Finally, the third category examines the level of confidence that can be placed in the artifacts of the IC supply chain as they pass from one stage to another.

## 2.1 Metering

Metering, deals with controlling the number ICs produced and for whom they are produced. Because of the prohibitively high costs to create foundries, contract fabrication facilities are used to supply a certain number of ICs of a given circuit design. Although the foundry may produce the requested number of ICs according to the contract, the current supply chain design does not enforce strict limiting on additional ICs produced which may be sold on the black market. Current research is attempting to provide methods that allow the IC designer to control the number of ICs produced or at least the usability of those ICs.

Metering approaches fit into either a passive or an active scheme. The passive approaches uniquely identify each IC and register that identity. Later, suspect ICs can be checked for proper registration. The uniqueness is usually derived from an uncloneable manufacturing variability that is unique for every IC, even those on the same wafer. Both temporal and spatial uniformity are tainted with the inherent parasitics of the IC fabrication process which decreases yield, but is exploited for IC identification. Variabilities such as the threshold mis-

match in MOSFET arrays [43], variability of silicon [44], delay characteristics [38], and Physical Unclonable Functions (PUFs) [24, 32, 51, 61, 64, 66] are all used to uniquely identify an IC. It should be noted that not all manufacturing variabilites provide adequate security guarantees [45].

The other approach, active metering, locks each IC until it is unlocked by the IP holder. Most active metering approaches also utilize an unclonable manufacturing variability to uniquely identify an IC, but they add an active locking mechanism to render the IC useless until properly unlocked. In [8], each IC generates a unique ID based on a spatial variability. The ICs are initialized to a locked state on power up, but can be unlocked by the IP holder who uses the unique ID to generate an unlocking key. [9] augments this process by replicating states of the FSM to increase security and by adding a key distribution process. Foundries are supplied with a set of challenge inputs by the IP holder which they apply to each IC. The response is securely transmitted to the IP holder who generates a unique unlocking key for that IC. A secondary security measure implemented in [8], augments the FSM with black hole states. Once the system enters into one of these states, it can never be unlocked since there are no valid transitions out of them. These black hole states are triggered by spurious inputs such as an attacker trying to brute-force the key combinations.

Another notable example of active metering is EPIC [54]. Instead of creating a FSM to lock the design, XOR gates are scattered randomly throughout the design on non-essential wires. One of the inputs to the XOR gate is the original wire and the other, one of the bits of a key. When the correct key is entered, the XOR gates allow the circuit to function as expected. Conversely, if a bit of the key is wrong, that XOR gate inverts the signal. A key distribution framework is also established using public-key cryptography. In this framework, the IP Holder, foundry, and each individual IC generate a set of public and private keys. Extra logic allows the IC to generate keys and the exploited manufacturing variabilites allow them to be unique for each IC. Secure channels are then established between the IP Holder, foundry, and individual IC. The key that unlocks the XOR gates of an IC is encrypted with the ICs public key before transmission. Extra encryption/decryption steps allow for further

levels of authentication. Our approach is similar in the framework that it fits into, but we focus significant effort on the selection of locking locations instead of randomly picking them. By designing more sophisticated heuristics for choosing nodes to augment and by raising the granularity from XOR gates to LUTs, we have not only generalized the key space, but also increased the burden placed upon an attacker without increasing the number of bits required to lock the IC. Our approach also does not reveal the entire design to the foundry which removes the foundries opportunity to reverse engineer the design.

Also extending from [54] is [27], which shifts the order and responsibilities for establishing secure communication channels, but still utilizes exactly the same XOR locking scheme as [54] for authentication. Our focus is a bit different from [27] in that we concentrate on the selection heuristics and not the secure communication channels.

In [12], a FSM is added to the netlist with inputs connected directly to all of the primary inputs. The FSM is initially locked and can only be unlocked when the correct sequence of primary input values is entered. The single output of the FSM is attached to XOR gates dispersed throughout the IC such that a locked FSM will act as unwanted inverters throughout the logic. The XOR gates are chosen iteratively by greedily selecting the nodes with the highest fan-in and fan-out cones which intuitively have more effect on the circuit. Our approach also looks into the intrinsic properties of the circuit for node selection, but uses a combination of better defined metrics, observability don't care $\boldsymbol{ODC}$ sets and node positioning, as the selection criteria. Our metrics are well-known and much more precise than restricting observation to fan-in and fan-out.

Others have suggested wafer banking where each layer of the IC is fabricated in a different facility and combined together after production. The cost of this method is prohibitive due to the large overhead in manufacturing at multiple facilities, the decreased yield associated with the inability to test the IC at the foundry, and the ad hoc manufacturing process.

## 2.2 Theft

The second category of concern in the IC supply chain is theft of information. Although this category does overlap with metering, it is not necessarily equivalent. Where metering tries to control who and how many ICs are created, information theft more generally encompasses gaining information that was not originally intended to be disseminated. The creation of extra ICs by unauthorized users is one of the direct consequences of stealing the netlist, but other information such as hardcoded secrets, IP cores, algorithms, etc. may also be the target.

Many different watermarking and fingerprinting technologies have added a unique signature at varying levels to the IP. This signature allows the IP to be traced through the IC supply chain and if the IP is stolen, points directly to the source of the leak. The work in this area is discussed briefly recognizing its main shortcoming. A unique signature does not protect the IP from theft, except as a deterrent, rather it provides a basis for a litigation once the crime has been committed.

The most important issues in watermarking are defined by [37]. The authors categorize them into three areas, copy detection complexity, mark removal vulnerability, and mark integrity. These issues are discussed in [14] in a more formal manner in order to establish a solution that incorporates hierarchical redundancy.

The work of [34, 35] targets IP for FPGAs by adding extra LUTs that are used to encode a long stream of information. This stream acts as the watermark for the IP used on the FPGA and is specifically tailored for the recipient. A technique called tiling is used to minimize the amount of placement and routing needed for successive watermarked designs. Tiling partitions the circuit into sections such that only the modified sections need to be processed by the EDA tools. The watermarking scheme they use is based on prior work in [36].

A different set of approaches use a state transition graph (STG) that allows specified paths through the graph to exhibit a rare pattern [49, 50, 67]. This pattern can be uniquely created and used as an IP watermark. Further extensions to this idea, [3, 4, 5], search the inherent FSMs of a design for unused transitions and attempt to coincide the extra watermark transitions with these FSMs.

Instead of adding extra STGs to the design, the authors of [11, 33] utilize the NP-complete design space in many of the algorithms that drive synthesis, placement, and routing. By adding constraints to these optimization searches, unique IPs can be generated without sacrificing quality since a phase such as routing may choose different connections with equivalent performance. Later papers improve the result of constraint based watermarking by using DES alongside a different routing approach [47] and ordering standard cells based on a seed before routing [46].

Another set of solutions, [59, 68] followed by [26], provide a protection mechanism for IP developed for FPGAs. A mutual authentication of IP modules and hardware platforms are established where the FPGA is registered with a 3rd party and then the IP in the same manner. By communicating through the 3rd party, IP can be deployed on a specific FPGA in order to minimize IP theft. The extension to [59], [26], hardens the communication protocols by placing less trust on the 3rd party and provides more details and analysis about the IC identification mechanism.

## 2.3 Trust

Along with the other two categories, trust in ICs has become a first-rate priority [6]. Classical validation and verification methods check that a circuit produces the correct outputs for a given set of inputs while also meeting non-functional requirements. But performing to a specification does not ensure an un-tampered circuit. New approaches must guard against tampering and be able to check for the presence of malevolent circuitry or absence of critical components. In February 2005, the Defense Science Board released a report entitled the "Task Force on High Performance Microchip Supply" [10] which examined long-term trust and security in the microchips used by the United States government. The conclusion of the report was that "urgent action is recommended." Following this report, a DARPA initiative, Trust in Integrated Circuits [16], was started to provide more security to the supply chain. The program is broken into three phases with industry and government support at each step. The phases examine trust in ASIC designs, trust in untrusted foundries, and trust in FPGA designs respectively. An independent effort with similar intentions, the NSA program on Trusted Foundries [48], sets

certain standards that must be established before a foundry can receive a stamp of approval.

The term hardware Trojans broadly describes a category of attacks that adds or removes circuitry with malicious intent. This category can be further sub-divided to form a hardware Trojan taxonomy. Trojans may be identified by their physical attributes such as their architecture, size, and type, their activation characteristics, or how they manifest themselves [71]. Beyond broad category descriptions, [18] applies threat modeling to hardware in order to quantitatively assess the risk of each component in a design. Built upon [18], the authors of [60] add structural checking which examines the components of the IC against malicious ones. Our work does not deal with hardware Trojan classification, but being able to digest the various threats that Trojans pose and how they function is useful in designing protection schemes.

Several methods have been proposed which either try to secure the supply chain so that Trojans cannot enter into an IC or to verify that the received IC does not contain a Trojan. Although our main contributions do not deal directly with the issue of trust, the hardware hacking competition we entered, Section 6.2, describes some proof of concepts for adding Trojans into the IC supply chain. Several other works showing proofs of concept continue to spur the need for such research. In [31], a flexible framework for launching hardware Trojans sits alongside existing CPU hardware. The Trojan framework allows the malicious user to bypass memory checks for privileged access of the memory and to run in a shadow mode where malicious instructions can directly execute on the processor invisible to software.

In order to thwart such a Trojan insertion, research has focused on establishing a trust model and framework for building secure systems. In [70], the trust between the layers of an embedded system from the hardware up through the client software are examined. Many attempts to limit an attacker's ability to compromise the system have been made by using a secure processor. The work of [28] creates an AES cryptography engine using a normal EDA tool flow and a second design using wave dynamic differential logic in order to obscure the key against side-channel attacks. Both were fabricated on the same die and their effectiveness against attacks compared. Other approaches target the architecture of processors, [62, 63, 66, 65], which first begin with an encryption scheme for trusting main memory under attack [62, 63]. Later

in [66, 65], the secure processor was implemented in an FPGA and extra levels of security are added that do not trust any of the peripherals and aim to provide secure services during physical attacks. These implementations also describe an implementation of a PUF used for off-chip authentication. In [58], the authors create a multi-layer encryption protocol on a dual-processor architecture that can handle a compromised processor. The approach relies on different parts being manufactured at independent foundries with no collusions existing between them. With that assumption, the architecture is resilient to hardware attacks embedded from one foundry.

Similar to the main memory security approaches of [62, 63, 66, 65], [41] creates a memory architecture, XOM, for executing instructions from memory while providing copy protection and tamper-resistance. Later, [40] implements a trusted computing platform on top of the XOM architecture. Two related approaches, [22, 69], provide a memory security solution by encrypting the memory and providing an integrity check. In [22], a Parallelized data Encryption and Integrity Checking Engine (PE-ICE) is described which adds memory integrity checking to a block encryption algorithm. Then in [69], a one-time pad is used in combination with a CRC-based integrity checker to secure the memory. Finally, [73], utilizes a shadow memory that is initialized by a trusted person and then cannot be altered. Calls are checked with the shadow memory and differences signal an attack. In all of the cited work on secure memory and secure processors, the authors make the assumption that the IC design process is secure and the attacker only enters once the system is put into use. We make the opposite assumption, the foundry and the IC supply chain are vulnerable. Our work is different from the secure memory and secure processor work because we focus on a different attacker which means that the approaches are complementary in providing security guarantees for the whole system throughout its life-cycle.

The work in [72] tries to find a Trojan once it has been added to a circuit by first heuristically searching for all possible target locations in a circuit design. From the search, a set of test vectors are generated that stimulate the areas that have a higher probability of containing a Trojan. By decreasing the large search space in which the rare event triggering a Trojan occurs, the likelihood of discovering them increases. In [7], the authors utilize side-channel technology

in order to analyze a few ICs that are known to be good. From this analysis, a fingerprint for that particular IC is developed. Then, all of the remaining ICs can be checked against the fingerprint to ensure that they are the same. Their protection relies on a fine enough granularity of fingerprinting such that a malicious change to the IC will cause the fingerprint to be modified. Following along similar lines, the authors of [30, 39] create a fingerprint for ICs in order to ensure Trojan-free circuits by focusing on path delays for identification. Another fingerprinting scheme, [13], utilizes controllability and observability to identify the least controllable and least observable nodes, augment the circuit with a FSM to exercise these nodes and then generate a challenge response pair which corresponds to the outputs observed when a test input is applied which exercises the least observable and least controllable nodes. Their approach assumes that a Trojan will be placed on the least observable and least controllable nodes since there are fewer test vectors which exercise these values and therefore the chances of a Trojan being identified are smaller. With this assumption, they specifically create test vectors which target those nodes and check that the outputs are expected for that set of inputs. Our approach shares some commonality with [13] in that we both utilize the observability and controllability properties inherent in a circuit, but in very different ways. Where they use it to choose test vectors for Trojan identification, we use it for node selection in order to combat piracy.

# CHAPTER 3. THREAT MODEL

As a basis for establishing our security guarantees, it is imperative that we establish a threat model. In order to accurately characterize the attacker, we need to know what the goal of the attacker is and what resources they have available. As mentioned in Section 2, the goals of an attacker targeting the IC supply chain are three-fold. Pirate circuits, steal information, and implement Trojans. Protecting against an attacker trying to achieve one of these three goals is difficult because of the breadth of the supply chain, but a thorough classification of the attacker will allow for more pertinent defenses.

## 3.1 Attacker Taxonomy

Attackers can be classified on the basis of where in the supply chain they attack and then further subdivided by what access they have to the supply chain. From these assumptions, a list of what the attacker can gain in terms of the previously stated goals is described. Finally, the possible protections that stop or hinder the attacker are mentioned although not in detail as Section 2 covers these approaches. In each of the cases, the attacker may be a mole, a disgruntled employee, a competing company, a hacker, or a terrorist. We assume that the attacker has significant resources, but they are limited and that the benefit from attacking the IC supply chain must outweigh the resources expended. Along with the taxonomy descriptions, Figure 3.1 shows the stages of the IC supply chain, who the attacker is at each stage, and what information can be gained in each of the three attacker goals.

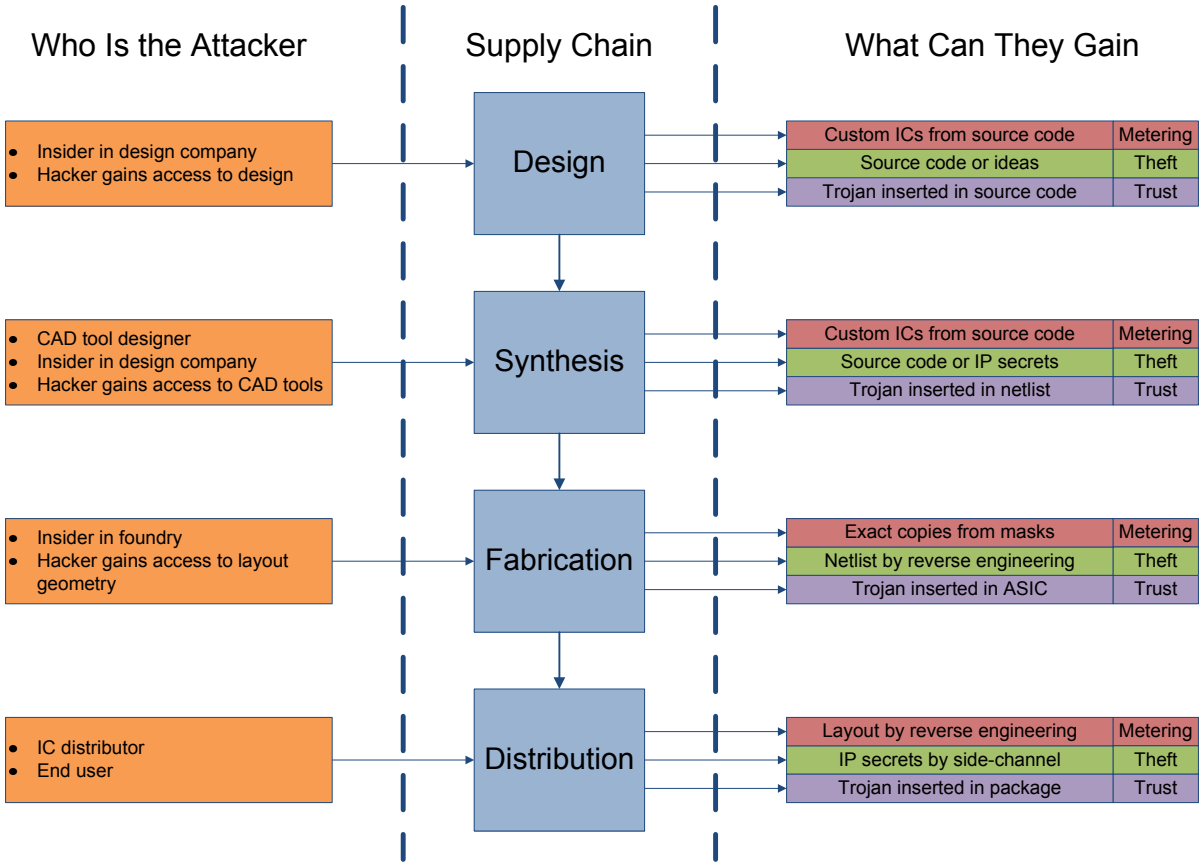



Figure 3.1: A taxonomy of an attacker including what is to be gained and how to protect against them at various levels of the IC supply chain

### 3.1.1 Design Attacker

**Who:** A design attacker enters the supply chain during the design phase between the conception of the idea and when the idea has fully materialized in the form of an RTL description. This attacker has full access to the design files and source code. The attacker is likely an insider who is given the access intentionally although they may also be a more traditional hacker who has compromised a computer system.

**Gained:**

- Metering - With access to the source code, custom ICs can be created. But, the attacker must have access to the foundry or enough resources to get the IC fabricated.
- Theft - Since the attacker has access to the entire design and source code, IP theft is trivial.
- Trust - If the attacker has write access, then adding and removing components to the design is also very feasible. Without that access, analysis can be done on the design in order to facilitate an attack at a later stage.

**Protection:** The measures necessary to protect computer systems that store IP are too numerous to mention and are out of the scope of this paper, but hardened security conscious networks can aid in IP protection. In the same manner, protecting IP from those who design it is also very difficult. The addition of Trojans may be minimized by careful code reviews and adequate checks and balances or through the use of tools such as TRUTH [17]. The TRUTH tool analyzes HDL source code pointing out potentially unsafe structures that may indicate a potential Trojan. Protecting from a design attacker requires a holistic security policy in order to minimize the risk. Our work does not deal with this type of an attacker as we assume that the IP is well protected inside the company that it is produced.

### 3.1.2 Synthesis Attacker

**Who:** A synthesis attacker usually appears well before the IP is actually synthesized. By compromising the computer aided design (CAD) tools or the scripts that run a series of CAD

tools, the attacker can modify the IP at any level from preprocessing the HDL all the way to generation of a netlist [53]. Since the attack happens inside the design house on a usually trusted platform, there is a decreased suspicion for this kind of attack. Also, since the attack occurs during synthesis, it is very difficult to discover as the logic is embedded into the design and in some cases the tests may actually verify the Trojans along with the original system. Also, with the increase in industry acceptance of open-source CAD tools, a synthesis attacker could compromise a system by hosting malicious pre-compiled binaries or direct modification of the source code. Finally, much of the process is automated using scripts. These could be modified by a design attacker through a number of different approaches. It is not a simple task to change commercial CAD tools, but a design attacker that attacked the CAD software instead of an individual IP holder would be able to compromise the IP inside the design house.

**Gained:**

- Metering - Similar to the design attacker, by stealing the IP the attacker could theoretically create extra ICs if they were willing to pay the price or had the ability to have them fabricated.
- Theft - The attacker has access to all levels of the CAD tool flow and can steal any information that exists in the IP.
- Trust - By searching for well-known patterns, the attacker can either add extra logic to the design or cripple logic such as a random number generator used in cryptography. With full access to the tool flow, there are many possibilities for attack.

**Protection:** Because of the prohibitive cost to design all of the CAD tools in-house, there is a necessary dependence on CAD software creators. A level of trust needs to be established with the creators of the CAD tools and a holistic security policy established to hinder in-house tampering of the CAD tool efforts.

### 3.1.3 Fabrication Attacker

**Who:** A fabrication attacker is usually external to the IP designer. As discussed in previous sections, much of the ICs that are produced in the world utilize contract foundries. After the IP designer has created, synthesized, placed, and routed their design, they generate a GDSII or OASIS file to express the physical layout geometry. The file is composed of a large set of polygons which describe the geometric structures of the IC. This file is then transferred to the foundry where the physical mask is created and subsequently, ICs are produced. At the same time, basic input/output specifications are supplied to which the ICs must perform. These initial functional tests can then be performed at the foundry. In this horizontal model, the foundry is given the complete design along with its specifications.

**Gained:**

- Metering - The foundry is given the design layout file from which they produce a set of masks for IC production. The Non-recurring engineering (NRE) cost of the design and the mask production are all paid by the IC designer and are the most costly items in the process. ICs are usually produced in large volumes and once production begins, creating extra ICs beyond the purchase order is inexpensive and trivial. Current IC fabrication practice does not incorporate any measures to limit the number of ICs created.

- Theft - With the layout level geometry of the entire design, reverse-engineering back to a netlist or even further to HDL is feasible, albeit difficult.

- Trust - The foundries have extended opportunity and exposure to the layout level geometry and masks they are tasked to create. Such exposure affords them the ability to add or remove components to every IC through layout geometry modification. Alternatively, after the creation of ICs, a select number of ICs may be modified using a Focused Ion Beam (FIB). This method becomes obsolete as the feature sizes shrink below 32nm, but more sophisticated techniques may allow IC modification in the future.

**Protection:** The protection against a fabrication attacker is at the core of our work. Several research avenues have focused on passive ways to identify the malevolent acts of an attacker,

our approach restricts their opportunity.

