# HARDWARE DIGITAL OBFUSCATION

Giorgi Basiashvili

Master Thesis, Tallinn University of Technology

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology

Giorgi Basiashvili 201616IASM

# Hardware Digital Obfuscation

Master Thesis

**Academic Supervisor**
Samuel Nascimento Pagliarini
PhD
**Academic Supervisor**
Zain UI Abideen
MsC

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Giorgi Basiashvili    201616IASM

# Riistvaraline digitaalne hägustamine

Magistritöö

**Juhendaja**
Samuel Nascimento Pagliarini
PhD
**Juhendaja**
Zain UI Abideen
MsC

Tallinn 2022

# Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

| | |
|---|---|
| Author: | Giorgi Basiashvili |
| Date: | May 9, 2022 |

# Annotatsioon

Viimased tehnoloogilised edusammud riistvaradisaini vallas võimaldasid meil luua keerukaid omavahel ühendatud süsteeme, mida kasutatakse tarbe- ja tööstustoodetes, seega on tegemist väärtusliku varaga, mida tasub kaitsta. Selle tulemusena kerkivad integraallülituse (IC) ökosüsteemis esile mitmed turvaohud. Nende hulgas on murettekitavad pöördprojekteerimise tavad, mille eesmärk on IC võltsimine, ületootmine või muutmine. Seetõttu on intellektuaalomandi (IP) kaitsemehhanismide arendamine kriitilise tähtsusega. Viimastel aastatel on eelnimetatud ohtude maandamiseks välja pakutud erinevaid tehnikaid, kuid ükski tehnika ei näi olevat piisav disaini hierarhia varjamiseks. Selline hierarhia hägustamise võimalus on eriti oluline korduvaid mooduleid sisaldavate kujunduste puhul. Selles lõputöös pakume välja uudse viisi selliste kujunduste häguseks muutmiseks, kasutades tavapärast loogikasünteesi. Kasutame disaini mitmekesisuse loomiseks mitmeid sünteesitööriistas saadaolevaid optimeerimisi. Meie turbeanalüüs, mille viis läbi DANA pöördprojekteerimise tööriist, kinnitab nende optimeerimiste olulist mõju hägustamisele. Paljude segaseks peetud projekteerimisjuhtumite hulgast võivad kasutajad leida valikuid, mis tekitavad väga väikeseid üldkulusid, ajades samas segadusse pöördprojekteerija töö.

# Abstract

Latest technological advancements in the field of hardware design allowed us the creation of complex interconnected systems, used in consumer and industrial products, thus, it is a valuable asset that is worth protecting. As a result, numerous security threats are emerging from untrusted players in the integrated circuit (IC) ecosystem. Among them, reverse engineering practices with the intent to counterfeit, overproduce, or modify an IC are worrying. Accordingly, development of intellectual property (IP) defence mechanisms are of critical importance. In recent years, various techniques have been proposed to mitigate the aforementioned threats, but no technique seems to be adequate to hide the hierarchy of a design. Such ability to obfuscate the hierarchy is particularly important for designs that contain repeated modules. In this thesis, we propose a novel way to obfuscate such designs by leveraging conventional logic synthesis. We exploit multiple optimisations that are available in the synthesis tool to create design diversity. Our security analysis, performed by the DANA reverse engineering tool, confirms the significant impact of these optimisations on obfuscation. Among the many considered obfuscated design instances, users can find options that incur very small overheads while still confusing the work of a reverse engineer.

# List of abbreviations and terms

| | |
|---|---|
| AI | Artificial Intelligence |
| BEOL | Back end of the Line |
| CPU | Central Processing Unit |
| DPA | Differential Power Analysis |
| EDA | Electronic Design Automation |
| FEOL | Front end of the Line |
| HPWL | Half Perimeter Wirelength |
| IP | Intellectual Property |
| IC | Integrated Circuit |
| PDF | Probability Density Function |
| PE | Processing Engine |
| RE | Reverse Engineering |
| RTL | Register Transfer Level |
| SPA | Simple Power Analysis |
| VLSI | Very Large Scale Integration |
| 3PIP | Third party Intellectual Property |