### 3.1.4 Distribution Attacker

**Who:** A distribution attacker enters the supply chain after the ICs have been fabricated. They do not have access to the HDL source code nor the layout level geometry. This type of attacker is not usually associated with either the IP designer or foundry and could either be one of the IC distributors or end users. The attacker probably does not have a set of input/output test vectors, but instead a set of specifications to which the IC is supposed to perform.

**Gained:**

- Metering - To copy the IC requires reverse-engineering the design to re-establish a netlist from which ICs can be fabricated. This is a very difficult task and more so with shrinking feature sizes.

- Theft - Information can be stolen using various methods by either deconstructing the IC or passively observing the IC through side-channel attacks. These approaches have been extensively documented, especially side-channel attacks.

- Trust - At this stage of the supply chain, it is very difficult, although not impossible, to modify the IC. Because of the processing in the previous stages, the attacker is severely limited in their freedom. The granularity of their options has also been raised and instead of being able to deal with individual gates as an attacker in previous stages could, they are forced to deal with a per package or possibly per component level granularity. Although it is technically feasible to modify individual gates at this level, the practicality of it is very low and decreases with the advent of each new feature size.

**Protection:** Besides for the difficulties imposed by small feature sizes, many schemes have been implemented in academia and commercially to address specific vulnerabilities of this type. These include anti-tampering packaging, chemical passivation and obfuscation against side-channel attacks [28].

## 3.2 Assumed Threat Model

In order to establish what we are protecting against, a threat model needs to be established. Our approach assumes the threat of a fabrication attacker who is external to both the individual IP creators and the IC designer which follows directly from the horizontal contract model that is widely popular today. Since the attacker enters the supply chain during the fabrication phase, they have access to the layout level geometry and a set of test vectors for which the fabricated IC conforms. These were both given to the foundry by the IC designer in order to facilitate fabrication of the ICs. The attacker also has significant resources temporally, fiscally and computationally, but their resources are finite. These resources include advanced technical knowledge of IC design and fabrication, access to a foundry, ability to fabricate ICs, and advanced reverse engineering tools. Since we are focused on the problem of IC metering, we assume that the attackers goal is piracy of ICs. The pirate is motivated by either profit or acquisition of specialized functionality of an IC. In both cases, the cost to pirate the IC must be outweighed by the piracy potential. In the former case, the profit potential from the sales of the counterfeit products must exceed the cost of counterfeiting and in the later case, the IC must be more valuable than simply creating the IP alone or more valuable than the cost to allocate enough resources to the functionality of a comparable IC for it to achieve comparable performance.

# CHAPTER 4. INSIGHT AND APPROACH

## 4.1 Approach Overview

The standard horizontal IC supply chain is vulnerable in the fabrication phase because the entire IC design and specifications are transferred to the foundry without control for what they will do with them. It is assumed that the foundry is good so that they will not make extra copies of the IC, steal information or add a Trojan, but there are no guarantees. We focus our efforts on hindering an attacker's ability to pirate ICs. This problem can either be solved by increasing the trust in the foundry, or decreasing their ability to pirate. In the first case, the entire IC design and specifications are still given to the foundry, but because of a contractual agreement, direct supervision of the employees that handle the IP or a system of checks and balances, the foundry is assumed to be trustworthy and therefore, even though they have the ability to pirate, they will not. This policy, although better than none, is unacceptable since it still affords the foundries the same ability to pirate ICs as they originally had. In the second case, the foundry is assumed to be bad such that the burden for anti-piracy falls not on the foundries to enforce, but upon the IC designer who can establish certain security guarantees before the IC design ever leaves their company. We chose to follow the second approach.

In our protection scheme, Figure 4.1, we transfer the control over anti-piracy to the IP designer who has the most interest in the success of the scheme and away from the possibly untrusted foundry. In order to do this, we create a scheme where the functionality of the IP, $F(x)$, is decomposed into two regions, $F_{\text{fixed}}$ and $F_{\text{reconfig}}$. The majority of the design, $F_{\text{fixed}}$, is given to the foundry to fabricate in a traditional ASIC design flow, but the reconfigurable portion, $F_{\text{reconfig}}$, of the design does not leave the IP design house. This is represented as the key in the figure and is only accessible to the IC designer. Instead of being fabricated as the

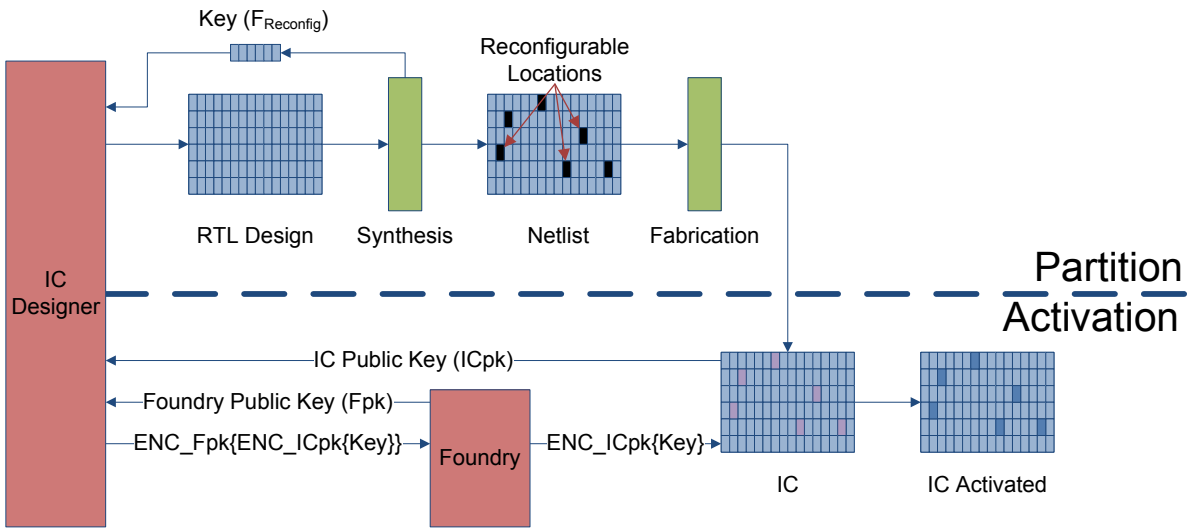



Figure 4.1: Approach overview for combating IC piracy. A subset of the logic is abstracted before fabrication. Only after the key has been securely distributed can the IC be used.

original logic, $F_{\text{reconfig}}$ is fabricated as reconfigurable logic. The withheld $F_{\text{reconfig}}$ partition can be programmed into the reconfigurable locations in the activation stage using a secure key distribution framework described in Section 4.2. The foundry is not restricted from making extra copies of the IC, but since they do not know the configuration of the reconfigurable locations, they are unable to create a functioning IC. They are also able to guess at what should be in the reconfigurable locations, but as we will show in the following sections, an educated decision on the choice of reconfigurable locations by the IC designer requires the foundry to invest more time and resources than are practical in order to discover the correct configuration. Another way to view our method is that an IC with reconfigurable locations embedded into it requires the correct key to unlock. If the correct key is not applied, the reconfigurable locations are not programmed with the correct values and the IC does not function as intended. Only when the correct key is applied can the IC be unlocked and function as expected. The key then becomes of utmost importance because it allows the IC to function correctly and is the only thing stopping the foundry from producing unsolicited ICs.

## 4.2 Unlocking Framework

Because of the importance of the key, a distribution framework needs to be established where the IP designer can securely unlock each IC separately. Our research does not create any new methods for distribution since several popular methods exist and we focus on the heuristics of key selection. The choice of frameworks is left up to the IP designer as our work is not limited to a single method. Their choice is governed by the level of security they require and the amount of risk they are willing to take in the distribution of the key. The following sections describe various topics which play a part in the key distribution.

### 4.2.1 Public-key Cryptography

Cryptography in its basic form involves transmitting plaintext messages from one party to another such that others cannot see the original plaintext message. The original plaintext message is encrypted by the sender, transmitted across an insecure medium, and then decrypted back to the plaintext by the receiver. For many thousands of years this was accomplished with a symmetric-key algorithm in which the key used to encrypt the plaintext was the same used to decrypt it. The major drawback of this system is that both the sender and the receiver need to share the same key which in turn must be securely transmitted.

Because of the many downfalls of this method, Whitfield Diffie and Martin Hellman developed public-key cryptography in 1976 [19] which was extended by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978 [52]. In public-key cryptography, the need to share a secret key is removed and replaced with a new system in which the sender and the receiver both have a public and a private key. The public keys can be shared through untrusted mediums while the private keys must remain a secret. When one party wants to transmit a plaintext message to another, they use the receivers public key to encrypt the message before transmitting it through insecure mediums. The encrypted message can only be decrypted by the private key. The security of this method comes from the use of one-way functions which make decrypting a message extremely difficult in the absence of the private key that corresponds to the public key with which the message was encrypted. It is feasible to guess the right private key with enough

time or resources, but as the size of the key grows, the difficulty is elevated. In addition to sending a message that only the owner of the private key can decrypt, the message can also be signed so that the receiver knows the origin of the message. If the sender encrypts the message with his private key and the receivers public key, then the receiver who uses their private key and the senders public key to decrypt the message knows from whom the message originated. This type of public-key cryptography is widely used today.

In the context of our metering scheme, public-key cryptography is used in order to establish secure communication channels for the transmission of the configuration used to program the reconfigurable locations. Each IC that is locked uses the same configuration for the reconfigurable locations as it is too expensive, on the order of millions of dollars, to change the masks per IC. It is also desirable to be able to meter the production on a per-IC level which implies a unique unlocking key for each IC. For this to work effectively, each IC needs to be able to generate a unique random public and private key pair in order to be able to participate in secure communications. On one hand the goal of IC fabrication is to minimize variations and produce identical ICs. On the other hand they need to be unique so that they can be metered. Years of tuning the IC fabrication process provides the first criteria and hardware true random number generators or physical unclonable functions (PUFs) can be utilized to fulfill the second.

### 4.2.2 IC Identification

In order for the IC to generate a unique random public and private key pair, each IC must distinguish itself in some way. Although the IC process attempts to fabricate each IC identically, manufacturing variations occur which cause the ICs to be slightly different even on the same die. While these variations sometimes cause the ICs to be different enough to merit separation as in the case of CPU binning, they usually are within specifications for the design. But, these slight differences can be exploited as a means of identification. The first step in being able to generate a set of keys is to be able to establish uniqueness at a per-IC level. The process by which the unique qualities of an IC are extracted is usually referred to as a Physical Random Function or Physical Uncloneable Function (PUF). The method by which

the uniqueness is realized is what differentiates the following approaches.

In [43], the authors observe that the impurity dopants found in silicon cause MOSFETS to have slight variances in drain currents. By using an addressable array of MOSFETS, an identification system can be created. In a similar manner, the authors of [44] utilized the variations in the grains of polycrystalline silicon thin-film transistors to identify an IC. The slight differences in the grain boundaries of the polycrystalline silicon cause slight current differences which are exploited to determine uniqueness. In [24, 38], various gate level structures are created to compare signals along various paths. A specialized PUF described in [32, 61] exercises the instability of a cross-coupled inverter, similar to those used in the creation of flip-flops and SRAM cells. The variations cause the PUF to tend to one of the two stable operating points. By adding a series of these structures, a unique identifier can be created. It should be noted that not all manufacturing variabilites provide adequate security guarantees as shown in [45].

Even though exploiting the random characteristics of manufacturing can produce a unique per-IC identification, it still has not produced a suitable key for public-key cryptography. In [21, 42], the authors addresses this issue by adding error correction units to increase the consistency of the output from the PUF. Specifically geared toward ICs, [64, 66] perform the correction operations and use the results to feed a key generation unit that is seeded by the PUF.

### 4.2.3 Key Distribution Schemes

The key distribution is flexible in what is to be communicated, with whom, the minimum security guarantees, and maximum resource budget. Some possibilities have been proposed in [27, 54] but the ultimate choice is left to the user. The distribution scheme in [54] uses three key parties, the IP holder, the foundry, and the individual IC. Each one of these must generate a public and private key pair; the IC through the methods described in Section 4.2.2 and the other two using well-known software approaches. The public key is shared with all parties and then the IP holder can encrypt the correct configuration of the reconfigurable locations with IP holder's private key, the ICs public key, and the foundries public key. This encrypted message

is transmitted to the foundry who is the only one that can receive the message by using their private key. The result is sent to the intended IC which can decrypt the message using the ICs private key and the IP holders public key. The final result is the plaintext configuration that will unlock the IC. It should be noted that the plaintext configuration is the same for all of the ICs fabricated from the same mask, but the transmission from the IP holder is encrypted with a specific ICs public key. This allows the IP holder to maintain control of which ICs are being activated.

In [27], the authors start with a different focus and trust model while still utilizing a key distribution scheme. Instead of withholding part of the design, the authors hardcode the configuration on the IC, but require a certain condition before the configuration is activated. This is comparable to the FSM locking seen in Section 2. The public key of the IP holder is also hardcoded on the IC. A user who wishes to be authenticated in order to use the IC must generate a secret key known only to themselves. This secret key is taken as an input to a PUF on the IC which generates a random value that is unique to both the IC and the input key. This means that changing either the IC or the key will change the value generated by the PUF. The IC can then encrypt the value with the public key of the IP holder so that only the IP holder knows the value generated by the PUF. The user must then be authenticated with the IP holder and establish a secure communication channel using any of the numerous well-defined methods. The IP holder can then transmit the output of the PUF back to the user of the IC to be used as a second input to the IC. When the IC verifies the input value and the value generated from the PUF match, the IC can be unlocked and the correct configuration loaded. This mechanism gives the IP holder control over authenticating a user and authenticating which IC that user can use.

Various key distribution schemes can be constructed which, at the core, deal with how to securely unlock a specific IC so that it can be metered. But, every distribution scheme becomes worthless if an attacker can easily generate the unlocking pattern themselves. It does not matter how well protected the transmission of a secret is if the secret can be discovered independently of the transmission. Because of this reasoning we focused our efforts on trying

to increase the security of the reconfiguration independent of any distribution model while at the same time minimizing the overhead associated with this type of locking mechanism

## 4.3 Key Creation

The previous combinational locking schemes [27, 54] both used 2-input XOR and XNOR gates as their primary gate abstractions. The gates were inserted on wires throughout the design with one of the inputs a single bit of the key and the other, the original wire value the gate interrupted. Their choice of wires for gate placement was random with a small provision for avoiding the critical path. Although this does yield some level of security, we propose a method with better security and minimized overhead. To meet this end, the level of gate abstraction is raised from the single XOR and XNOR gates to a k-input LUT. The selection criteria is also improved by creating a heuristic algorithm focused on selecting the best set of nodes.

### 4.3.1 Gate Abstraction

The choice to use LUTs instead of XOR gates was based on a couple of factors. First, even though the XOR scheme effectively locks the IC, the entire design is still present in the IC. The XOR gates serve to cripple the design by acting as inverters when incorrectly guessed key bits are used. In the case of the LUT, not only is a locking mechanism provided, but only a partial design is exposed to the foundry. Where the XOR schemes use extra logic to lock the IC, our approach uses part of the original design as the key. As the feature size shrinks reverse engineering a design from a fabricated IC is increasingly difficult, but it is still very viable from a layout level geometry. By only using XOR gates, it is possible to extract the circuit netlist and then simply remove the XOR gates. It is also possible to use a process such as FIBing where additional wires are added to bypass the XOR lock. Although it is not trivial to carry out either of those approaches, we prefer to not leave it as an option. The LUT based approach does not use additional logic, but instead replaces some of the original logic which forces pirates to exercise the key space in order to unlock the IC.

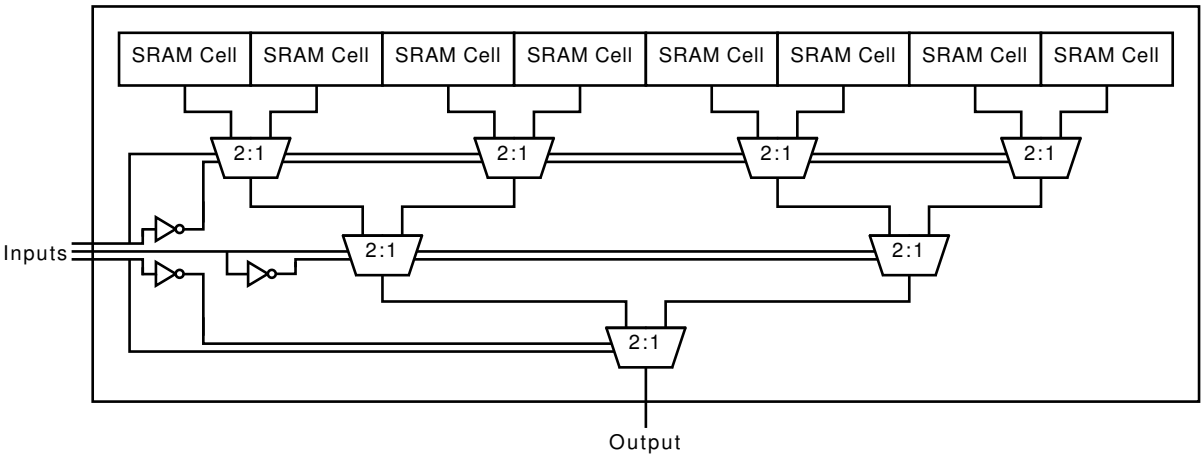



Figure 4.2: Schematic of a 3-LUT implemented using SRAM cells and 2:1 muxes

#### 4.3.1.1 Overhead

The overhead that is associated with our approach must be taken into consideration. The overhead comes in various forms including area, power, and delay where each is dependent on the number of transistors required to implement the approach.

A LUT can be implemented as an array of SRAM cells with a MUX tree to select the particular array location. In our case, we are always dealing with a k-input and 1 output function. Since a k-LUT has $2^k$ possible combinations of inputs, it also requires $2^k$ SRAM cells to store a 1-bit output for each combination. A mux must also accompany the SRAM cells in order to choose the appropriate output. This can be accomplished with a multi-level 2:1 mux tree or $2^k - 1$ 2:1 muxes for each k-LUT. Finally, $k$ inverters are needed to generate the inversion of each input signal. Each of the SRAM cells require 6 transistors, the 2:1 mux, 4 transistors, and the inverters, 2 transistors. In total, a k-LUT requires $10 \times 2^k + 2 \times k - 4$ transistors to be implemented. The XOR scheme requires 2 transistors per XOR, but there are many more XOR gates than LUTs and the XOR gates are in addition to the original logic while the LUTs implement part of the original logic.

Since the LUT is assuming some of the logic inside its functionality, the transistor count overhead must take into consideration the logic it is overlapping. The amount of logic that the LUT represents is dependent on the mapping algorithms and how the logic was partitioned.

The LUT can take the place of any k-input, one output function which may be as simple as a set of wires or a more complex Boolean expression. Because of this, it is difficult to say exactly how many transistors are replaced by the LUT. The overhead of a single LUT compared to the original design is can be expressed as

$$(10 \times 2^k + 2 \times k - 4) - (\text{LUT Function Transistor Count}) \tag{4.1}$$

Also, since the XOR gates in the XOR scheme are strictly overhead, the LUT based approach requires

$$\left(\left(\left(10 \times 2^k + 2 \times k - 4\right) - (\text{LUT Function Transistor Count})\right) \times \frac{n}{2^k}\right) - 2 \times n \tag{4.2}$$

more transistors than the XOR approach. In this equation, $k$ is the size of the LUT and $n$ is the number of key bits required. In order to make a comparison of the transistor overhead, we assume that the LUT does not replace any functionality and is therefore completely overhead. This is not the case in practice and would negate the logic exclusion property mentioned in Section 4.3.1, but it does give a general comparison. For this case, comparing the LUT based approach to the XOR approach while keeping the key bits the same for each shows that the LUTs use approximately 5 times the number of transistors for all configurations of key bits and k-input LUTs.

Area scales approximately linearly with transistor count. Looking at Equation 4.1 and Equation 4.2 the amount of extra area required to implement the design can be calculated approximately by adding the same percentage of area to the original design as the percentage of transistors increased from these equations. The area overhead is often not the critical factor in ASIC design and its cost must be weighed against the security benefits of the approach.

The delay overhead is how much the delay from the inputs to the outputs is lengthened. Since a LUT is an array of SRAM cells followed by a MUX tree, the longest path is through an inverter and the mux tree. For a $k$-input LUT, the height of the MUX tree is $log_2(SRAMCells)$ and there are $2^k$ SRAM cells so the longest path is $k+1$. For the XOR gate, the delay is always a constant value of 1. But, just like the transistor count, the delay must take into account the logic that the LUT has absorbed. It is also difficult to compare the two approaches since there

is no guarantee that the delays are all independent from the others and instead my accumulate on certain paths. Our approach does minimize this by selecting nodes along a cut instead of randomly, but if the budget is large, more than one cut may be used. Like the transistor comparisons, by assuming the worst case delay through the LUT and no logic is replaced by the LUT, the LUT based approach has $\frac{n}{k}$ more unit delays than the XOR based approach for the same number of key bits.

Both static and dynamic power combine to form the overall power requirement of a circuit. Static power is a function of transistor count which was calculated in Equation 4.1 and Equation 4.2. Dynamic power is proportional to the product of the number of switching transistors and the switching rate. Since the switching rate remains the same, it is dependent on the number of switching transistors. It is difficult to fully compare the two approaches because the statistical probability that the transistors switch is application dependent, but a generalized comparison can be made. The static power will be larger in the LUT based approach since there are more transistors. The dynamic power will also be larger in the case of the LUTs since there are more transistors and they will switch more frequently than the XORs. Both approaches, however, are configured relatively infrequently, so the power required for configuration is negligible.

### 4.3.2 Observability and Controllability

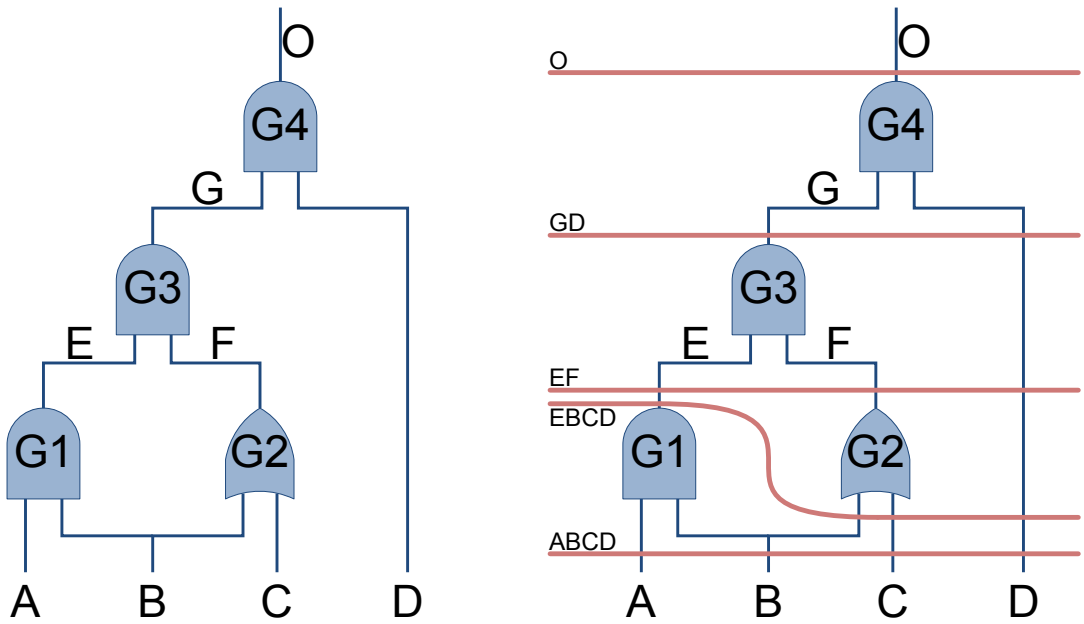
Observability and controllability are two inherent metrics of a logic network and are used as part of our selection heuristic. A logic network is a acyclic directed graph, $\mathcal{N} = \{\mathcal{V}, \mathcal{E}\}$, where each node, $v_i \in \mathcal{V}$, represents a Boolean function with multiple inputs and a single output. The edges, $\mathcal{E}$, connect the nodes beginning with the primary inputs and flowing towards the primary outputs. Significant research has been done on logic networks resulting in various representations and manipulation operations. In the ASIC design flow, the logic network representing the HDL description undergoes various optimization steps before it is mapped into primitive gates that cover all of the functionality that the original logic network encompassed in the nodes. At one extreme we can view a logic network as a single node

connected to all of the primary inputs and outputs with one function to represent the entire circuit. In the other extreme, the single node is broken into many interconnections of nodes where each node only represents a primitive function.

During logic synthesis, many optimization steps are performed on the logic network. These steps require analysis of the logic network in order to discover dimensions of flexibility to perform the optimizations. One such avenue relies on controllability and observability analysis to determine a set of don't care conditions, $\boldsymbol{DC}$. Informally, it describes a set of inputs or outputs which do not matter to the circuit because the nature of how the nodes of $\mathcal{V}$ are connected. More formally, they are described as follows.

Controllability is a measure of how easy it is to control the inputs of a given node or how easy it is to generate a specific set of inputs for that node. The set of don't care conditions for a node can be defined by including all input patterns that are never produced at that node. These input patterns form the controllability don't care set, $\boldsymbol{CDC_{in}}$. This means that the more controllability don't care conditions a node has, the harder it is to control. If a node only had primary inputs connected to it, then it would be very controllable since every primary input pattern can be exercised for a circuit and therefore every combination of inputs for the node. But, a node buried in the logic network may have an input combination that is never seen even if every combination of primary inputs was exercised. Take for example the circuit in Figure 4.3a. The primary inputs, $A$, $B$, $C$ and $D$ can be exercised with all $2^4$ combinations and the rest of the circuit responds accordingly. Both gates $G1$ and $G2$ are fully controllable because all four of their input combinations can be exercised. $G3$, however, is not fully controllable because it is not possible to exercise all of the combinations of $E$ and $F$. This can be seen when $E$ is true, $F$ can never be false because if $B$ is true to make $E$ true, it cannot also be false to make $F$ false. This means that $E\overline{F}$ is a don't care condition for $G3$ since it will never occur in this circuit.

Observability is a measure of how easy it is to see a particular set of values on the output. In the same manner as the controllability, the observability is often expressed by a set of don't care conditions or sets of values that describe conditions where the output is never seen,




(a) Original logic network

(b) Logic network showing cuts

Figure 4.3: Example logic network

$\boldsymbol{ODC_{out}}$. If a node had only primary outputs, the node would be very observable since every output that the node generated would be observable on the primary outputs. But, if the output of the node did not affect any other node, that node would not be very observable. By looking at Figure 4.3a, the observability don't care set of gate $G3$ can be calculated. If either $G$ or $D$ are false, the output of $G4$ is also false. This means that if $D$ is false, it does not matter what $G$ is because the output of $G4$ is fixed. In other words, if $D$ is false, then the output of $G3$, $G$, is not observable. Therefore, $\overline{D}$ is a don't care condition for $G3$ since $G3$ is never observed when that condition is true. In terms of the primary inputs it is: $\boldsymbol{G3_{ODC}} = ABC\overline{D}, AB\overline{CD}, A\overline{B}C\overline{D}, A\overline{BCD}, \overline{A}BC\overline{D}, \overline{A}B\overline{CD}, \overline{AB}C\overline{D}, \overline{ABCD}$. The entire set of external don't care conditions is formed by joining both the controllability and observability don't care sets, $\boldsymbol{DC_{ext}} = \boldsymbol{CDC_{in}} \cup \boldsymbol{ODC_{out}}$.

### 4.3.3 Cut Based Approach

A cut is a partitioning of the logic network into two separate parts such that every path from a primary input to a primary output must pass through the cut. Each logic network has many possible cuts and as the logic network grows large, the number of possible cuts grows prohibitively large even for moderately sized networks. Since the cut separates sets of nodes, it can be described by either the nodes on the input side or the nodes on the output side. We choose to describe a cut in terms of the nodes on the input side or in other words the nodes whose outputs go through the cut. Cuts are used in many of the logic synthesis algorithms and we also use them in our heuristics. Factoring in cuts as part of the heuristic allows us to choose nodes that are not just chosen by a greedy algorithm, but have a global outlook. The top nodes independently ranked by their observability may not be the best set to choose. Instead, by looking at how various cuts affect the outputs, we are optimizing at a group level in place of an individual node level. Because complete enumeration of the cuts is not possible as the logic network grows large, we traverse the network making cuts while keeping the height of the cut approximately even across the entire network. The first cut enumerated contains only the primary inputs which can be seen in Figure 4.3b. This cut is labeled by the names of the wires on the input side through which it cuts, $ABCD$. The second cut, $EBCD$, grows the original cut by one node and then removes any wires which it no longer cuts through. Since B is used twice, as inputs to $G1$ and $G2$, it still remains in the second cut even though it was dropped from the $G1$ gate. The cuts continue to advance by one node until the final cut is made of the primary outputs.

### 4.3.4 Height

The logic network representing the IC is a directed acyclic graph, $\mathcal{N} = \{\mathcal{V}, \mathcal{E}\}$. Each node, $v_i \in \mathcal{V}$, has an associated height value that corresponds to its position in $\mathcal{N}$. The height of $v_i$ is the length of the shortest simple chain, $< x_0, x_1, \cdots, x_n >$: $\forall\, i \in [0 \cdots N-1]$ : $x_i$ is adjacent to $x_{i+1}$, starting at one of the primary inputs. This means that the primary inputs all assume a height value of 0 and one or more of the primary outputs the maximum height

value. A cut can also be assigned a height value. For our selection heuristics, we calculate this value by averaging the height of each of the nodes that comprise the cut.

### 4.3.5 Budget

The addition of LUTs to replace some of the original logic incurs an overhead while providing a security benefit. As LUTs are added, the performance, area, power, and delay penalty increase, but so does the security. This can be taken to the extreme in the case of an FPGA where the performance is poor, but the foundry that fabricates the FPGA does not know what will be implemented on the FPGA and should not. In order to balance these two conflicting metrics, a budget needs to be defined which allows maximizing a metric within the budget of the other. The budget is configurable in order to be adapted for different applications that require either more performance with a weaker security or a very secure system at the cost of performance. The heuristic uses either the amount of security needed or the amount of performance willing to sacrifice.

### 4.3.6 Selection Heuristic

When choosing locations for insertion of the reconfigurable regions which represent $F_{\text{reconfig}}$, we want to pick locations that minimize the possibility of the attacker bypassing the regions or guessing the correct configuration. For our selection heuristic, we use a combination of the ***ODC*** and cut height with the constraint that the nodes are on the same cut in order to choose the best nodes.

As mentioned earlier, a cut-based approach was used. Logic network cuts completely partition the network into two parts so that every path from a primary input must go through the cut in order to get to the primary output. Since no path exists that does not go through a cut, the data must flow through our reconfigurable regions and in doing so be affected by the configuration of the LUT. If an incorrect LUT configuration is assigned to a LUT, the output will be affected causing a chain reaction throughout the rest of the network that eventually causes the outputs to be incorrect. Abstracting all nodes along the cut causes a cumulative

effect on the outputs with incorrect LUT configurations.

We utilize ***ODC*** because intuitively if $F_{\text{reconfig}}$ is important, any incorrect configurations render the circuit more functionally incorrect than a less important region and the region is necessary in order for the circuit to function correctly. When selecting $F_{\text{reconfig}}$, it is desirable to choose nodes that maximize the control by being highly observable over the outputs, $(y_0, y_1, \ldots, y_{N-1})$. This means that a change in the node will affect the output more than a less observable node. This property is useful since we are trying to maximize the difficulty for an attacker who is trying to figure out the correct configuration of the LUTs. If an attacker were to configure a more observable LUT with an incorrect configuration, the circuit would have more of the outputs incorrect when compared to a less observable LUT. In the extreme cases, if the node was not observable at all, it would not matter what the LUT configuration was because it would never affect the primary outputs. In the same way, if the LUT was completely observable, any change in the configuration would corrupt some of the output bits. The other property of highly observable nodes, especially when their observability is spread over many outputs, is that by guessing a LUT configuration, less information is gained as small configuration changes lead to large output changes.

Formally, for the logic network $\mathcal{N} = \{\mathcal{V}, \mathcal{E}\}$ corresponding to a function $f(x_0, x_1, \ldots, x_{n-1}) = (y_0, y_1, \ldots, y_{N-1})$ with the desired threshold of reconfigurable logic, $0 <= th_{\text{reconfig}} < 1$, a subset $\mathcal{V}_{\text{reconfig}} \subseteq \mathcal{V}$ is chosen such that $\frac{|\mathcal{V}_{\text{reconfig}}|}{|\mathcal{V}|} <= th_{\text{reconfig}}$ and the observability of the sub-units $G_i$ in $f$, $\sum_{G_i \in \mathcal{V}_{\text{reconfig}}} \sum_{j=0}^{N-1} min_{(x_0, x_1, \ldots, x_{N-1})} \left(\frac{\partial y_j}{\partial G_i}\right)$, is maximized.

We utilize height as a metric for our selection criteria. Specifically cuts that are located nearest to center of the logic network are better than cuts that deviate away from the center either towards the primary input or primary outputs. The reasoning for this choice can best be seen by an example logic network. Figure 4.4 shows the placement of LUTs for three different cuts. For the sake of simplicity only one LUT of a given cut is shown and examined. These three LUT placements, A, B, and C, are located at the primary outputs, primary inputs and in the middle respectively. The circuit depicted is also overly simplified as many inputs and output are replaced by a series of black dots. Much of the network is not shown and instead, dashed

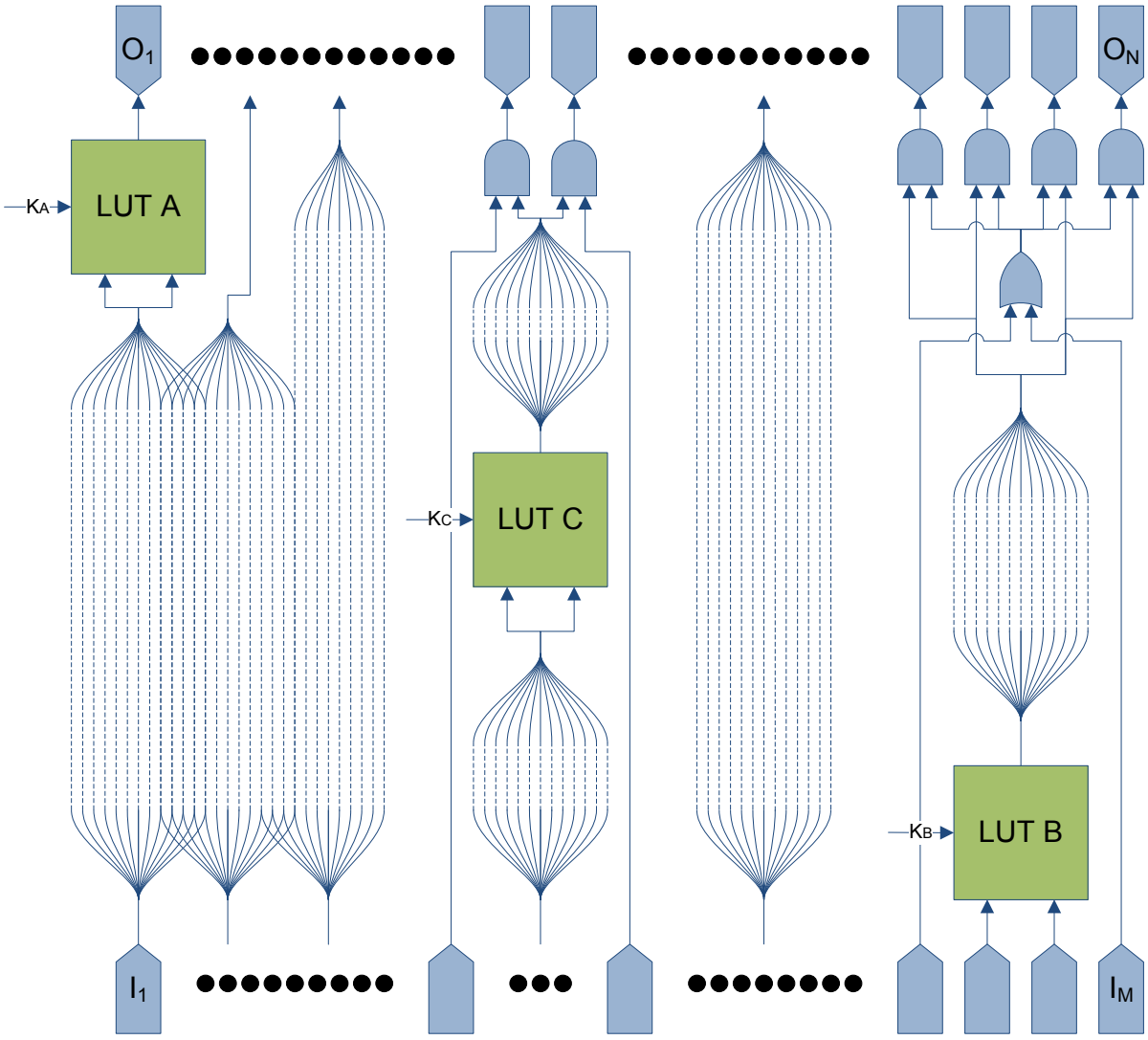



Figure 4.4: Logic network showing three cut choices at different levels of the network. LUT A connected to the primary outputs, LUT B connected to the primary inputs and LUT C in the middle of the network

lines show the information flowing from the primary inputs towards the primary outputs.

Each of the LUTs have the same ***ODC*** score which is the sum of the ***ODC*** sets per output. Since LUT A is connected directly to $O_1$, there is nothing in the ***ODC*** set for this output. For the other outputs, it is not connected so each have the entire input space as their ***ODC*** set or $2^M$ minterms where $M$ is the number of inputs. Therefore, the total score for LUT A is $2^M \times (N-1) + 2^M \times 0 = 2^M \times (N-1)$ where $N$ is the number of outputs. LUT C affects two outputs which are guarded by AND gates connected to primary inputs. This means that for all of the minterms, half of the time the value of the AND gates will be zero which does not allow the output of LUT C to be observed. This means that the total score for LUT C is $2^M \times (N-2) + \frac{1}{2} \times 2^M + \frac{1}{2} \times 2^M = 2^M \times (N-1)$. The third LUT, LUT B, affects four outputs, but because of the gates connected to the primary outputs, the total score for this LUT is $2^M \times (N-4) + \frac{3}{4} \times 2^M + \frac{3}{4} \times 2^M + \frac{3}{4} \times 2^M + \frac{3}{4} \times 2^M = 2^M \times (N-1)$. It should be noted that for LUT B and LUT C, the ***ODC*** set may be larger depending on what the logic is in the regions not shown, but for simplicity we assume that the logic does not change the ***ODC*** set. A selection heuristic that only focused on the number of minterms in ***ODC*** would have ranked each of the cuts as equal and would have chosen one of the three LUT configurations with the same probability, but these choices are not equivalent.

Looking at the LUTs again in terms of controllability and observability, we see that LUT A is completely observable on $O_1$ and a cut abstracting the final level of logic would contain LUTs which all were very observable on their respective outputs. Although not strictly true, the controllability of LUT A is less than other nodes closer to the primary inputs and certainly not any more controllable than LUT B. We are looking for nodes that are not very controllable considering our threat model. The attacker is targeting individual LUTs by exercising input combinations while using different key ($K_A$) combinations in order to see which configuration leads to the greatest amount of information learned. If the LUT is very controllable, it is easier to exercise all of the input combinations of the LUT where as if the LUT is not very controllable, many more primary input combinations must be exercised before the inputs of the LUT are all exercised. Switching now to LUT B, we see that it is highly controllable since the primary

inputs are connected directly to the inputs of the LUT. But, it has the same observability score as the other two LUTs. In general, although not always the case, the nodes located closer to the inputs have a larger observability score since there are more don't care conditions. LUT C, located in the middle of the circuit is neither completely observable for a single output, nor completely controllable.

We argue that nodes located near the middle of the logic network are better choices that do not allow the attacker to gain as much information about the circuit as nodes located near one of the other extremes. In the case of Figure 4.4, each of the three LUTs has the same observability score, but for most circuits, this would not be the case. The first metric in consideration of the selection heuristic is the number of ***ODC*** minterms, but this metric alone is not sufficient for determining the best choice. Intuitively, selecting a cut that has fewer ***ODC*** minterms since it is highly observable located at the last level of logic does not make a good partition of the circuit. On the inputs side, almost the entire circuit exists in its original form allowing the attacker to concentrate on the last level of logic. By cutting the circuit in the middle, a more interesting partition is made, allowing both the input and output side to contain approximately the same amount of the logic network. In these cases, even with the same number of ***ODC*** minterms, the effects are spread over many outputs and not concentrated on a single output making the attacker's ability to choose the correct configuration much more difficult. Similarly, choosing a node located at the primary inputs is not very wise since the attacker can easily control that LUTs inputs and does not have to search many more input combinations in order to exercise the inputs, LUT C for instance. For these reasons, we choose to figure the height of a cut into our selection heuristic.

In order to properly mix the ***ODC*** minterm score with the height value, a parameter $\alpha$ was introduced which tuned the cut selection's dependence on one of the metrics. With an $\alpha$ value of 0 only the ***ODC*** score was taken into consideration and when the $\alpha$ value was set to 1, only the height was considered.

Description of algorithm for choosing nodes.

**Algorithm 1** Selection Heuristic

**for all** $n_i \in nodes$ **do**
  Calculate ODC set of $n_i$
  Calculate height of $n_i$
**end for**
**for all** $n_i \in |nodes|$ in topological order **do**
  Add $n_i$ to cut
  Trim children of $n_i$
  CutScore$[n_i] = \alpha \times \left|\frac{maxHeight - 2 \times cutHeight}{maxHeight}\right| + (1 - \alpha) \times \left(\frac{cutODCSum}{maxODCSum}\right)$
**end for**
Select best cut $C$
**for all** $n_i \in C$ **do**
  Tag $n_i$
**end for**

## 4.4 Implementation

The overall tool flow for implementing our heuristics can be seen in Figure 4.5. The blue (medium gray) boxes represent unmodified steps in the traditional tool flow. The boxes shaded green (light gray) represent the additional steps integrated into a standard tool flow. On the left side of the diagram, the IC designer creates the entire design as a RTL description. The logic synthesis begins with the logic optimization step which takes in the RTL and also optimization goals. These goals vary significantly from design to design depending on what metric or set of metrics the designer desires. These may include optimizations for area, delay, and power. The logic optimization stage is very involved and is implemented in varying degrees in all logic synthesis software. After all of the optimizations are finished, the original flow would move on to mapping, but in our modified flow, this is where we begin.