# Table of Contents

**List of Figures** vii

**List of Tables** viii

**1 Introduction** 1

**2 State of the Art** 3

2.1 Side channel analysis . . . 3

2.1.1 Attack Methods . . . 4

2.1.2 Defence Strategies . . . 5

2.1.3 Metrics . . . 5

2.2 Hardware Trojans . . . 6

2.2.1 Attack Methods . . . 7

2.2.2 Defence Strategies . . . 7

2.2.3 Metrics . . . 7

2.3 Counterfeiting and IP piracy . . . 8

2.3.1 Attack Methods . . . 8

2.3.2 Defence Strategies . . . 8

2.3.3 Metrics . . . 9

2.4 Split manufacturing . . . 9

2.4.1 Attack Methods . . . 9

2.4.2 Defence Strategies . . . 11

2.4.3 Metrics . . . 11

2.5 Reverse engineering . . . 11

2.5.1 Physical Design obfuscation . . . 12

2.5.2 Structural Design obfuscation . . . 14

**3 Methodology** 17

3.1 GPS Correlator Architecture . . . 18

3.2 Proposed synthesis based approach . . . 18

3.3 Optimisation strategies . . . 20

3.3.1 Clock gating . . . 20

3.3.2 Ungrouping . . . 21

3.3.3 Datapath Analytical . . . 21

3.3.4 Bubble Pushing . . . 22

3.3.5 Tighten Max Transition . . . 22

3.3.6 Retiming . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.4 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

**4 Results** **24**
4.1 Power-Performance-Area evaluations . . . . . . . . . . . . . . . . . . . . . . 24
4.2 Dana security analysis . . . . . . . . . . . . . . . . . . . . . . . . 27
4.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

**5 Conclusions** **33**

**Bibliography** **34**

**Appendices** **40**

**Appendix 1 - GPS Correlator Code** **40**

# List of Figures

1 The left column shows the goals of the attack, and the right column shows the location of the attacker. Source:[1] . . . . . . . . . . . . . . . . . . . . 3

2 Hardware trojan classification. Source:[13] . . . . . . . . . . . . . . . . . . 6

3 Anatomy of an integrated circuit. Source: [16] . . . . . . . . . . . . . . . . 10

4 Use of atypical doping to make apparent PMOS transistor realise a constant $V_{DD}$ output. Source: [18] . . . . . . . . . . . . . . . . . . . . . . . . . . 13

5 Transistor level diagram of NAND gate. . . . . . . . . . . . . . . . . . . . . 13

6 Simple example of logic locking. Source: [24] . . . . . . . . . . . . . . . . 15

7 Approaches to obfuscating a hierarchical design, from locking to design diversity. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

8 State diagram of GPS correlator. . . . . . . . . . . . . . . . . . . . . . . . 18

9 The methodology to evaluate the hierarchy of design in the context of reverse engineering. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

10 Ungrouping. Source:[34] . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

11 Architectural overview on DANA. Source:[8] . . . . . . . . . . . . . . . . 22

12 PDFs of the area ($\mu m^2$) . . . . . . . . . . . . . . . . . . . . . . . . . . 25

13 PDFs of the number of cells . . . . . . . . . . . . . . . . . . . . . . . . . 26

14 PDFs of the leakage power ($mW$) . . . . . . . . . . . . . . . . . . . . . . 27

15 PDFs of the dynamic power ($mW$) . . . . . . . . . . . . . . . . . . . . . . 28

16 Graph of the register group for baseline design . . . . . . . . . . . . . . . 30

17 Graph of the register group for the clock-gated design . . . . . . . . . . . 30

18 Graph of the register group for the retiming (delay) . . . . . . . . . . . . 31

19 Graph of the register group for the clock-gating and retiming (delay) . . . 31

20 Graph of the register group for the clock-gating and retiming (delay) with steered mode (Register size 10). . . . . . . . . . . . . . . . . . . . . . . . 32

# List of Tables

1 Number of unique designs generated by an optimisation strategy. . . . . . 24

2 Minimum and Maximum values of area, number of cells, leakage and dynamic power of the generated designs . . . . . . . . . . . . . . . . . . . 26

3 Percent increase/decrease in the baseline design and a variants generated with the corresponding optimisation technique . . . . . . . . . . . . . . . . 29

# 1. Introduction

Microprocessors have become an integral part of our everyday lives. Day by day electronics are getting smarter and smarter, we are already seeing smart appliances such as coffee machines or toasters, and even some light bulbs have integrated ICs so that they can be controlled remotely. These devices are designed with performance, power consumption, and cost in mind, accordingly security of these designs might often be overlooked [1]. Moreover, the tools needed for hacking the chips have become more available and accessible to everyone [2]. Meaning hardware designs are becoming more vulnerable to hardware threats, some of which include: Hardware Trojans, IP piracy and IC overbuilding, Reverse engineering (RE), and Side-channel analysis. Even though the security of a light bulb might not be critical, hardware security in safety-critical systems, such as automotive or medical devices, becomes increasingly important [2].

Most adversaries can be categorised into three main groups, depending on their capabilities and objective [2]. First are nation-states, which have unlimited resources and their objectives can range from verifying the correctness of their manufactured designs, i.e. no hardware Trojans were injected during the manufacturing process, to analysing enemy technology [2]. Second, are criminals who are looking for a payoff [2]. It can be either by IP piracy or monetizing by exploiting design flaws, namely through developing malware [2]. Third, thrill-seekers looking to disrupt something [2].