The first step is to calculate the ***ODC*** values for all of the nodes in the circuit. As was discussed in Section 4.3.2, the ***ODC*** set allows us to determine the global significance of an internal node at a per-output level. The height is calculated by a tree traversal and represents the shortest number of nodes required to pass through in order to reach a primary input. After the ***ODC*** calculation is finished, each node stores this information. The cut calculation follows the same algorithm that was discussed in Section 4.3.3. By iterating over the nodes in

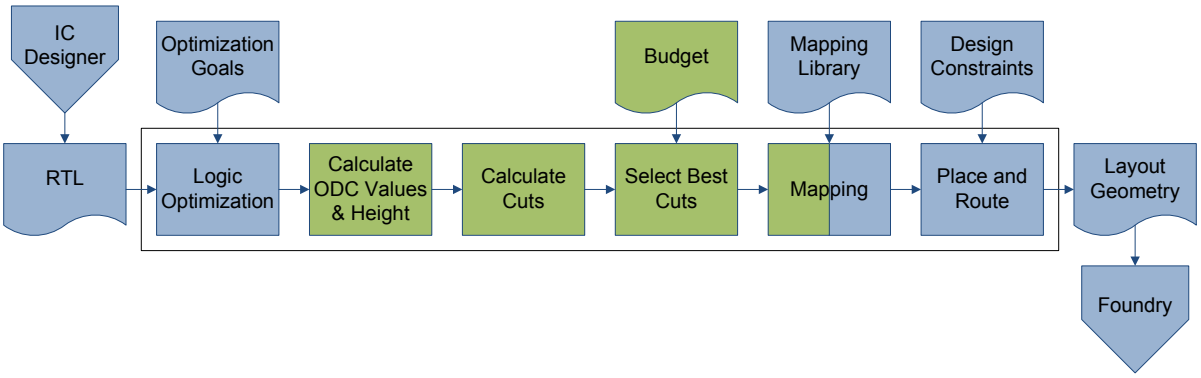



Figure 4.5: Tool flow used to create layout geometry from RTL description. Blue (medium gray) boxes represent standard tool flow. Green (light gray) boxes represent modifications made by this work.

the network and using the $\boldsymbol{ODC}$ values for every node along the cut, a measurement can be determined for each cut. This cut metric is the sum of the number of $\boldsymbol{ODC}$ values for each output of the nodes along the cut. A second metric, the average cut height, is calculated by averaging the heights of each node in the cut. The next stage ranks all of the cuts according to the cuts total $\boldsymbol{ODC}$ value and the average height value calculated in the previous step. These two metrics are weighted with the $\alpha$ parameter to determine the influence of each metric on the cut score. When $\alpha = 0$ only the $\boldsymbol{ODC}$ score is considered and when $\alpha = 1$ only the height is considered. All other values of $\alpha$ proportionally weight the final cut score. By using the budget to determine how many nodes can be selected, the top cuts are selected and tagged.

The final step in the modified process overlaps the original mapping stage. Since the tagged nodes must be mapped to LUTs and not to the ASIC gates which their function corresponds, the mapping phase has been augmented to map each of the tagged nodes to LUTs. The rest of the nodes are mapped according to the functionality found in the mapping library. Once the mapping phase completes, a netlist is generated. This is not necessary for fabrication of the IC, but allows us to run the simulations shown in Section 5. From this point, the process can continue as usual where the design is placed and routed according to the design constraints and finally, a layout geometry file is generated. The foundry can then use this file to fabricate the ICs.

# CHAPTER 5. EXPERIMENTAL ANALYSIS

This section provides the experimental results obtained from implementing the theoretical ideas of the previous sections. In order to evaluate the effectiveness of our approach, various experiments were conducted. These include an examination of how a cut-based selection heuristic compared to random selection, the effectiveness of a single cut, and a discussion of how the parameters of the heuristic can be tuned for increased performance. The results of the experiments are favorable and show that using our tuned selection heuristic on a variety of professional benchmarks is more resilient to attacks than previous methods while minimizing the amount of overhead induced. They also highlight the feasibility of integrating security features into the IC design flow aimed at decreasing IC piracy.

## 5.1 Benchmarks

All of the benchmarks used in the experiments come from the Microelectronics Center of North Carolina (MCNC) benchmark suite [74] developed for the International Workshop on Logic Synthesis. These benchmarks have been widely used throughout the logic synthesis literature with the technical report describing them cited over 400 times. Out of the benchmarks, we utilized the combinational multi-level subset, Table 5.1, since our tool flow does not support sequential circuits and they provided logic-heavy circuits. Benchmarks with very repeatable and predictable patterns such as memory cells or multipliers are not well suited for our approach. In these benchmarks, replacing functionality with LUTs does not hide information as the predicable pattern or ordinary structure already indicates what was abstracted by LUTs. Also, benchmarks with fewer than 100 nodes were not used because the small node count did not provide enough information for interesting results to be generated and smaller circuits are

Table 5.1: Circuit characteristics of the MCNC benchmarks used

| Benchmark | Inputs | Outputs | Height | Total Nodes |
|---|---|---|---|---|
| apex2 | 39 | 3 | 14 | 942 |
| apex4 | 9 | 18 | 11 | 722 |
| C432 | 36 | 7 | 17 | 160 |
| C1355 | 41 | 32 | 23 | 514 |
| C1908 | 33 | 25 | 40 | 880 |
| C2670 | 233 | 139 | 32 | 1161 |
| dalu | 75 | 16 | 24 | 1131 |
| des | 256 | 245 | 13 | 1498 |
| ex5p | 8 | 63 | 12 | 527 |
| ex1010 | 10 | 10 | 11 | 850 |
| i4 | 192 | 6 | 4 | 101 |
| i7 | 199 | 67 | 3 | 406 |
| i8 | 133 | 81 | 8 | 1183 |
| i9 | 88 | 63 | 7 | 353 |
| k2 | 45 | 43 | 2 | 225 |
| seq | 41 | 35 | 11 | 1020 |
| Average | 90 | 53 | 15 | 729 |

not likely candidates for full ASIC design. For these reasons, twenty logic-heavy combinational multi-level benchmarks circuits from the MCNC benchmark suite were chosen. Table 5.1 describes the benchmarks listing the number of inputs and outputs for each benchmark, the total nodes in the circuit, the number of nodes that were chosen in a single cut using the ***ODC*** heuristic, and the number of nodes chosen by the height heuristic.

The benchmark files are all in the Berkeley Logic Interchange Format (BLIF) which is a standard logic synthesis description format and can be generated and interpreted by many common CAD tools. A BLIF file is a textual representation of a directed graph with named nodes being assigned single output functions. These nodes are wired together by assigning the output of one node to the input of another. Finally, the inputs and outputs of the entire circuit are enumerated and connected to the internal nodes.

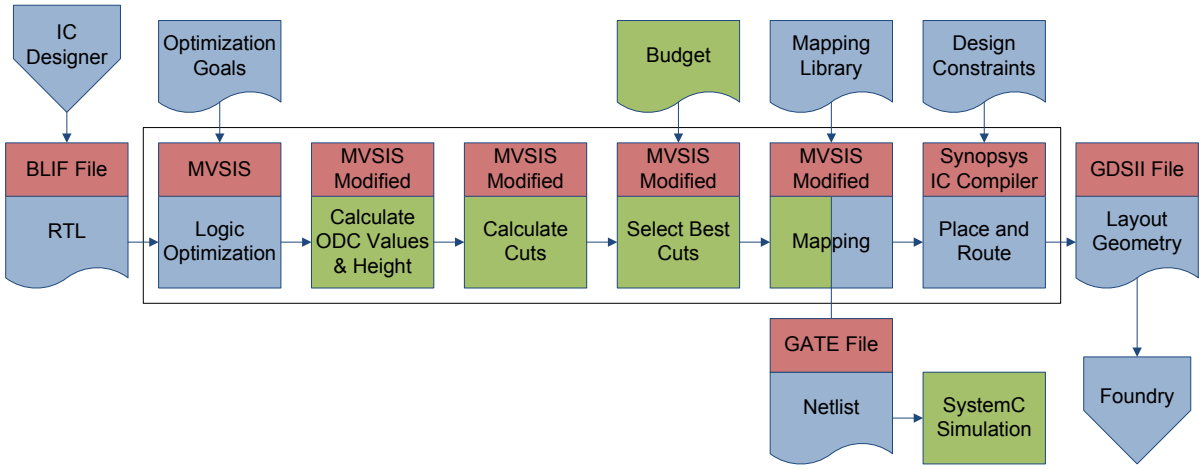



Figure 5.1: Tool flow used to create layout geometry from RTL description. Blue (medium gray) boxes represent standard tool flow. Green (light gray) boxes represent modifications made by this work. Red (dark gray) boxes represent the specific tool or file used.

## 5.2 Experimental Setup

Following along with the tool flow in Figure 5.1, the blue (medium gray) stages are the original flow, the green (light gray) stages are the custom steps added, and the red boxes (dark gray) are the specific tool that implements the stage. The logic synthesis was provided by MVSIS and the results feed into three custom steps added to the source code of MVSIS. These include, calculating the ODC and height values of each node, calculating the cuts, and selecting the best cuts which are tagged for mapping. Synopsys IC Compiler was used for placing and routing the circuit in order to generate a layout geometry. In parallel, SystemC simulations were done on the circuit, post-mapping, in order to conduct the experiments.

MVSIS, [23], is a popular open-source academic tool produced by the University of California Berkeley used to perform logic synthesis and mapping. MVSIS is an extension to SIS, [57], also developed by Berkeley, that adds support for multi-valued logic synthesis. The MVSIS tool reads BLIF format benchmarks and performs a user specified set of optimizations to the logic network. The optimized network can then be mapped and written to a netlist format. Our work augments these steps with separate passes over the logic network in order to identify and tag the best nodes according to the cut-based heuristic in Section 4.3.6. The functional

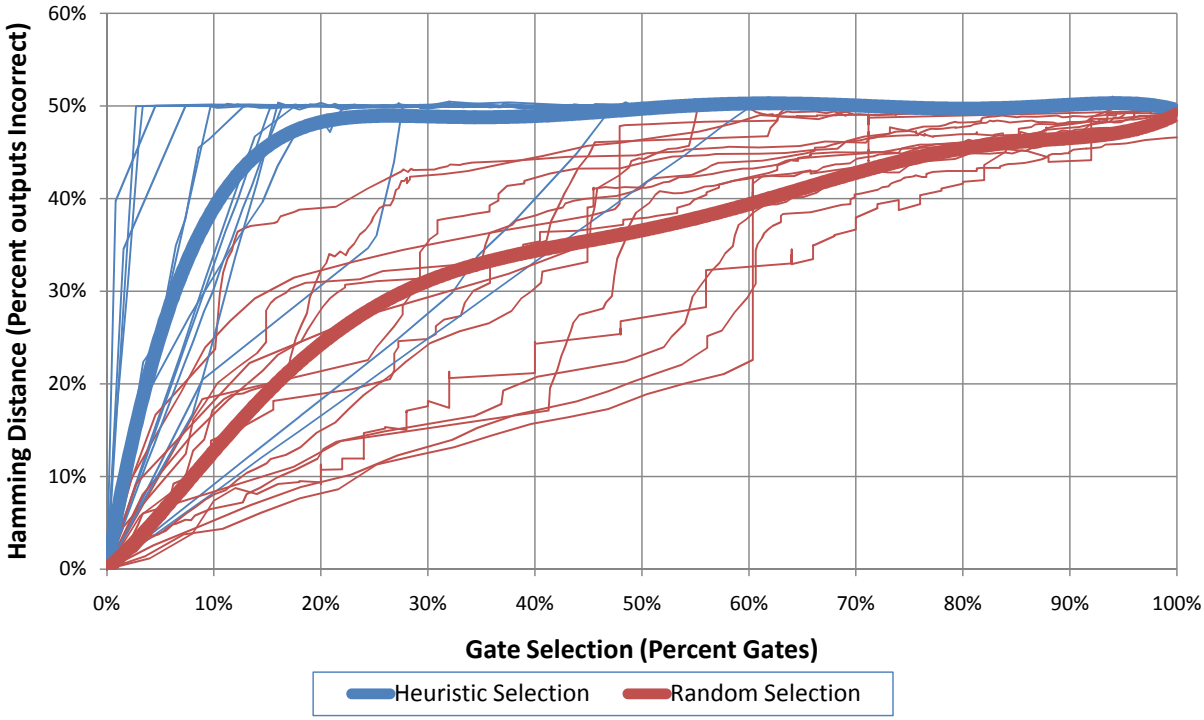



Figure 5.2: Heuristic selection vs. random selection for all benchmarks. The thin lines represent individual benchmarks and the thicker line the average of either the heuristic selection in blue (medium gray) or the random selection in red (dark gray).

netlists produced post-mapping were simulated using SystemC [29]. SystemC is a system description language based on C++ used to model and simulate the IC. By exercising billions of simulation test vectors, the changes that occurred under various LUT configurations could be observed. Both the generation of netlists and the simulation process were all automated such that a set of benchmarks could be specified, netlists generated for various parameters, each netlist simulated, the results recorded, and graphs plotted.

## 5.3 Experiments

### 5.3.1 Heuristic Selection vs. Random Selection

In previous works, [27, 54], nodes were chosen at random to be replaced by LUTs. Our work uses a selection heuristic in order to select the nodes for replacement and in doing so, is more effective. In this experiment, the effects of random selection and heuristic selection

are compared. As a metric for performance, we consider how closely the outputs of a circuit match the expected outputs, their hamming distance. The hamming distance is measured by the number of outputs that are incorrect, but has been normalized to the percentage of outputs incorrect in order to compare benchmarks. Therefore, if the hamming distance is 0% then for every input set, the outputs match the original circuit and conversely if the hamming distance is 100%, then all of the outputs are inverted for every input set.

For each of the benchmarks, various percentages of nodes were selected from 0% to 100% in increments of 1%. Two passes were performed at each percentage in order to generate two netlists or 202 total netlists. The first netlist pass replaced logic elements with LUTs according to the selection heuristic in Section 4.3.6. The second pass replaced the same amount of logic, but instead of utilizing the heuristic selection, it chose random nodes. The netlists were converted into SystemC and a functional simulation run on each netlist. The simulation assumed that the configuration values for the LUTs were unknown in both cases since the attacker randomly guessed LUT configurations and randomly guessed input combinations, but they knew what the expected outputs should be. A more effective attacker who uses targeted attacks instead of brute force attacks will be considered in following experiments.

For each netlist, $2^2 1$ input combinations were generated with the LUT values changing every 1000 iterations. For each input combination, the outputs were checked to see how closely they matched the expected outputs and a hamming distance score was assigned. The hamming distances were averaged over every input combination to determine an overall score for that netlist. Since we are trying to obscure the logic network such that an attacker cannot gain information about the configuration of the LUTs, it is desirable to have a large hamming distance. Overhead is also a concern, so minimizing the number of nodes replaced by LUTs allows for less performance penalties. These two metrics conflict as 0% selected nodes would be the original circuit, have the least amount of overhead, but no security and 100% of the circuit replaced with LUTs would be very secure, while maximizing the overhead. It should also be noted that a completely random circuit will generate output values of ones and zeros with the same frequency and therefore be correct, when compared to the functionality of the

circuit under test, approximately 50% of the time. This makes a model building attack more difficult since the attackers burden is increased with a randomized circuit when compared to the expected outputs as there is no bias for them to build from. It is therefore desirable for a given benchmark to have a hamming distances close to 50% and to climb to that percentage quickly as the percentage of total nodes selected increases.

Figure 5.2 shows the results of the simulations for all of the benchmarks. In this graph, the red (dark gray) lines represent the random selections and the blue (medium gray) lines, the heuristic selections. Each one of the thin lines represents a single benchmark consisting of 101 data points, one for every percentage of gates selected. The thinker lines are 6th order polynomial trendlines fit to the heuristic selections and the random selections respectively. These lines represent the general trends of the heuristics compared to the random selection and help to distill the large amount of information presented. The hamming distance of the heuristic selection trendline climbs more quickly towards 50% than the random selection showing that it requires a smaller percentage of the gates to be replaced with LUTs in order to obscure the design. Even though the trendline for the heuristic selection does not reach 50% hamming distance until slightly after 40% gate selection, for the majority of the heuristic netlists, they rise up to 50% before 20% and a few lagging benchmarks drag out the plateau. These numbers correspond to the location of the first cut and will be discussed in details in Section 5.3.2.

### 5.3.2 Single Cut Selection

As mentioned in Section 5.3.1, the percentage of gates selected and the overhead for selecting gates are conflicting metrics that must be optimized in order to get an acceptable hamming distance. In Figure 5.2, it was observed that the point at which each of the benchmarks reached a 50% hamming distance score was very near the point when the selection heuristic finished selecting a single cut of the network. After that point, the hamming distance remained almost constant as the number of gates selected increased. This occurs because once every node of a single cut has been chosen, the entire network is separated into two networks with a series of LUTs in the middle causing all of the paths from inputs to outputs to go through a LUT.

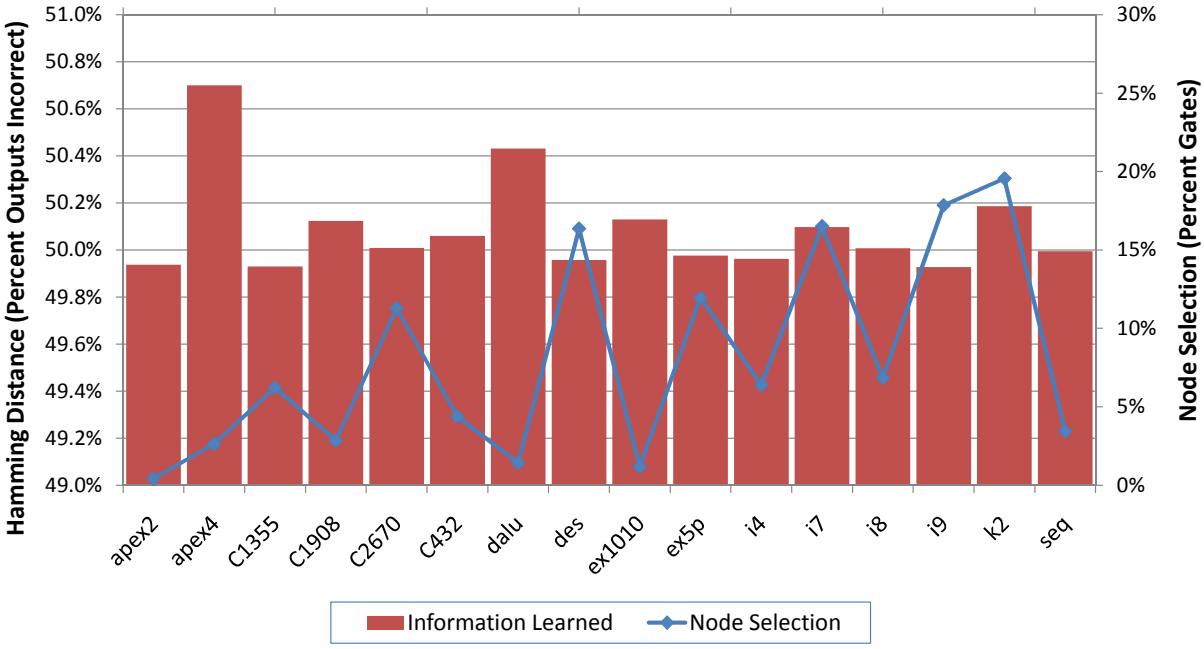



Figure 5.3: Single cut experiment for all benchmarks. The red (dark gray) bars represent the hamming distance of a single cut and the blue (medium gray) line, the percentage of total gates selected in a single cut.

Without having the correct configurations, the propagated inputs are obscured in the LUTs and lead to incorrect output values. Figure 5.3 shows how the benchmarks are affected by a single cut heuristic selection. For each benchmark, two data points are shown. The red (dark gray) bar chart represents the hamming distance observed for a single cut heuristic selection. Overlaying the bar chart is a blue (light gray) line graph representing the percentage of gates selected in order to replace an entire cut. The hamming distances for a single cut are very good and for every benchmark are within 0.1% of being at 50% with some slightly above. The percentage of gates selected is also very low for most of the benchmarks, especially `apex2` (0.42%), `apex4` (2.63%), `dalu` (1.41%), `ex1010` (1.18%) and `seq` (3.43%). On average a single cut requires 8.08% of nodes to be selected. Additional overhead is introduced due to the reduced controllability and observability. If only $k$ of the $n$ input nodes of the cut are controllable, the multiplicative factor for each of input to output relation is $2^{n-k}$. In the same way, if the sum of the observability of the $m$ nodes of the cut, $q$, such that $q < m$, the multiplicative factor is $2^{m-1}$. This means that a single cut is sufficient to make the circuit act as random logic while

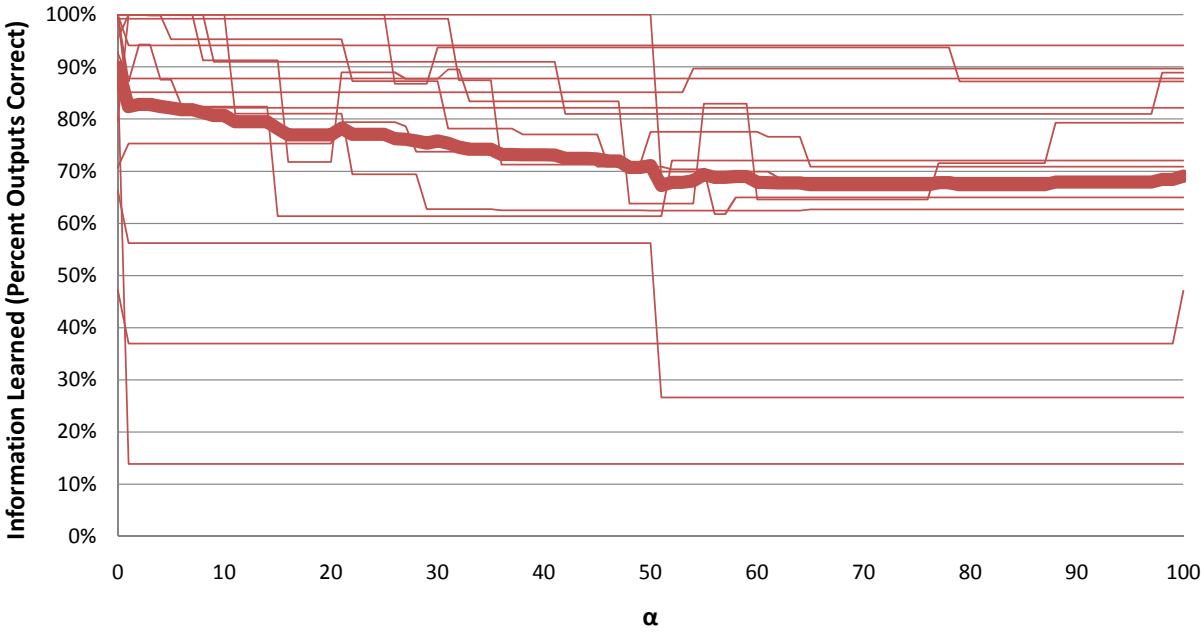


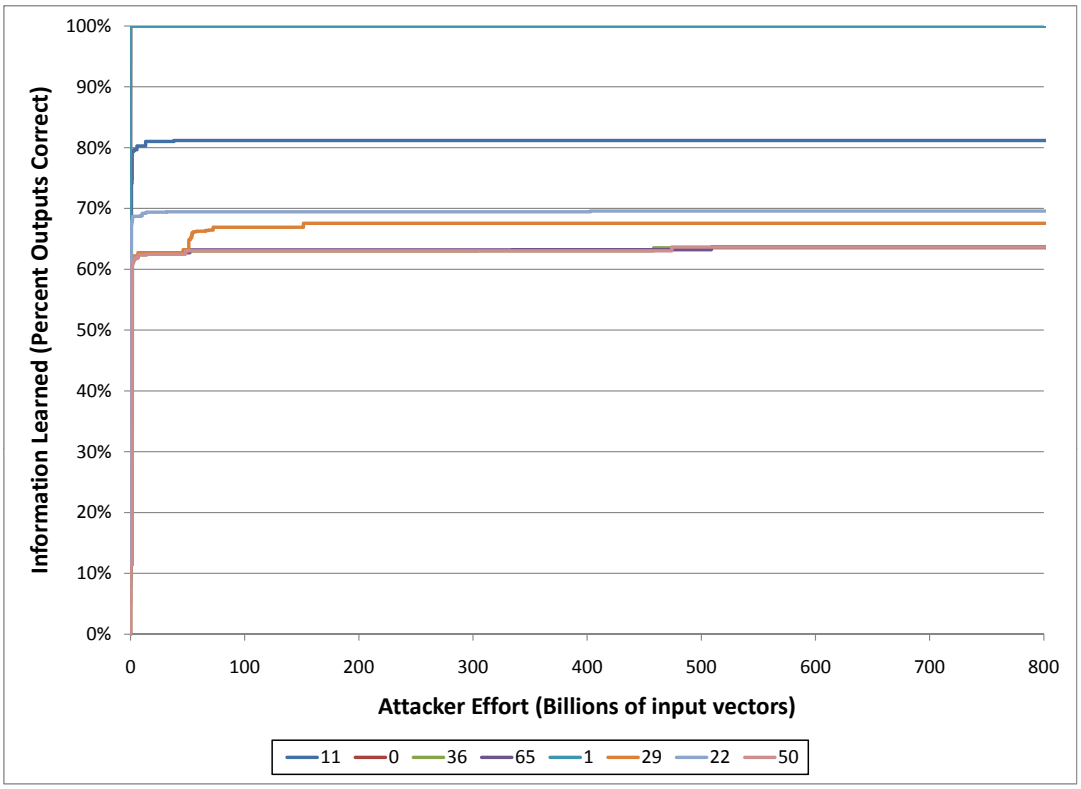

Figure 5.4: Percentage of information learned as a function of $\alpha$ for all benchmarks. The thin lines represent the individual benchmarks and the think line, the average over all benchmarks.

only requiring a small percentage of the logic to be abstracted as LUTs. For all of the following experiments, the heuristic selection uses a single cut across the benchmark for the selection criteria.

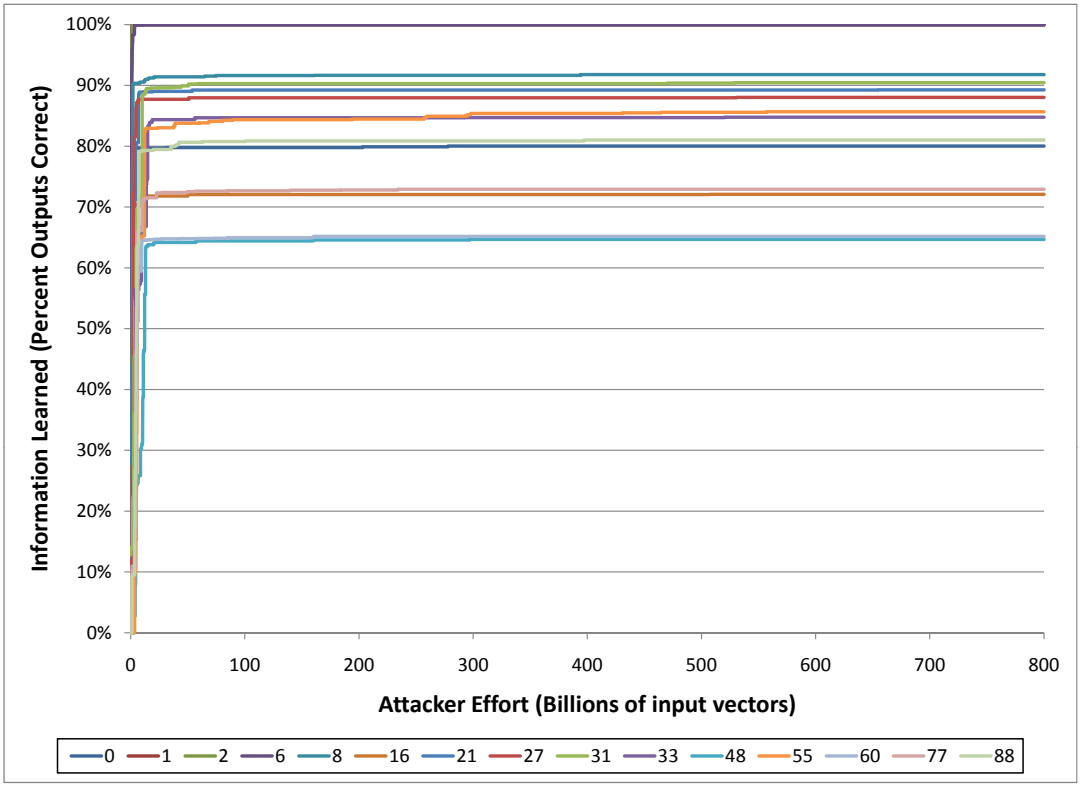
### 5.3.3 Cut Height

Section 4.3.4 showed why cuts located near the middle of the circuit in terms of height are better than cuts located towards either of the extremes. In this experiment, we show empirically this is the case. To add a height factor, a parameter $\alpha$ was introduced into the heuristic selection criteria. The heuristic selection, explained in detail in Section 4.3.6, is a weighting parameter that represents the ranking heuristic's dependence on the $\boldsymbol{ODC}$ and height values. If $\alpha$ is set to 0, only the $\boldsymbol{ODC}$ values of a cut are considered and if it is set to 100, only the height is used to determine cut ranking. All $\alpha$ values in between proportionally weight each of the criteria in the decision. For the previous experiments, $\alpha$ was set to 0, but as will be shown in this section, $\alpha$ can be tuned for more desirable results.

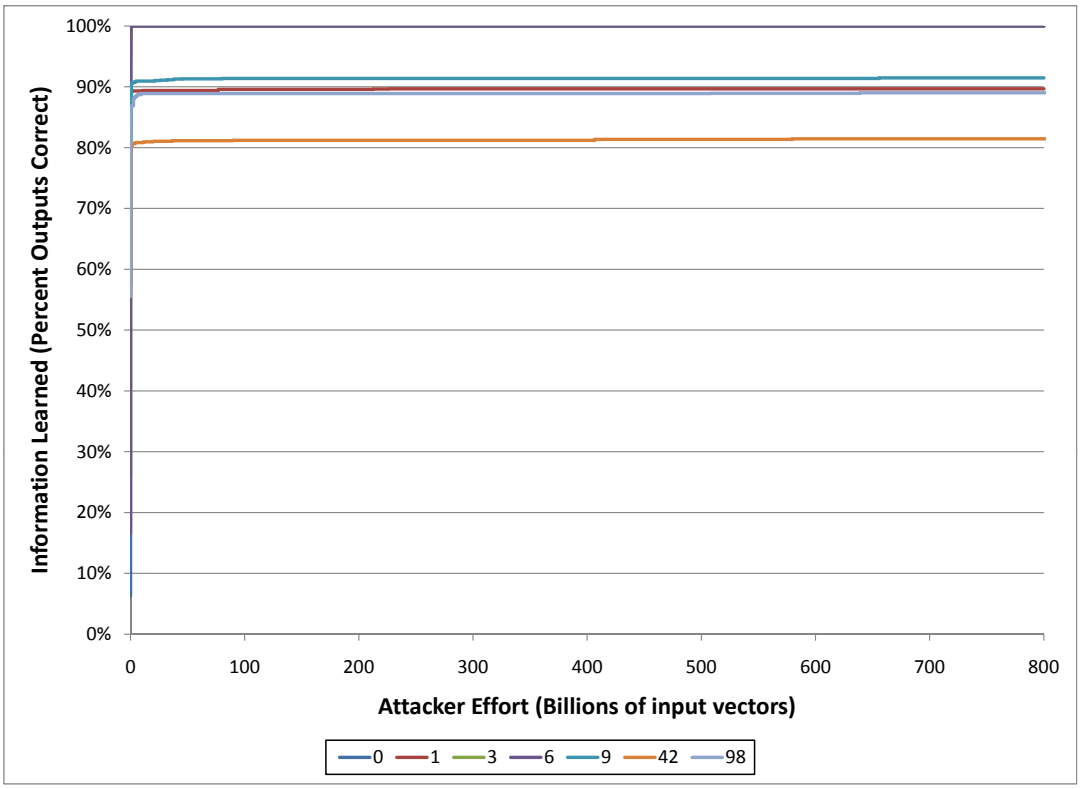
In this experiment, the same benchmarks were used with the percentage of gates fixed



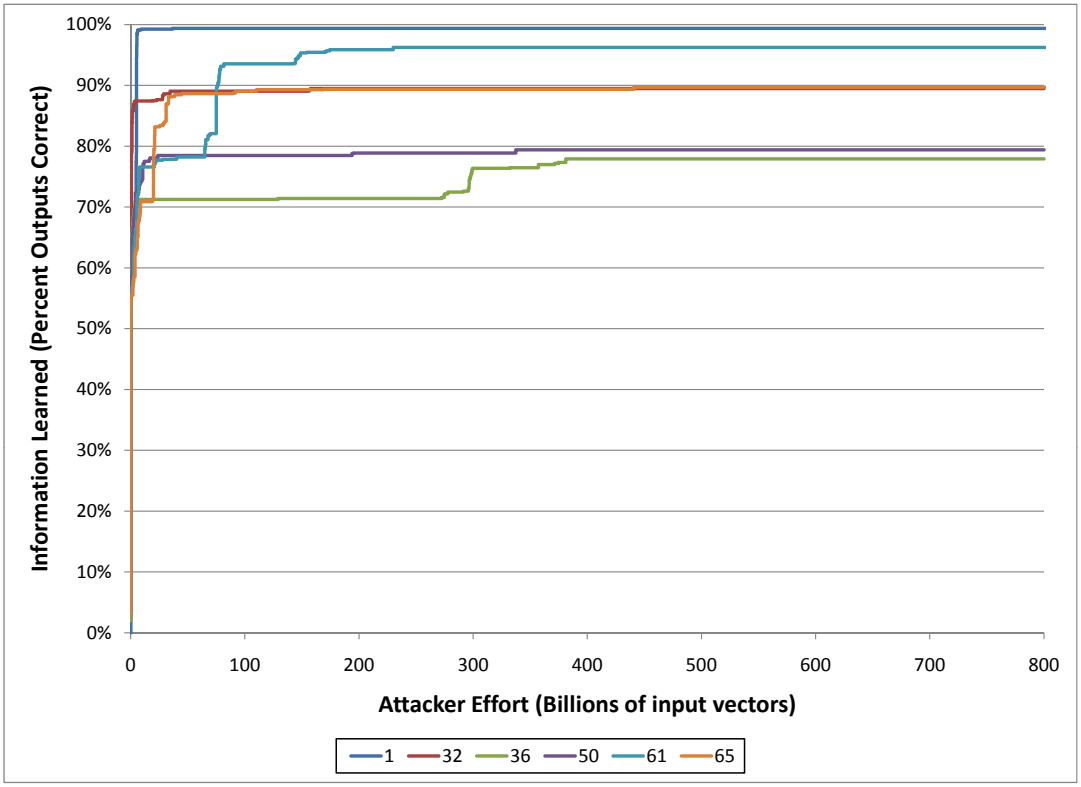

(a) apex2



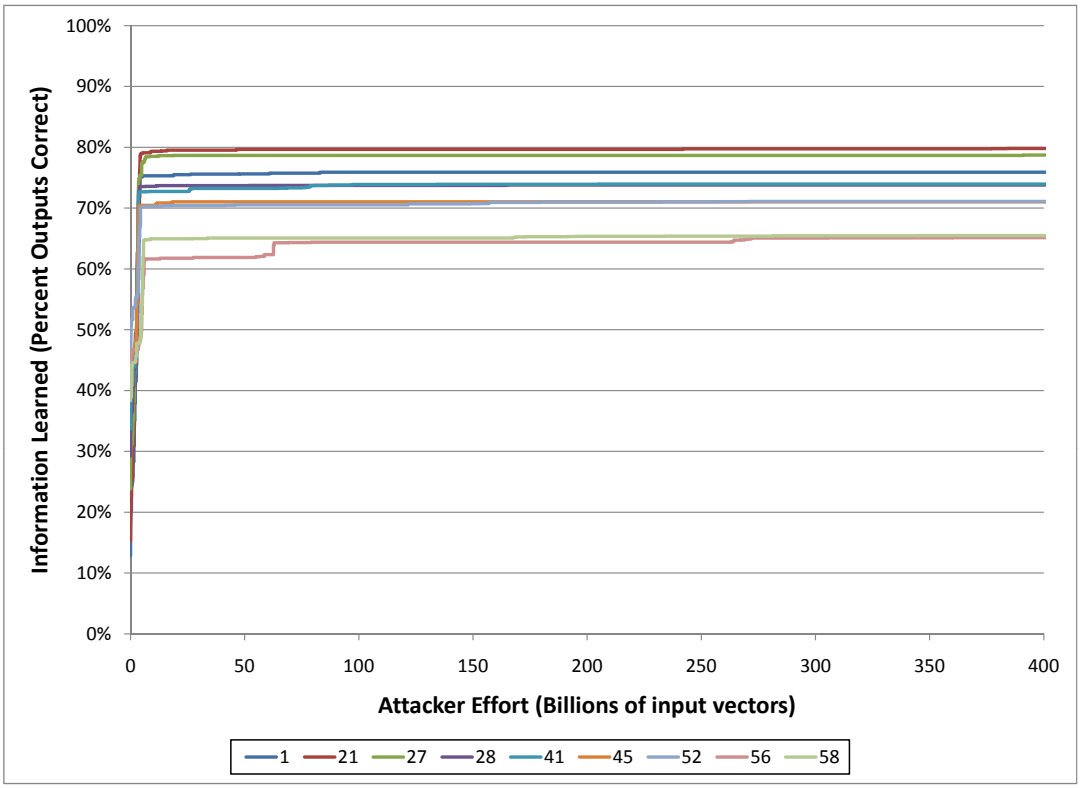

(b) apex4



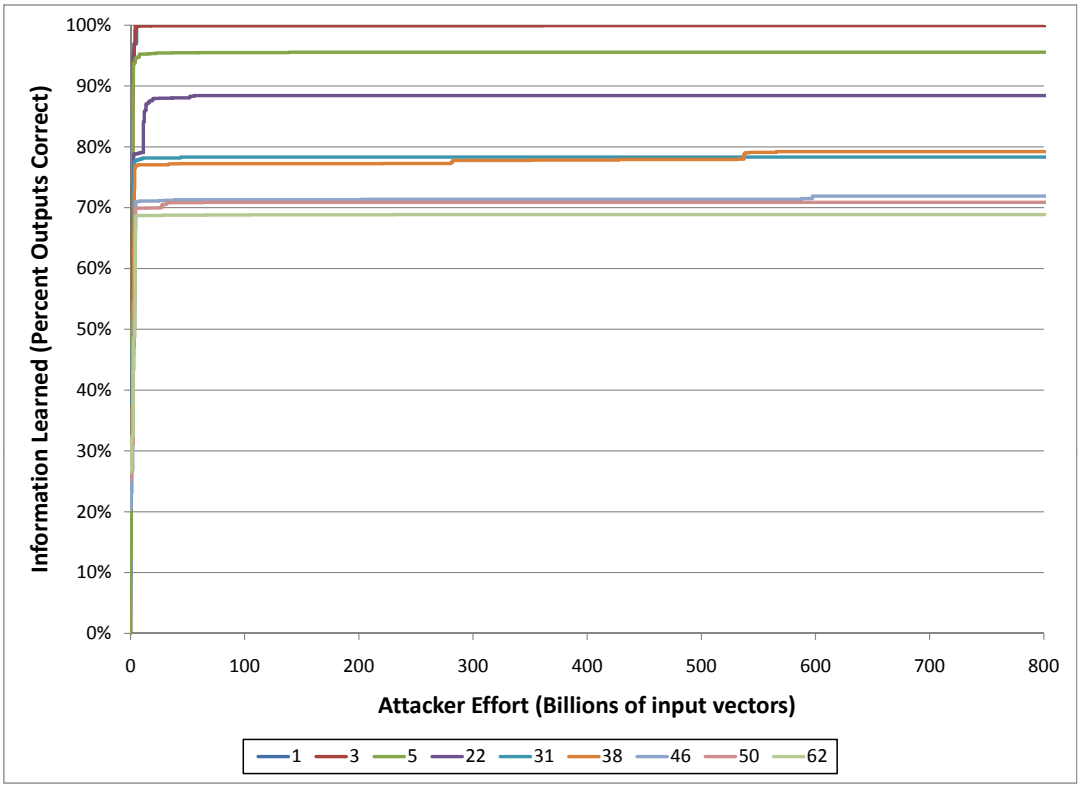

(c) C1908




(d) C432




(e) ex5p




(f) seq

Figure 5.3: Percentage of information learned as a function of the number of input vector guesses for six benchmarks, `Apex2`, `Apex4`, `C1908`, `C432`, `ex5p`, `seq`. Series names correspond to the $\alpha$ value which led to the cut selection.

to a single cut since Section 5.3.2 showed that selecting more gates than a single cut was unnecessary. For each of the benchmarks, the value of $\alpha$ was varied from 0 to 100 reflecting complete dependence on the $\boldsymbol{ODC}$ value at one end and the complete dependence on height at the other extreme. At each of the 101 increments, a netlist was generated such that the best cut was selected using the $\alpha$ parameter. Since $\alpha$ is a weighting parameter, it does not mean that 101 unique cuts were selected and in fact on average only 5.68 unique cuts were selected per benchmark. The number of unique cuts per benchmark can be seen in Table 5.2. For each of these unique cuts, a simulation was performed to measure how much information could be gained about the configuration of the LUTs in the circuit.

For this experiment, we assumed a smarter attacker than the previous ones. The new attacker is able to determine if the outputs of a circuit are expected given a set of inputs. This would correspond to an attacker understanding what the circuit is supposed to do, but not how it works. The attacker is also aware that a logic replacement scheme has been used and that instead of treating the LUTs as one large key space, targets individual LUTs. We model this by allowing the attacker to repeatedly configure one LUT while holding the rest constant and observing how closely the outputs agree with their expected values for known test vectors. When the LUTs are treated as completely dependent, the key space size is $2^{(2^K \times N)}$ where $N$ is the number of LUTs in the circuit and $K$ is the size of the LUTs. If each LUT was independent of the others and could be solved individually, the key space would be reduced to $2^{(2^K)} \times N$. But, as will be shown in this section, with smart choices of nodes, the LUTs cannot be treated as independent.

For each of the unique cut netlists, the simulated attacker attempted to solve the LUTs individually. Each LUT was initially configured with all zeros and then iteratively LUTs were chosen and every possible configuration, $2^{(2^K)}$, was tried while the other LUT configurations were held constant. For each configuration tried, $2^{11}$ input combinations were exercised and the information learned for each configuration recorded. The information learned was calculated as the percentage of outputs out of the number of possible outputs correct, $\frac{ouputsCorrect}{(outputs \times 2^{11})}$. Out of the $2^{(2^K)}$ configurations, the one that led to the most information being learned about the circuit

was chosen. The process repeated with a different LUT while the best guess configuration of the previous LUT was fixed. The process continued through each LUT and then re-iterated over the LUTs until 400,000,000 different LUT configurations were tried. This number was chosen because it allowed the largest benchmark to complete a single iteration through the LUTs. This number is also sufficient because as Figure 5.3 shows, as the number of configuration attempts increases the amount of information learned plateaus for all benchmarks and most, well before. Again, these numbers are further amplified by a reduced controllability and observability.

Figure 5.3 also shows how $\alpha$ affects the amount of information learned. Since our metering approach tries to minimize the possibility of an attacker figuring out the key value, we want to choose nodes for LUT replacement such that it minimizes the amount of information that the attacker can learn about the circuit making it more difficult for the attacker to replicate the design. Therefore, we would like to see the amount of information learned by the attacker level off at a low percent. For each of the six graphs in Figure 5.3, several lines are graphed each representing a unique cut. Since 101 unique cuts are not generated, the $\alpha$ value represents the first in a range between that value and the next listed $\alpha$ value. From each of these charts, the $\alpha$ values which levels off at the lowest information are located near 50 while the ones that level off much higher are closer to one of the extremes, 0 or 100. This means that a good balance between the $\boldsymbol{ODC}$ and the height, around 50%, when computing the cut scores yields a cut that is more resistant to the attacker modeled in this experiment then other weightings. The plateau effect shown in these figures is very beneficial because it means that even an attacker with infinite time and resources who attacks as we modeled will not be able to gain 100% information. The attacker is forced to either brute force the key or determine a different manner in which to crack the key value.