The number of in-house designed ICs is increasing daily, accordingly, reverse engineering and IP piracy are becoming one of the most pressing issues in hardware security [2]. With RE adversary can obtain information about the technology used in the device and intellectual property, thus he will be able to manufacture it and sell it for profit, or it can be used to detect design flaws and reduce the reliability of the design [1]. The primary defence mechanism against RE is design obfuscation, which has two different approaches: physical design obfuscation and structural obfuscation [1]. Obfuscation aims to hide the real functionality of the IC, by either camouflaging the cells to look alike, in such a manner that they can be mistaken for each other and thus result in incorrectly extracted functionality or insert additional logic to lock the design so that it will produce correct output only if the correct key is applied [1]. It should be noted that any IC can be reverse engineered to the desired level of abstraction, granted that enough resources and time are provided. Accordingly, the main goal of the obfuscation is to delay the adversary [2].

With machine learning and AI becoming more popular, special AI accelerator chips are being developed. One distinct feature AI accelerators have over conventional ICs is the presence of multiple processing engines [3]. Instead of doing calculations on the CPU the presence of PEs allows concurrent execution using highly specialised circuits, which results in a significant performance increase [4]. Similar to AI accelerator's processing engines, GPS modules contain multiple copies of a correlator module. Since these circuits have the same module instantiated multiple times, by reverse engineering only one module adversary can detect all the copies of it. In addition to applying defence strategies, similar to camouflaging, on every module, therefore making it harder to RE, we can disguise modules so that it is not obvious that they are the same. This way, instead of analysing only one module, the adversary would have to analyse each module separately, thus dramatically increasing the time and effort needed for RE.

In this thesis, we will be discussing the structural obfuscation technique, which focuses on circuits that contain multiple copies of an entity and tries to increase the effort needed to learn the complete function of the circuit, and a case study of a GPS correlator module. Accordingly, we will assume that through complex imaging and delayering techniques, the adversary was able to correctly extract the complete netlist and has to examine it. Extracted netlist and structural isomorphism can be used to reverse engineer the functional unit [5]. To reverse engineer the functionality of an unknown unit, it can be compared and matched against a library of components with known functions [6, 7]. Accordingly, we will try to generate distinct designs of an entity, in this case, a correlator module for GPS, through different optimisation and implementation strategies. We will also analyse the overhead, in terms of area and power of generated designs.

First, we will discuss the state of the art. What are current threats and their corresponding defence strategies. Secondly, we will discuss the methodology used and thoroughly examine the optimisation strategies used. Then we will move on to the results that each of the strategies produced. Next, the results of the reverse engineering tool, DANA [8], will be discussed. Finally, we will evaluate the effectiveness of the proposed methodology.

# 2. State of the Art

Globalisation enabled the rapid development of ICs. Designing and manufacturing every component of the IC in-house is related to tremendous costs. Thus, during the production, IC might go through several third parties. Firstly, due to the ever-increasing complexity of circuits in most cases, some parts of the IC are designed in-house the rest is provided by the third party designers [1]. Next, the design is sent to a foundry, which fabricates the wafers [1]. Finally, before shipping, wafers are tested, either at the same foundry where it was produced or by a different company [1]. Nevertheless, due to the involvement of multiple parties, the risk of IP violations and hardware Trojans increases. However, depending on the adversary's goal and location, he might employ other strategies as well. Figure 1 vividly visualises strategies adversaries might adopt depending on the goal and the location.

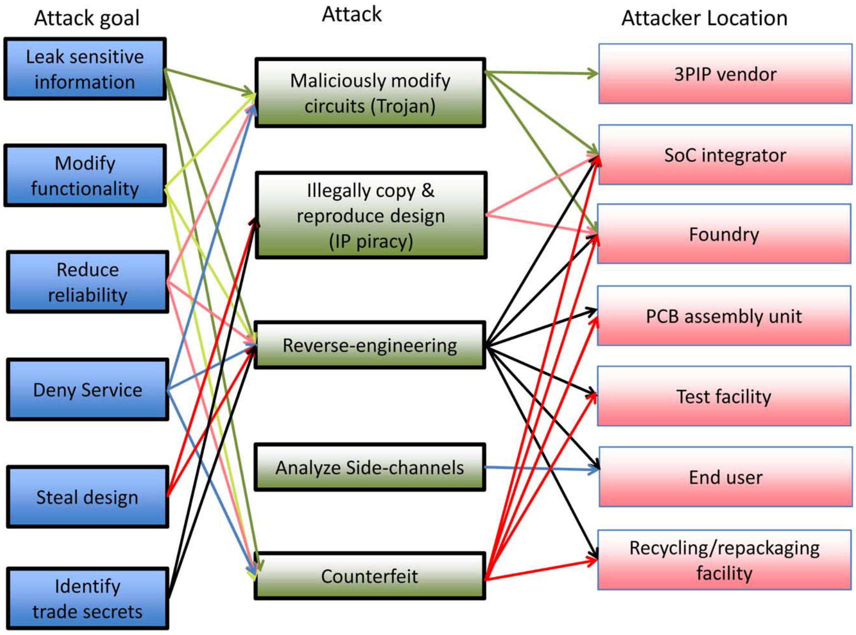


Figure 1. The left column shows the goals of the attack, and the right column shows the location of the attacker. Source:[1]