Figure 5.4 aggregates the data from all of the benchmarks. In this graph, the thin lines represent one benchmark as the value of $\alpha$ is incremented from 0 to 100. The steps occur since not all of the $\alpha$ values produce unique cuts which means many $\alpha$ values yield exactly the same circuit. The average of all of the benchmarks is plotted as the thicker red line. The line shows that the amount of information learned decreases as $\alpha$ increases until $\alpha$ is just past 50 at which

Table 5.2: Experimentation details of the MCNC benchmarks used

| Benchmark | $\alpha = 0$ Nodes | Best $\alpha$ | Best Cut Nodes | $\alpha = 100$ Nodes | Unique $\alpha$ Cuts | Key Length |
|---|---|---|---|---|---|---|
| apex2 | 4 | 50 | 135 | 140 | 8 | 1848 |
| apex4 | 19 | 48 | 192 | 198 | 15 | 1082 |
| C432 | 7 | 50 | 33 | 39 | 6 | 280 |
| C1355 | 32 | 51 | 64 | 74 | 2 | 576 |
| C1908 | 25 | 42 | 91 | 101 | 7 | 216 |
| C2670 | 131 | 53 | 135 | 272 | 4 | 778 |
| dalu | 16 | 48 | 70 | 89 | 3 | 440 |
| des | 245 | 51 | 459 | 459 | 3 | 4184 |
| ex5p | 63 | 56 | 169 | 257 | 10 | 2478 |
| ex1010 | 10 | 51 | 103 | 103 | 7 | 296 |
| i4 | 6 | 51 | 33 | 33 | 2 | 82 |
| i7 | 67 | 51 | 275 | 275 | 2 | 1416 |
| i8 | 81 | 46 | 430 | 430 | 3 | 1154 |
| i9 | 63 | 44 | 202 | 202 | 3 | 2152 |
| k2 | 44 | 45 | 98 | 210 | 6 | 2056 |
| seq | 35 | 62 | 293 | 317 | 10 | 3242 |
| Average | 53 | 50 | 174 | 200 | 6 | 1393 |

point the line slowly increases. There is a sharp drop in the curve when $\alpha$ is near 50 dropping down to 66.4% for $\alpha = 53$. The information learned fluctuates within a few percent around this value as dropping to that level twice in the 60s, once in the 70s and then also in the 80s before rising as $\alpha$ approaches 100. This means that an $\alpha$ value that is a minimum would be a good choice, a value slightly above 50.

From these experiments, we can see that our node selection heuristic outperforms the random selection of previous approaches. We observed that by separating the logic network into two parts with a cut yields the best results for the amount of overhead it introduces. Also, by introducing an extra parameter, $\alpha$, which weights the height of a cut along with the number of ***ODC*** minterms, the quality of the selection was further increased.

# CHAPTER 6. CONCLUSIONS

## 6.1 Summary of Results

This work began with some implementations of Trojans in an IC supply chain. These proof of concepts showed that it was feasible to insert Trojans into HDL source code in order to leak information, cause a denial of service and to utilize the IC for a function it was not intended. This work was part of our teams first place entry in the CSAW design contest. A thorough literature survey led to our taxonomy of attackers in the IC supply chain. These classifications are useful in understanding the bigger picture of vulnerabilities and allow us to establish a location inside the framework of the bigger picture.

The main contribution of our work is an active metering solution to IC piracy that is more secure and has less overhead than previous attempts. Our combinational locking scheme partitions an IC circuit design into a fixed and reconfigurable region before fabrication. The foundry never sees the configuration destined for the reconfigurable regions, but instead fabricates them onto the IC with facilities to configure post-fabrication. In this way, part of the design is withheld and acts as a key to unlock fabricated ICs which are useless without proper activation. Our approach augments the traditional CAD flow with an additional pass over the logic network in order to partition the network into the two regions. Our cut-based node selection heuristic utilizes a combination of ODC analysis and height metrics in order to select the nodes that increase the security guarantees while minimizing the overhead. The chosen nodes are abstracted as reconfigurable LUTs which allows the CAD tools a large design transformation space.

Our active metering solution was implemented in open-source CAD tools and tested using a popular set of benchmarks. Automated experiments were run to demonstrate how much

information an intelligent attacker could gain from attacking an IC protected by our approach. Our results showed that our heuristic selection was better than previous random node selection schemes and that a single cut provided the best protection to overhead ratio. We also demonstrated the effects on information leaked by tuning the heuristic selection parameters.

## 6.2 Future Works

Although our work has created an effective IC metering scheme and shown considerable improvements over past work, there is room for future work. Section 4.3.1.1 quantified some of the overhead associated with our approach, but a more thorough investigation of the overhead including area, power, and delay characteristics would be useful. The results from the overhead could be used to drive the selection heuristic or other metrics could be considered for inclusion which could increase security or decrease overhead. This could include provisions for the nodes located along the critical path or consideration of the application in which the IC is used such as the resource constraints of embedded systems. The implementation of our heuristic in the CAD tools could be optimized in order to support the processing of larger circuits in shorter time. A better understanding of what architectures are well suited for our approach would be useful. This information could be used in the selection heuristic in which large designs where partitioned according to their architecture and certain styles targeted for selection. Several possible attacks on our approach were suggested and tested, but other more effective ones may exist. The use of reduced order binary decision diagrams (ROBDDs) may be useful in solving the unknown LUT configurations. Using ROBDDs would allow an attacker to compare networks created from incorrect configurations in order to make educated guesses on the next configuration to try.

Our solution used LUTs as an abstraction for the functionality of targeted nodes of the logic network. Another avenue could be to explore the abstraction of non-functional characteristics including delay, area, and power. For example, instead of hiding the logic of a circuit, the delay characteristics could be hidden such that an incorrect key value would cause the delay to be skewed and effectively cause the functionality to be affected. Another structure could be added

to the IC design which would not allow multiple LUT configuration attempts or which would cause permanent damage to the IC with incorrect configuration attempts. Such approaches could also be used in combination with the existing one in order to be more effective.

Other points in the supply chain where an attacker could enter are an open avenue to explore. Our approach focuses on IC metering, but as discussed in previous sections, there are many attack avenues. Research at various levels may allow for a more elegant and broader solution than just patching existing ideas together.

# APPENDIX PROOF OF CONCEPT

In the fall of 2008, the Computer Security Awareness Week (CSAW) Embedded Systems Challenge [1] was held at the Polytechnic Institute of NYU. This competition highlighted some of the vulnerabilities of the IC supply chain in the form of a hardware hacking challenge. The story-line for the challenge began with the creation of a new cryptographic device, code named *Alpha*, for a fictitious army. The Alpha device allows soldiers to type messages and transmit them securely to other soldiers and their command station. The HDL design for the device has been leaked through a mole and was given to the participants. With this source code in hand, each student-led team had one month to implement as many undetectable hardware Trojans as possible. The Alpha device, augmented with Trojans, had to pass a set of functional tests, use the same reference power, maintain the configuration memory usage, pass a brief code inspection and be undetectable by a general user. With these requirements in mind, our team from Iowa State University designed a wide set of applicable Trojans and performed a proof-of-concept implementation using a provided Field-Programmable Gate Array (FPGA) board.

## Competition Details

The CSAW Embedded Systems Challenge began in September 2008, concluding a month later in New York. Each team was given a BASYS development board [20] that contained a Xilinx Spartan FPGA along with basic peripheral I/O. A basic schematic and picture of our experimental setup can be seen in Figure A.1 and Figure A.2. The original design of the Alpha required a PS/2 keyboard to be connected to the board for input and a VGA monitor along with a serial connection for output. Four other buttons changed the state of the system. Xilinx ISE 10.1 was used for development of the mixed-mode HDL code and for generating the

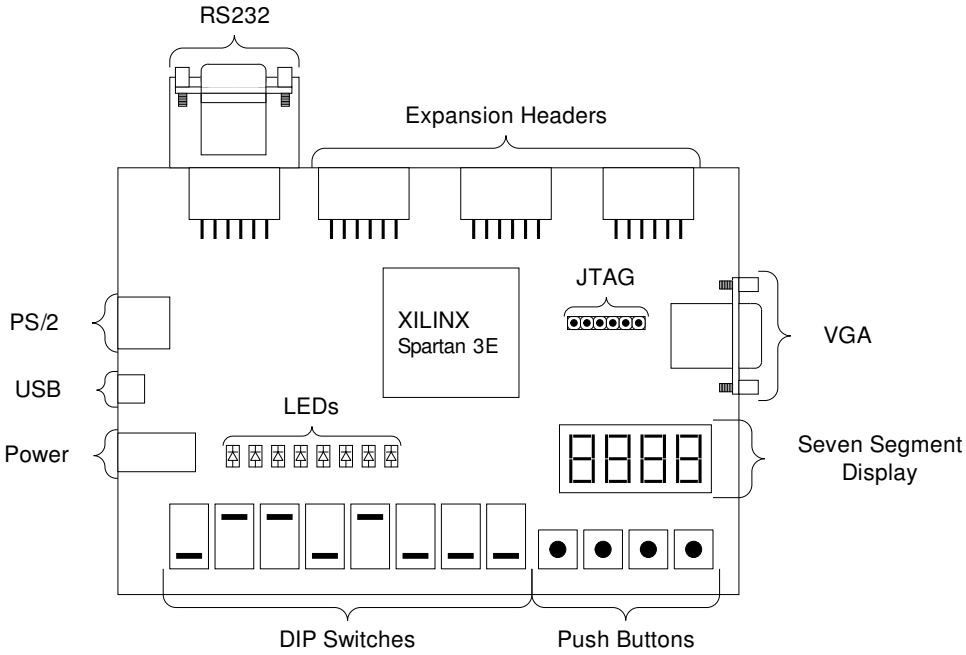



Figure A.1: Diagram of components used from the BASYS board

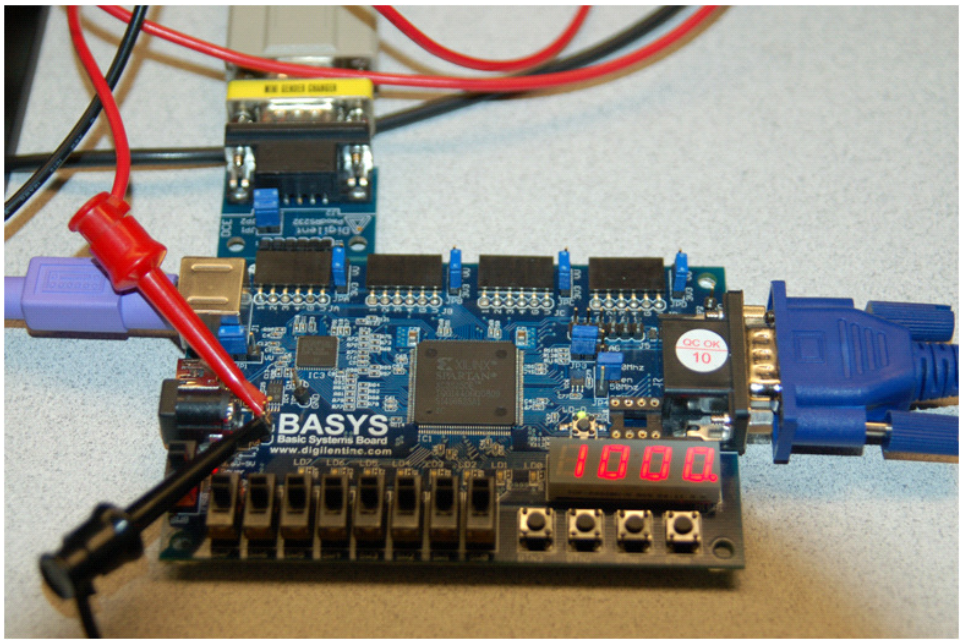



Figure A.2: Experimental setup of BASYS board and peripherals

bitstream for the FPGA. Modelsim SE 6.3 was used to simulate each Trojan. Additionally, our team made use of an oscilloscope, power supply, multimeter, thermometer, HAM radio, and custom circuits to verify the functionality of each Trojan.

In normal operation of the Alpha device, a message is typed in plaintext on the keyboard and displayed on the VGA monitor. When the encrypt button on the board is pressed, the message is sent through an AES-128 encryption block with the key selected by the 5 dip switches.[1] Finally, when the transmit button is pressed, the system sends the encrypted message over the serial port. The receiver of the message can then decrypt the message using the correct key. For the purpose of this contest, a C program called encVerifier was provided in order to receive and decrypt the message. For scoring the designs, the judges evaluated the completeness and uniqueness of the Trojans along with their ability to be undetectable to a set of functional tests, deviation in power consumption, deviation in bitstream size, and their ability to pass a brief code inspection.

## Implemented Attacks

Our team's approach started with breaking a Trojan down into three areas of functionality so that we could better target each area. The three categories, further explained below, are thwarting the user's plans, gaining extra knowledge, and exercising additional functionality:

1. Affecting the operation of the device either by making it function incorrectly or not function at all. This form of attack could be seen as a Denial-of-Service (DOS) where some subset of the functionality does not work as intended.

2. Leaking sensitive information that was not originally intended to be leaked from the device. This could include allowing a malicious user to capture the plaintext messages, the key used to encrypt those messages or any other information that would allow that malicious user to have more knowledge of the system then they were originally intended to have.

[1] There is an obvious cryptographic limitation in using 5 dip switches to select unique 128-bit keys, but conceivably a more elegant model could be used for obtaining the key in a production environment.

3. Utilizing the device for a function which it was not intended to be utilized. In this attack, the device may function correctly, produce the correct outputs, and not leak information, but it still may be used in a manner not intended by the original specifications.

Besides the functional aspects, a Trojan must be very well hidden to escape detection. The existence of each attack must be undetectable during normal operation because its worth is calculated by its ability to escape detection in addition to how well it meets the objectives of the three categories listed above. On another level, the Trojan must be well hidden in the HDL code before it gets a chance to be synthesized. Significant effort was spent in ensuring this second method of hiding. Most of the details pertaining to how the attack was hidden in the HDL code are not described in the following sections, as this deals primarily with specific wires and modules that are not generalizable to other hardware Trojan circuits. In general, methods such as extending bus widths, rerouting signals, utilizing naming conventions, decentralizing the Trojan logic, writing misleading comments, and exploiting language nuances were used to hide the various attacks described in this section.

A second consideration when designing the Trojans is the triggering of malicious functionality. Some of our Trojans, like those creating a DOS, should not be triggered immediately (or they will fail validation tests) and instead should remain hidden. Other Trojans, such as the RF-based information leakage attacks, Section 6.2 and Section 6.2, do not need to be triggered because they remain ostensibly hidden as they operate. For the Trojans that require explicit triggering, various methods can be used such as a timer, special sequence of keystrokes, special packet sent to the board, etc. Each trigger has its disadvantages, but by decreasing the likelihood of the triggering event occurring randomly, the Trojan has a better chance of remaining hidden. For our Trojans which required triggering, we utilized a hardware counter to delay activation, but this scheme could be easily switched in an manner independent of the Trojan's functionality.

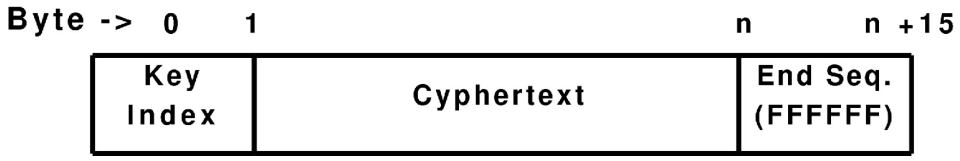



Figure A.3: RS232 message format

**RS232 End Sequence Information Leakage**

**Explanation**

We created three Trojans using the RS232 module. Although they all modify the transmission of data, they are fundamentally different, Figure A.3. The first Trojan takes advantage of the message structure. The message structure begins with the key index used to decide the key needed to decrypt the message. The key index is based on the dip switches found on the physical board used to add some entropy to the master key. Following the key index is the ciphertext, which is the original message encrypted using the AES-128 algorithm and the private key. Finally, an ending sequence composed of fourteen bytes of 0xFF values is sent.

The program used to decrypt the messages on the receiver side, encVerifier, uses the key index to know which key was used to encrypt the message and consequently which key needs to be used to decrypt the message. This does trivialize the security, but provides a convenient key distribution mechanism. The encVerifier program then reads the ciphertext, but since it is of variable length, it needs the ending sequence to determine the end of the message. The source code for the encVerifier program checks for the presence of five bytes of 0xFF in the last available eight-byte segment. It then loops through the data from the last read byte to the first read byte in the eight byte sequence and increments a counter for each byte of 0xFF observed. If at least five bytes of 0xFF have been read in the last eight byte segment, the encVerifier program decrypts the message and displays it to the screen. A section of the encVerifier program can be seen below:

```
while(STOP==FALSE)
{
//Read up to 8 bytes from the serial port
```

```
rx = read(fd,buf+totalRX,8);
for(i=0;i<rx;i++)
{
  if(buf[totalRX-i]==(unsigned char)'\\xff')
    {e++;}
}
if(e>4)
  STOP=TRUE;
}
```

Exploiting the message structure, specifically the ending sequence, allows for two attacks. The first attack places information after the transmission of the ending sequence, but still in the original message. This attack is flexible in what information can be transmitted and the amount of information that can be transmitted. We chose to leak the entire key at the end of one message. Since the encVerifier program stops reading from the RS232 stream once it has detected the ending sequence, extra information may be placed on the stream unnoticeable to the verification program. A malicious program monitoring the stream would be able to read past the ending sequence and receive the extra data placed on the end of the message.

**Results**

When this Trojan is enabled, the ending sequence is embedded with extra data. Using the original encVerifier program it produced the expected output as it ignored the added information:

```
> ./encVerifier
Waiting for transmission to begin.....
Total bytes received: 24
Ciphertext:
   003BB579322F5F2E4AF7E9476D411DA83BFFFFFFFFFFFFFF
Using Key:
   FB7915BDF1E5C8B84BB718DD34D73300
Plain Text:
   CSAW 2008 - ISU
```

When the malicious encVerifier program is run, it ignores the ciphertext message and all of the 0xFF bytes of the ending sequence and reads the embedded key from the RS232 transmission without prior knowledge of the key. The output from this program can easily be seen:

```
> ./encVerifierTrojan1
Waiting for transmission to begin.....
Total bytes received: 60
Key:
    FB 79 15 BD F1 E5 C8 B8 4B B7 18 DD 34 D7 33 00
```

Looking at the raw hex values from the RS232 stream reveals how the key was embedded. The first line shows the original message format that the encVerifier program expects. The second line continues right after the first in the transmission stream, but the encVerifier program ignores that data allowing a malicious encVerifier to read past the point the original program stopped reading:

```
00 3B B5 79 32 2F 5F 2E 4A F7 E9 47 6D 41 1D A8 3B FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

FB 79 15 BD F1 E5 C8 B8 4B B7 18 DD 34 D7 33 00 FF FF
FF FF FF FF FF FF FF FF FF FF FF FF
```

### RS232 End Sequence Information Leakage 2

#### Explanation

The second exploit to the end sequence takes advantage of the number of ending sequence bytes that are sent. Since the ciphertext can be of variable length and the encVerifier program checks for the presence of five bytes of 0xFF in the last eight byte segment, more than five bytes of 0xFF must be sent, but fewer than the fourteen bytes of 0xFF that are sent by the reference system. With a variable length message, there are eight possible locations within an eight byte block where the end sequence can start. Each of these cases is depicted in Figure A.4, where the fourteen byte wide table represents the fourteen bytes of 0xFF sent as the ending sequence of a message transmission, the rows represent each of the eight cases, and the numbers in the

| Byte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Case 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 |
| Case 2 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 |
| Case 3 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 |
| Case 4 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 |
| Case 5 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 |
| Case 6 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 |
| Case 7 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Case 8 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Figure A.4: Ending sequence cases. Gray shading differentiates eight bytes segments. White indicates which bytes cause encVerifier to stop receiving.

| Byte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Case 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 |
| Case 2 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 |
| Case 3 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 |
| Case 4 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 |
| Case 5 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 |
| Case 6 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 |
| Case 7 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Case 8 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Figure A.5: Ending sequence cases. Same coloring as Figure A.4 but the dotted white cells indicate where hidden messages could be placed.

row represent the location of that byte in an eight byte segment. The two shades of gray show adjacent messages and the white cells are the five bytes of 0xFF that the encVerifier program uses to determine the end of a sequence. The fourteen bytes of 0xFF are more than enough to create an ending sequence and other data can be contained in the original message size as seen in Figure A.5. The markings in this figure are the same as Figure A.4 with the addition of dotted white cells representing the five bytes of data that could be used for adding arbitrary information. In the later figure, each case gives the encVerifier program sufficient amounts of 0xFF bytes for that program to correctly terminate, but it also allows for five bytes of arbitrary data to be embedded in the stream.

Because of the AES-128 encryption scheme, the ciphertext is always in sixteen byte blocks. Those sixteen bytes plus one byte of key index and the fourteen bytes of 0xFF for the end sequence result in a message of $(16 \cdot n + 14 + 1)$ bytes, where $n$ is the number of cipher blocks. This means that the message transmissions will always fall into case eight from Figure A.4 and Figure A.5. Now, instead of being limited to five bytes by case three, the first nine bytes out of the fourteen bytes of 0xFF can be used to embed information as long as the last five bytes read contain 0xFF. This method is more covert than Trojan 6.2 because the five bytes of added data appear to be part of the ciphertext and not an extension to the end of the original message. Both Trojans are invisible to the original encVerifier program and allow that program to correctly operate in every instance.

### Results

When the message "CSAW 2008 - ISU" is sent to the encVerifier program without modifying the transmission stream, it produces the following output:

```
> ./encVerifier
Waiting for transmission to begin.....
Total bytes received: 24
Ciphertext:
    003BB579322F5F2E4AF7E9476D411DA83BFFFFFFFFFFFFFF
Using Key:
```

```
    FB7915BDF1E5C8B84BB718DD34D73300
Plain Text:
    CSAW 2008 - ISU
```

Notice that the above output receives 24 bytes of data when actually 31 bytes of data were sent. Reading the raw hex values from the RS232 port gives the following output. This is also the expected output of the entire message format:

```
00 3B B5 79 32 2F 5F 2E 4A F7 E9 47 6D 41 1D A8 3B FF
FF FF FF FF FF FF FF FF FF FF FF FF FF
```

When the extra nine bytes of data are embedded between the ciphertext and the ending sequence, the encVerifier program still produces the same plaintext output. These bits of the key that were transmitted appear as part of the ciphertext:

```
> ./encVerifier
Waiting for transmission to begin.....
Total bytes received: 31
Ciphertext:
    003BB579322F5F2E4AF7E9476D411DA
    83B00CCEB2CBB18EDD21DFFFFFFFFFF
Using Key:
    FB7915BDF1E5C8B84BB718DD34D73300
Plain Text:
    CSAW 2008 - ISU
```

This time there are only five bytes of 0xFF since the rest of the nine bytes that were previously 0xFF have been changed to parts of the key:

```
00 3B B5 79 32 2F 5F 2E 4A F7 E9 47 6D 41 1D A8 3B 00
CC EB 2C BB 18 ED D2 1D FF FF FF FF FF
```

### RS232 Multiple Transmission Rates

#### Explanation

The third attack on the RS232 port takes a different approach and instead of exploiting the message structure, it exploits the protocol itself. The RS232 protocol uses a single data wire

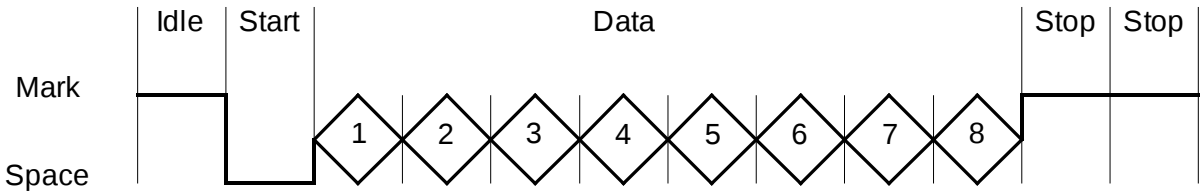



Figure A.6: A valid RS232 message frame. The data bits are shown generically, but would either be a mark or space.

for transmission. When the line is idle, it is a mark condition representing a negative voltage and logic '1'. Likewise, when the line is pulled up to a positive voltage, logic '0', it is a space condition. The RS232 specification allows for various combinations of baud rates, data bits in each packet, the number of stop bits, parity bits along with various other extensions. The Alpha uses a 9600 baud rate transmission with eight data bits in each packet and a start and stop bit corresponding to Figure A.6. The start bit must go from a mark to a space in order for the receiver to recognize the beginning of the asynchronous transmission. Once the receiver recognizes this condition, it can begin sampling the data bits at the agreed upon baud rate in order to extract all of the data from the packet.

The Alpha board and receiver currently receive at 9600 baud, but both are also able to transmit and receive data at faster rates. This Trojan crafts packets that are transmitted at a faster rate, yet when viewed at the slower rate appear to be a valid transmission. In this manner, two transmissions can be overlaid to allow two messages to simultaneously be transmitted at different baud rates. In order to maintain transmission at two baud rates, both transmissions must have complete frames including a start bit, stop bit(s), and data bits in addition to looking similar to valid data when reading the same data at two different rates.

Another standard transmission rate is 115200 baud (twelve times faster than 9600 baud), meaning that twelve bits are transmitted for every bit of data transmitted at the 9600 baud rate. In order to keep from encountering framing errors, a complete packet must fit into these twelve bits. This packet consists of an idle mark bit, a space start bit, eight data bits, and two stop bits for a total of twelve bits. This packet of twelve bits must appear similar enough to the constant '1' bit transmitted at the same time at 9600 baud for the encVerifier program to still

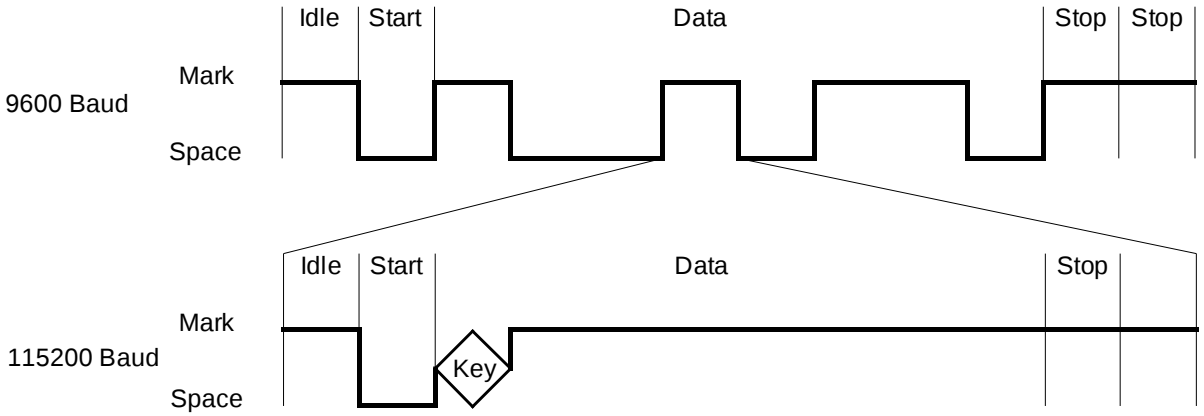



Figure A.7: A valid RS232 frame at 115200 Baud can be shaped as a mark bit.

identify the 9600 baud rate transmission. There is also some inflexibility in which bits can be shaped to look similar to the slower transmission since the initial mark bit and start bit must remain fixed along with two stop bits. This means that if the 9600 baud rate transmission is transmitting a mark bit, the 115200 baud rate transmission can match it on eleven out of the twelve bits, but if it is a space, then the best that can be managed is nine out of twelve bits. It turns out that both of these are acceptable to appear as the constant 9600 baud rate value. These two scenarios can be seen in Figure A.7 and Figure A.8, respectively. In these figures, the upper transmission packet is sent at 9600 baud and the lower transmission packet at 115200 baud. Since the lower transmission is twelve times faster than the upper, the bottom transmission frame represents a single bit of the upper transmission frame and has been shaped to look as much like the bit from the upper frame that it is representing.

The transmissions of the mark bit at the slower rate can be used to embed one bit of additional information, more specifically the key. By shaping the faster transmission to look like the slower transmission, the mark bit can be more accurately represented, given that eleven out of the twelve bits are the same. If one bit of information is embedded in the data section of the faster transmission, then the resulting transmission would match either ten or eleven out of the twelve bits, both of which are accurate enough be received without error at the slower transmission rate. Our modified Alpha transmitter functions at the 115200 baud rate sending out signals that look like the 9600 baud rate but with one bit of the key embedded

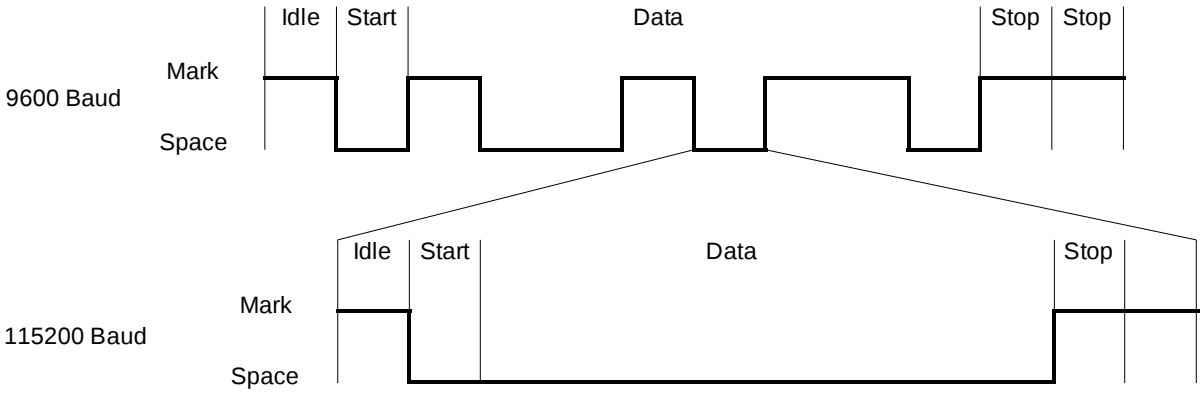



Figure A.8: A valid RS232 frame at 115200 Baud can be shaped as a space bit.

in each mark bit of the 9600 baud rate transmission. This allows the encVerifier program to verify that the 9600 baud rate transmission contains the correct information, but a malicious program listening at the 115200 baud rate can extract the key from the signal. The program encVerifierTrojan3 verifies that the 115200 baud rate transmission does in fact contain the key.

### Results

When this Trojan is enabled, it transmits both the expected data and the secret data as the same signal. Using the original encVerifier program it produced the expected output:

```
> ./encVerifier
Waiting for transmission to begin.....
Total bytes received: 24
Ciphertext:
    003BB579322F5F2E4AF7E947
    6D411DA83BFFFFFFFFFFFFFF
Using Key:
    FB7915BDF1E5C8B84BB718DD34D73300
Plain Text:
    CSAW 2008 - ISU
```

Reading the raw hex values from the RS232 port gives the following output. This is also the expected output of the entire message format:

```
00 3B B5 79 32 2F 5F 2E 4A F7 E9 47 6D 41 1D A8 3B FF
```

```
FF FF FF FF FF FF FF FF FF FF FF FF FF
```

When the malicious program, encVerifierTrojan3 is run, it listens at 115200 baud rate and returns the leaked key without any prior knowledge of the system:

```
> ./encVerifierTrojan3
Waiting for transmission to begin.....
Key: FB7915BDF1E5C8B84BB718DD34D73300
```

Reading the raw hex values from the RS232 port also gives the expected output. The 115200 baud rate signal shapes its bits to appear as a '1' or as a '0', so the eight bits of data sent at the faster baud rate should all be either zeros or ones, except for the bit of information embedded in the transmission of the ones. In the transmission output, the eight bits of zeros appear as 0x00 and the eight bits of ones as 0xFF. The 0xFE bytes represent a leaked zero and the 0xFF bytes a leaked one.

```
FE 00 00 00 00 00 00 00 00 00 FE FE FE 00 FE FE 00 FE
... FF 00 FE FE FF FF FE FE FF FF FF FE
```

### Denial of Service

#### Explanation

The goal of this attack was to create a DOS which occurs during operation, that would go unnoticed during normal device testing. The DOS attack can be implemented in many ways, but at its core, it degrades the performance or validity of the Alpha device. In order to implement a DOS, an acceptable location must be chosen such that it would remain hidden during the verification tests. Finding an appropriate target is actually trivial since almost any signal can be modified to produce incorrect output. The clock could be frozen to a value so that the entire device quits functioning, the transmission of the data could be corrupted so that it does not send correctly formatted RS232 frames, even the data read from the keyboard could be skewed so that incorrect messages were transmitted.

Our specific implemented DOS Trojan attacks the key used to encrypt messages and accomplishes its goals of denying service and being well-hidden by making only minor important

modifications to the code. It also is set on a timer so that it intermittently functions, switching approximately every 3.58 minutes. Finally, the user in the field will not notice that the Trojan is active since it only corrupts the ciphertext being sent out so that the intended receiver will receive a ciphertext encrypted with a different key than they expect. The attack makes use of a counter within the seven segment driver routine and a slight modification of the AES-128 routine. The seven segment driver already has a 12 bit counter running at 625 kHz. Adding 17 additional bits allows a 7.15 minute cycle time for the upper order bit. Adding a few more bits would substantially increase the cycle time allowing it to pass validation tests. The upper order bit is threaded through the seven segment driver as a fake enable signal which is connected to the AES-128 core as a similar enable. Within the AES-128 module, this bit is XORed with one of the key bits to corrupt approximately 50% of the ciphertext.

**Results**

From the user's perspective, the operation is not changed but the data sent over the RS232 protocol is corrupted. As a test, a single 'A' character transmitted with a random encryption key gives the serial output:

00 B6 97 60 D7 AA 40 62 03 B7 B1 20 4F 1D E9 25 39 FF
FF FF FF FF FF FF FF

When the Trojan is active, the output is:

00 6E FA 67 DB c8 82 56 A6 C4 01 E7 8C 92 B5 B8 5C FF
FF FF FF FF FF FF FF

XORing the two data streams gives:

00 d8 6d 07 0c 62 c2 34 A5 73 B0 C7 C3 8F 5C 9D 65

Showing that 62 bits out of 128 bits that were transmitted incorrectly (approaching the expected error rate of 50%).

## Thermal Leakage

### Explanation

Another interesting route for data leakage is through thermal transmission. In this attack, the FPGA is systematically heated up or allowed to operate at its normal state to create a binary code used for conveying information. This allows a malicious user to place a temperature probe on the FPGA and monitor the temperature of the device to collect the data being leaked (e.g. key bits). Even though this attack requires physical access to the FPGA, it has very real applications. These could include a scenario where one of the Alpha devices is captured allowing the captors to extract the key and decrypt all previous and future Alpha transmissions. This attack would also work if a malicious user had temporary physical access to the FPGA and was able to extract the key at that time.

The FPGA must be able to generate enough heat to be sensed by a temperature probe, on the order of a couple of degrees Fahrenheit. As with most devices, on an FPGA, static and dynamic current leakage account for the power dissipation and therefore heat generation. In our usage, since the configuration is fixed at run-time, the static power remains very similar between the original design and the modified thermal leakage design, and consequently most of the power variation comes through the dynamic power dissipation. Dynamic power dissipation can be modeled as $Power = CEq \times Vcc^2 \times F$ where $CEq$ is the total capacitive load, $Vcc$ is the supply voltage and $F$ is the switching frequency. In this Trojan, a series of output pins are switched at 50MHz causing extra capacitive loads by driving the output pins. Also, since these are driven at such a fast rate, extra power is dissipated when compared to the reference design and the FPGA heats up. To communicate the key, we represented a ‘0’ as the temperature of normal operation and a ‘1’ as the temperature of the heated up operation. A large counter allows the shifting of the bits of the key to occur slowly, on the order of a minute, and provides ample time for the FPGA to heat up and cool down. By sampling the temperature of the FPGA at defined intervals, the key was obtained.

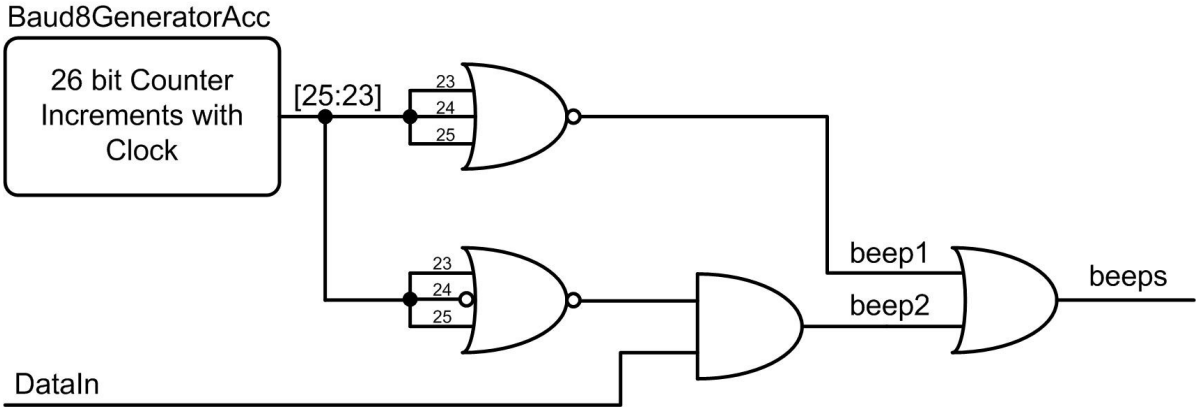



Figure A.9: Hardware to generate the beep pattern

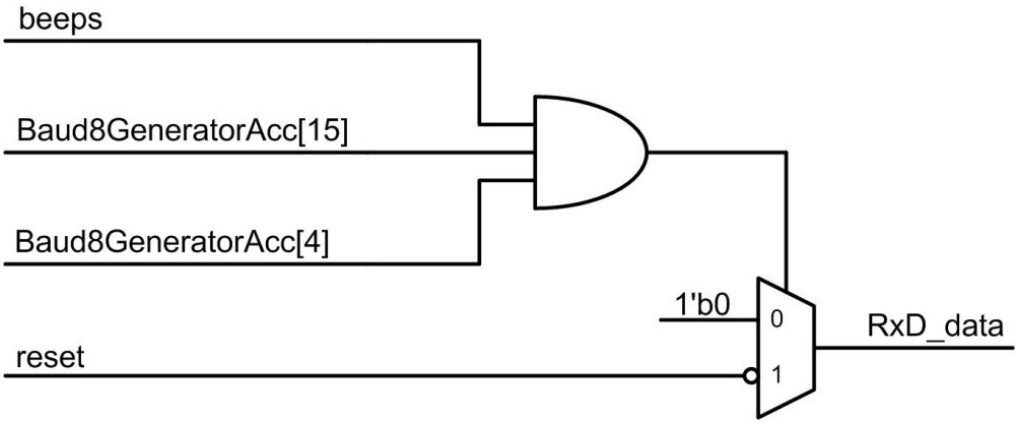



Figure A.10: Hardware to convert beeps to an audible tone

#### Results

By observing the temperature on the multimeter produced by the thermocouple at defined time increments, the key was observed. This method successfully leaked out the key allowing all of the bits to be verified with the expected result.

### AM Transmission

#### Explanation

By modulating a pin on the FPGA, an RF signal can be generated and used to leak the key. For the RF attacks, we utilized one of the pins of the socket. Since this socket is perpendicular

to the ground plane of the board, it creates a far better antenna than the expansion header pins. To allow this attack to be verified without any specialized equipment and also to demonstrate the range capabilities, it was performed at two different frequencies. One transmission occurs at 1560 KHz which can be received with an ordinary AM radio. The other attack transmits at 50 MHz and requires a specialized radio, such as a HAM radio, to receive the signal. The AM transmission has an extremely short range, on the order of inches, as compared to the 50 MHz transmission which can be received over 4 feet away. In both cases, if the pin was touched with a finger or paper clip, the transmission range increased by several orders of magnitude. Since the two attacks are effectively the same, the AM attack will be described in detail and only the differences with the 50 MHz variant will be subsequently pointed out.

The data to be carried by the AM signal needs to be easily interpreted by a human. We utilized a beep scheme where a single beep followed by a pause represents a '0' and a double beep followed by a pause represents a '1'. Figure A.9 shows the hardware necessary to generate the sequence of beeps based on an input value. After the counter recycles, a shift register is signaled to shift in the next bit of the master key. As can be seen in the figure, the top three bits of the counter are used to form eight sequential states. The first state is always a beep, followed by the second state which is always a pause. The third state will generate a beep only if the data input is a one. The remaining states are used to generate the long pause between beeps. For a person to be able to hear the beeps, the beep signal needs to be converted into an audio tone and then modulated. Figure A.10 shows the logic to accomplish this. When the beep line from Figure A.10 is a '1', it is ANDed with bits fifteen and four of the counter. Bit four toggles at a rate of 1560 KHz, the AM carrier, and bit fifteen toggles at a rate of 762Hz, the audible tone. After this AND gate, a mux is used to decide if the transmitter is enabled.

**Results**

We viewed the signal generated by this Trojan with a standard oscilloscope, Figure A.11. Figure A.11a shows the beeping sequence, where two beeps represent a '1' and a single beep represents a '0'. Figure A.11b represents the beginning of the 760Hz audio tone. Finally,

Figure A.11c shows the RF carrier wave.

### 50MHz Transmission

#### Explanation

The 50MHz transmission was implemented to demonstrate how much the range can be increased by increasing the frequency. We were able to decode the key from over 4 feet away. This increased range has to do with the length of the pin being used to transmit. As this pin length becomes closer to 1/4 wavelength of the carrier, the radiation pattern becomes better and the amount of radiated energy increases. If we transmitted at even higher frequencies such as 300MHz, the distance should increase, but will be capped as we approach microwave frequencies due to the uncorrected parasitics on the board. Instead of using bit 4 of the baud rate counter, this Trojan uses the 50MHz board clock to modulate the data.

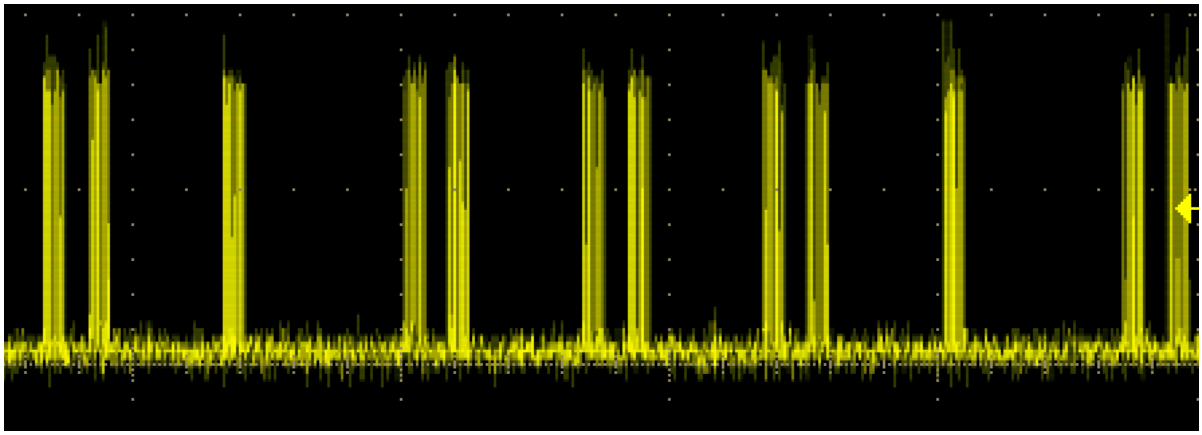
#### Results

When this Trojan was viewed with an oscilloscope, the results looked almost identical to Figure A.11 except for the change in frequency. Using a HAM radio, the signal was audibly picked up while standing in the room.

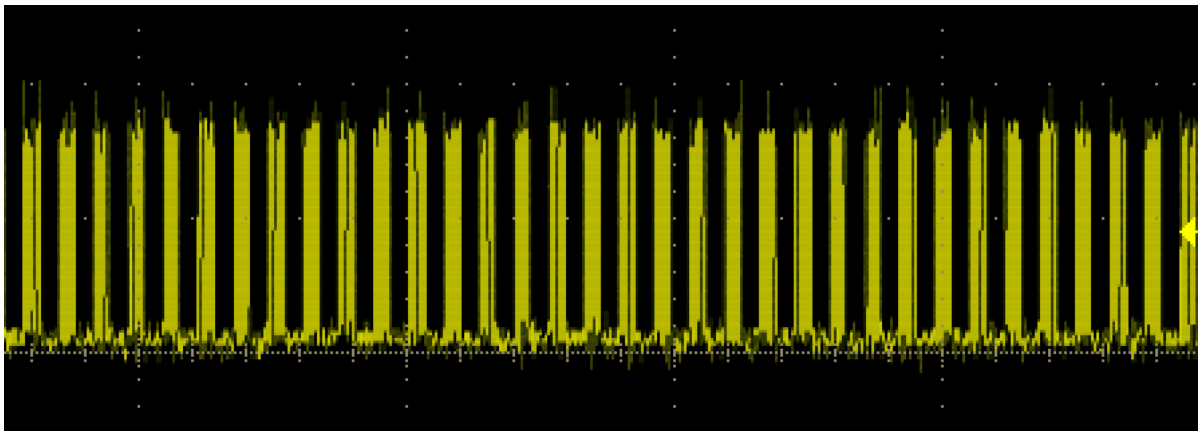
### LED Transmission

#### Explanation

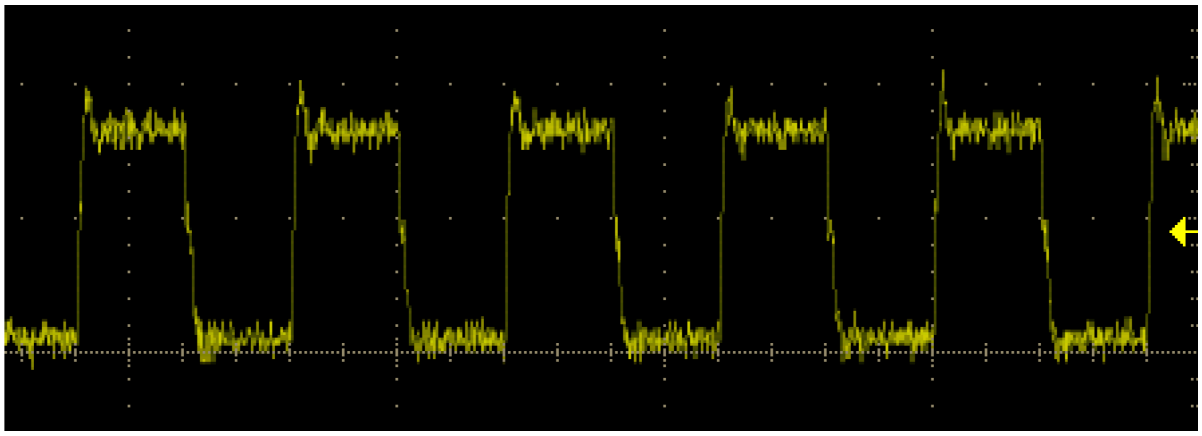
The same beep pattern as the RF Trojan was used to leak the key with a high frequency blinking LED. This time, instead of using an audible tone, two different blink rates were used. The LED blinks at 1 KHz to indicate a beep and at 2 KHz to indicate silence. In order to make it less noticeable to the user, the LED constantly blinks at a high frequency preventing the intensity from changing when it is switched into transmission mode. The difference between the LED blinking at 1 KHz and 2 KHz is not noticeable to the human eye simply by observing the LED. A separate circuit with a photodiode and band pass filter was then used to detect



(a) Beeping Pattern



(b) 760Hz audio tone



(c) RF carrier wave

Figure A.11: Measured patterns for RF signal leakage attack

the blinking pattern. Once detected, the circuit turns its own LED which blinks exactly as the beeps are heard by a person using the RF Trojans.

#### Results

This Trojan requires a specialized circuit to view the transmission by shifting the frequencies in the LED. This can be seen by placing the circuit near the LED that is transmitting and reading the values by observing the LED of the external circuit. Using the external circuit held a few inches from the Alpha device, the key was received.

## Considered Attacks

The following Trojans were considered, but not implemented in the final submission. For some of these Trojans, steps were made to implement them, but either their prohibitively high difficulty, infeasibility, potentially destructive nature or our lack of time and resources prevented us from implementing them. The following short descriptions provide a high-level conceptual approach to the these attacks.

### Keyboard LED

#### Explanation

The Keyboard LED Trojan is similar to the LED transmission but follows a different protocol. Since the transmission of information between the keyboard and the host device is relatively slow, it is not possible to blink the keyboard LED at a rate fast enough to be invisible to the eye as Trojan 6.2 does. Instead, a different scheme must be used where one of the LEDs is kept lit for the majority of the time and then only momentarily turned off and back on again. This could be used to transmit data where the light, sampled at defined intervals, transmits binary data. Preliminary tests showed that it was possible to flicker a keyboard LED such that it was almost impossible to see when staring directly at it and would be less noticeable when not paying specific attention to the LED.

### Blinking Cursor

#### Explanation

Similar to the above LED Trojans, the cursor on the VGA screen blinks at a predefined rate. This rate could be slightly altered such that it leaks information. By varying the rate at which the cursor blinks, the key could be extracted while still remaining hidden from the user. This attack would require access to the device, either if the Alpha was captured or if a malicious user was in close proximity to the device. This Trojan is a bit stealthier than having to physically access the FPGA (to extract the temperature information, for example) since the VGA monitor is an external device that is meant to be visible to the user.

### VGA Sync

#### Explanation

Because of how the horizontal and vertical syncs are generated by the BASYS VGA controller, a small gap is created. The gap can be used to hide data in the VGA signal such that it does not affect the output as displayed on the monitor. A specialized decoder circuit that had access to the VGA signal would need to extract the information from the small gap to recreate the key.

### Change Bitstream on PROM

#### Explanation

The Cypress chip on the BASYS board is used for implementing the Universal Serial Bus (USB) protocol and incorporates a fully programmable microcontroller. Because USB can be used to program the FPGA the Cypress chip has connections to the Erasable Programmable Read Only Memory (EPROM) used to store the FPGA configuration. A Trojan implemented on the Cypress chip using the embedded microcontroller could erase the EPROM memory after some time delay causing a permanent DOS attack that can only be fixed by reprogramming the entire board. To implement a Trojan on the Cypress chip, firmware must be downloaded

through the USB and can be saved in the EPROM for the Cypress chip. A software virus could be developed that will install the Trojan when the BASYS board is connected by USB to an infected computer.

### Destruction

#### Explanation

Another idea for a DOS attack was to physically destroy the device when triggered. Attempts were made to get the device to self-destruct, but none were successful. The means to this attack varied significantly, and included generating enough heat to be damaging, creating HDL code or tweaking the Xilinx ISE synthesis settings such that it would generate a bitstream that was self-destructive. Some progress was made in generating heat and was successfully implemented as a key leakage attack, but not enough heat was able to be generated that would damage the FPGA. Other attempts were made to write self-damaging HDL code, but no feasible structure could be created that violated safe FPGA configurations. Although none of these destructive Trojans were implemented given the constraints of the CSAW contest, small gains toward their realization were made.

### Store One Bit

#### Explanation

Each DOS attack, save the destruction of the FPGA, is temporary in that a reset of the bitstream would disable the Trojan. The Trojan would then have to be retriggered in order to begin functioning, which might not be very challenging if it were on a timer for instance, but could be difficult if it took a more elaborate triggering mechanism to reinstate the Trojan. But, if one bit of persistent data could be stored in the Alpha, then that bit could be set upon successful triggering of the Trojan and then used to reinstate the Trojan upon each reset. Significant effort was put forth to find a way to store one bit of data, but no way was successfully found. Attempts were made to communicate with the PROM from the FPGA and also with the Cypress chip used for the USB which had its own PROM. It was discovered that it was not

Table A.1: The variance in power from the reference design for each of the states of the eight Trojans

| | Ref mA | T1 Δ | T3 Δ | T4 Δ | T5 Δ | T6 Δ | T7 Δ | T8 Δ |
|---|---|---|---|---|---|---|---|---|
| Reset | 146.4 | 0.4 | 0.6 | 0.6 | 65.4 | 0.7 | 0.7 | 0.8 |
| Init Min | 156.0 | 0.3 | 0.5 | 0.5 | 22.8 | 1.6 | 0.6 | 1.0 |
| Init Max | 185.0 | 0.4 | 0.5 | 0.6 | 22.5 | 0.7 | 0.7 | 0.7 |
| Encrypt | 144.7 | 0.0 | 0.0 | 0.0 | 00.0 | 0.0 | 0.1 | 0.5 |
| Transmit | 153.1 | 0.4 | 0.6 | 0.6 | 22.2 | 1.0 | 0.8 | 1.2 |

possible to write directly to the Cypress PROM, but instead, a USB device plugged into the USB port of the board was required in order to change that PROM.

## Judging

We presented our hardware Trojans at CSAW 2008 to a panel of industry and academic judges along with the other contest entrants. The judges evaluated the Trojans on the their use of power, variance in bitstream size, stealth, and novelty. Table A.1 shows each Trojan for the given states of the system and how much their power usage varies from the reference design. The measurements were made using a bench multimeter in series with the power supply. Within a 20 minute window, these values varied by 0.5mA from the same measurement a few minutes earlier. It should be noted that Trojan T5, Section 6.2, requires large amounts of power by the nature of its design. In order to heat up the FPGA, heat needs to be generated by consuming more power. The bitstreams for each of the designs augmented with the Trojans matched the size of the original reference design and each Trojan was well hidden in both the source code and by inspection of the operating device. Using these metrics, our team was awarded first place.

# BIBLIOGRAPHY

[1] Cyber security awareness week 2008. `http://isis.poly.edu/csaw/`, 2008. Accessed on 10/2008.

[2] Intellectual property (IP) challenges and concerns of the semiconductor equipment and materials industry. White paper, SEMI, 2008.

[3] Amr Abdel-Hamid and Sofiène Tahar. Fragile IP watermarking techniques. In *Proceedings of the Conference on Adaptive Hardware and Systems (AHS)*, pages 513–519, 2008.

[4] Amr Abdel-Hamid, Sofiène Tahar, and El Mostapha Aboulhamid. A public-key watermarking technique for IP designs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 330–335, 2005.

[5] Amr Abdel-Hamid, Sofiène Tahar, and El Mostapha Aboulhamid. Finite state machine IP watermarking: A tutorial. In *Proceedings of the Conference on Adaptive Hardware and Systems (AHS)*, pages 457–464, 2006.

[6] Sally Adee. The hunt for the kill switch. *IEEE Spectrum*, 45, May, 2008.

[7] Dakshi Agrawal, Selcuk Baktr, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan detection using IC fingerprinting. In *Proceedings of the Symposium on Security and Privacy (SP)*, pages 296–310, 2007.

[8] Yousra Alkabani and Farinaz Koushanfar. Active hardware metering for intellectual property protection and security. In *Proceedings of USENIX Security Symposium*, pages 291–306, 2007.

[9] Yousra Alkabani, Farinaz Koushanfar, and Miodrag Potkonjak. Remote activation of ICs for piracy prevention and digital right management. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 674–677, 2007.

[10] Defense Science Board. Task force on high performance microchip supply. `http://www.acq.osd.mil/dsb/reports/200502HPMSReportFinal.pdf`, 2005. Accessed on 05/2009.

[11] Andrew Caldwell, Hyun-Jin Choi, Andrew Kahng, Stefanus Mantik, Miodrag Potkonjak, Gang Qu, and Jennifer Wong. Effective iterative techniques for fingerprinting design IP. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 208–215, 2004.

[12] Rajat Subhra Chakraborty and Swarup Bhunia. Hardware protection and authentication through netlist level obfuscation. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 674–677, 2008.

[13] Rajat Subhra Chakraborty, Somnath Paul, and Swarup Bhunia. On-demand transparency for improving hardware Trojan detectability. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 48–50, 2008.

[14] Edoardo Charbon. Hierarchical watermarking in IC design. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*, pages 295–298, 1998.

[15] Peter Clarke. Fake NEC company found, says report. `http://www.eetimes.com/showArticle.jhtml?articleID=187200176`, 2006. Accessed on 04/2009.

[16] DARPA. TRUST in integrated circuits (TIC). `http://www.darpa.mil/MTO/solicitations/baa07-24/index.html`, 2007. Accessed on 01/2009.

[17] Jia Di. Trustable recognition of undesired threats in hardware (TRUTH) analysis tool, for analysis of pre-synthesis behavioral and structural VHDL designs. `http://comp.uark.edu/~jdi/truth.html`, 2009. Accessed on 06/2009.

[18] Jia Di and Scott Smith. A hardware threat modeling concept for trustable integrated circuits. In *Proceedings of the Region 5 Technical Conference*, pages 354–357, 2007.

[19] Whitfield Diffie and Martin Hellman. New directions in cryptography. In *IEEE Transactions on Information Theory*, pages 644–654, 1976.

[20] Digilent. Basys system board. `http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS`, 2008. Accessed on 09/2008.

[21] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Proceedings of Advances in Cryptology (EUROCRYPT)*, pages 523–540, 2004.

[22] Reouven Elbaz, Lionel Torres, Gilles Sassatelli, Pierre Guillemin, Michel Bardouillet, and Albert Martinez. A parallelized way to provide data encryption and integrity checking on a processor-memory bus. In *Proceedings of the Design Automation Conference (DAC)*, pages 506–509, 2006.

[23] Minxi Gao, Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Subarna Sinha, and Robert Brayton. MVSIS. In *Proceedings of the International Workshop on Logic Synthesis (IWLS)*, 2001.

[24] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the Conference on Computer and Communications Security (CCS)*, pages 148–160, 2002.

[25] Intellectual Property Protection Development Working Group. Intellectual property protection: Schemes, alternatives and discussion. Technical report, VSI Alliance, 2000.

[26] Jorge Guajardo, Sandeep Kumar, Geert-Jan Schrijen, and Pim Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 63–80, 2007.

[27] Jiawei Huang and John Lach. IC activation and user authentication for security-sensitive systems. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 76–80, 2008.

[28] David Hwan, Kris Tiri, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont, and Ingrid Verbauwhede. AES-based security coprocessor IC in 0.18- $\mu$m CMOS with resistance to differential power analysis side-channel attacks. In *IEEE Transactions on Solid-State Circuits*, pages 781–792, 2006.

[29] Open SystemC Initiative. Systemc 2.2 standard. `http://www.systemc.org`, 2009. Accessed on 06/2009.

[30] Yier Jin and Yiorgos Makris. Hardware Trojan detection using path delay fingerprint. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 51–57, 2008.

[31] Samuel King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. Designing and implementing malicious hardware. In *Proceedings of the Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[32] Sandeep Kumar, Jorge Guajardo, Roel Maesyz, Geert-Jan Schrijen, and Pim Tuyl. The butterfly PUF protecting IP on every FPGA. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 67–70, 2008.

[33] Andrew Kahng John Lach, William Mangione-Smith, Stefanus Mantik, Igor Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang, and Gregory Wolfe. Constraint-based watermarking techniques for design IP protection. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1236–1252, 2001.

[34] John Lach, William Mangione-Smith, and Miodrag Potkonjak. Fingerprinting digital circuits on programmable hardware. In *Proceedings of the International Workshop on Information Hiding (IH)*, pages 16–31, 1998.

[35] John Lach, William Mangione-Smith, and Miodrag Potkonjak. FPGA fingerprinting techniques for protecting intellectual property. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*, pages 299–302, 1998.

[36] John Lach, William Mangione-Smith, and Miodrag Potkonjak. Signature hiding techniques for FPGA intellectual property protection. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 186–189, 1998.

[37] John Lach, William Mangione-Smith, and Miodrag Potkonjak. Robust FPGA intellectual property protection through multiple small watermarks. In *Proceedings of the Design Automation Conference (DAC)*, pages 831–836, 1999.

[38] Jae Lee, Daihyun Lim, Blaise Gassend, Gookwon Edward Suh, Marten van Dijk, and Srinivas Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Proceedings of the Symposium on VLSI Circuits*, pages 176–179, 2004.

[39] Jie Li and John Lach. At-speed delay characterization for IC authentication and Trojan Horse detection. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 8–14, 2008.

[40] David Lie, Chandramohan Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, pages 178–192, 2003.

[41] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 168–177, 2000.

[42] Jean-Paul Linnartz and Pim Tuyls. New shielding functions to enhance privacy and prevent misuse of biometric templates. In *Proceedings of Audio-and Video-Based Biometric Person Authentication (AVBPA)*, pages 393–402, 2003.

[43] Keith Lofstrom, Robert Daasch, and Donald Taylor. IC identification circuit using device mismatch. In *Proceedings of International Solid-State Circuits Conference (ISSCC)*, pages 372–373, 2000.

[44] Shigenobu Maeda, Hirotada Kuriyama, Takashi Ipposhi, Shigeto Maegawa, Yasuo Inoue, Masahide Inuishi, Norihiko Kotani, and Tadashi Nishimura. An artificial fingerprint device (AFD): a study of identification number applications utilizing characteristics variation of polycrystalline silicon TFTs. In *IEEE Transactions on Electron Devices*, pages 1451–1458, 2003.

[45] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Testing techniques for hardware security. In *Proceedings of the International Test Conference (ITC)*, pages 1–10, 2008.

[46] Naveen Narayan, Rexford Newbould, Jo Dale Carothens, Jefrey Rodriguez, and Timothy Holman. IP protection for VLSI designs via watermarking of routes. In *Proceedings of the International Conference on ASIC/SOC*, pages 406–410, 2001.

[47] Tingyuan Nie, Tomoo Kisaka, and Masahiko Toyonaga. A watermarking system for IP protection by a post layout incremental router. In *Proceedings of the Design Automation Conference (DAC)*, pages 218–221, 2005.

[48] NSA. Trusted access program office. `http://www.nsa.gov/business/programs/tapo.shtml`, 2009. Accessed on 04/2009.

[49] Arlindo Oliveira. Robust techniques for watermarking sequential circuit designs. In *Proceedings of the Design Automation Conference (DAC)*, pages 837–842, 1999.

[50] Arlindo Oliveira. Techniques for the creation of digital watermarks in sequential circuit designs. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1101–1117, 2001.

[51] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. In *Science*, pages 2026–2030, 2002.

[52] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM*, pages 120–126, 1978.

[53] Jarrod Roy, Farinaz Koushanfar, and Igor Markov. Circuit CAD tools as a security threat. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 65–66, 2008.

[54] Jarrod Roy, Farinaz Koushanfar, and Igor Markov. EPIC: Ending piracy of integrated circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1069–1074, 2008.

[55] Semiconductor Industry Association (SIA). Global billings report history (3-month moving average) 1976 - March 2009. `http://www.sia-online.org/galleries/Statistics/GSR1976-March09.xls`, 2009. Accessed on 05/2009.

[56] Joseph Lieberman (U.S. Senator). Whitepaper on national security aspects of the global migration of the U.S. semiconductor industry. White paper, U.S. Senate, 2003.

[57] Ellen Sentovich, Kanwar Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai Alexander Saldanha, Hamid Savoj, Paul Stephan, Robert Brayton, and Alberto Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, EECS Department, University of California, Berkeley, 1992.

[58] Rahul Simha, Bhagirath Narahari, Joseph Zambreno, and Alok Choudhary. Secure execution with components from untrusted foundries. In *Proceedings of the Workshop on Advanced Networking and Communications Hardware (ANCHOR)*, 2006.

[59] Eric Simpson and Patrick Schaumont. Offline HW/SW authentication for reconfigurable platforms. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 311–323, 2006.

[60] Scott Smith and Jia Di. Detecting malicious logic through structural checking. In *Proceedings of the Region 5 Technical Conference*, pages 217–222, 2007.

[61] Ying Su, Jeremy Holleman, and Brian Otis. A 1.6pJ/bit 96% stable chip ID generating circuit using process variations. In *Proceedings of International Solid-State Circuits Conference (ISSCC)*, pages 406–407, 2007.

[62] Gookwan Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the International Conference on Supercomputing (ICS)*, pages 160–171, 2003.

[63] Gookwan Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. Efficient memory integrity verification and encryption for secure processors. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 339–350, 2003.

[64] Gookwan Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the Design Automation Conference (DAC)*, pages 9–14, 2007.

[65] Gookwan Edward Suh, Charles O'Donnell, and Srinivas Devadas. AEGIS: A single-chip secure processor. In *IEEE Transactions on Design and Test of Computers*, pages 570–580, 2008.

[66] Gookwan Edward Suh, Charles O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the AEGIS single-chip secure processor using physical random functions. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 25–36, 2005.

[67] Ilhami Torunoglu and Edoardo Charbon. Watermarking-based copyright protection of sequential functions. In *IEEE Transactions on Solid-State Circuits*, pages 434–440, 2000.

[68] Steven Trimberger. Trusted design in FPGAs. In *Proceedings of the Design Automation Conference (DAC)*, pages 5–8, 2007.

[69] Romain Vaslin, Guy Gogniat, Jean-Philippe Diguet, Eduardo Wanderley, Russell Tessier, and Wayne Burleson. A security approach for off-chip memory in embedded microprocessor systems. In *Transactions on Microprocessors and Microsystems*, pages 37–45, 2009.

[70] Ingrid Verbauwhede and Patrick Schaumont. Design methods for security and trust. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 672–677, 2007.

[71] Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. Detecting malicious inclusions in secure hardware: Challenges and solutions. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 15–19, 2008.

[72] Francis Wolff, Chris Papachristou, Swarup Bhunia, and Rajat Subhra Chakraborty. Towards Trojan-free trusted ICs: Problem analysis and detection scheme. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1362–1365, 2008.

[73] Francis Wolff, Chris Papachristou, David McIntyre, Daniel Weyer, and William Clay. An embedded flash memory vault for software Trojan protection. In *Proceedings of the International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 97–99, 2008.

[74] Saeyang Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, Microelectronics Center of North Carolina (MCNC), 1991.