## 2.1 Side channel analysis

Side channel analysis is one of the most commonly used strategies in IC analysis. Very often it is used to retrieve secret keys, but can also be used to gain information about different aspects of the circuit. Traditionally, side channel analysis is related to measuring delays, power consumption, and electromagnetic radiation and retrieving secret keys by analysing measurements [9].

### 2.1.1 Attack Methods

Common attack strategies for side channel analysis are: Simple Power Analysis (SPA), Differential Power Analysis (DPA), and template attacks [10]. Moreover, with the advancements in ML and AI, numerous side channel analysis methods are being introduced incorporating different ML strategies.

One of the first attack strategies developed was simple power analysis and differential power analysis. Both of them involve recording the power consumption across time and some understanding of how the circuit was implemented, nevertheless, SPA is highly dependent on the adversary and how he interprets the data [10]. On the other hand, DPA, developed several years after SPA, introduced the model-based side channel attacks which allow the automation of the process. While in SPA adversaries manually analyse the power consumption traces in DPA traces are grouped into two and statistical methods are used to determine if they are different [10].

A more advanced and most commonly used attack strategy is template attack. Besides the device under attack, it also requires an identical copy of it, which can be fully controlled for experimentation, during which the adversary builds up multiple templates [11]. During the experimentation phase, a large number, several thousand for each operation, of power traces are recorded. Then mean values are calculated for each operation, which is used to select the points with large enough differences between each other, thus resulting in a multivariate distribution of the power signals of the selected points [11]. Additionally, a Gaussian distribution can be applied to the selected points to further reduce the number of points and simplify the calculations [11]. Finally, the templates can be constructed by computing the noise covariance matrices for each pair of the components of the noise vectors for each operation [11]. After the experimentation phase, the adversary has templates consisting of mean signal and noise probability distribution for each unknown value of key bits and can analyse the traces from the device under attack. Since it was assumed that we are dealing with a multivariate Gaussian probability distribution, the Bayes theorem can be used to estimate the key [11].

Advancements in AI and machine learning enabled researchers to move away from multivariate analysis and employ machine learning algorithms, which can deliver more accurate results [9]. Usually, when applying ML algorithms the complete dataset is split into two, the larger dataset, typically ranging from 65% to 85%, is used for training the model, whereas the smaller one is used for validating the predictions of the model. Numerous ML algorithms, such as Support Vector Machines, Random Forest, Rotation Forest, and MultiBoost, can be used to analyse the power traces from side channel [9]. However, the accuracy of the model not only depends on the algorithm and the parameters used, but also on the circuit and data traces it produces. Despite the long process of parameter tuning, which can have a significant impact on the accuracy of the model, ML algorithms are gaining traction and are employed more frequently in side channel analysis.

### 2.1.2 Defence Strategies

Primary defence strategies against side channel analysis are the introduction of noise in the system, by adding random operations that consume power, and leakage reduction, which decreases the dependence of secret key on power consumption [1]. Since most of the attack strategies assume Gaussian noise distribution, injecting artificial noise in the side channel can throw off the adversary. The simplest defence strategy involves performing extra multiplication operations, which increases the noise and thus reduces the dependence of the secret keys on leakage current [1]. However, adding dummy logic can have significant performance overheads and can be overcome by using a larger sample size and averaging the results [1].

Nonetheless, some attack strategies, like template attacks, require multiple devices, thus by using a large number of keys and changing them frequently the adversary might not be able to acquire identical devices, consequently, is unable to analyse the design [11]. Furthermore, introducing randomisation in the computation process, such as address and data scrambling, can play a significant role in mitigating side channel attacks [11].

### 2.1.3 Metrics

Core metrics of the side channel attacks are the amount of secret information that is available and the number of samples that are required to extract the secret keys [1]. Nevertheless, the correlation between the secret information and recorded data traces can be quantified with the side channel vulnerability factor [12]. Moreover, since template attacks are very powerful and common, some authors use them to evaluate the effectiveness of their proposed approach.

## 2.2 Hardware Trojans

Hardware Trojans are malicious circuits that can be integrated with a system to disable it when a set of parameters are matched or leak sensitive information [13]. As we can see from figure 2 Trojans have three main components, physical, meaning the modification to the circuit that is necessary for injection, activation, meaning when and under what circumstances Trojan gets activated, and finally action, does it transmit information or modifies the functionality of the system [13].

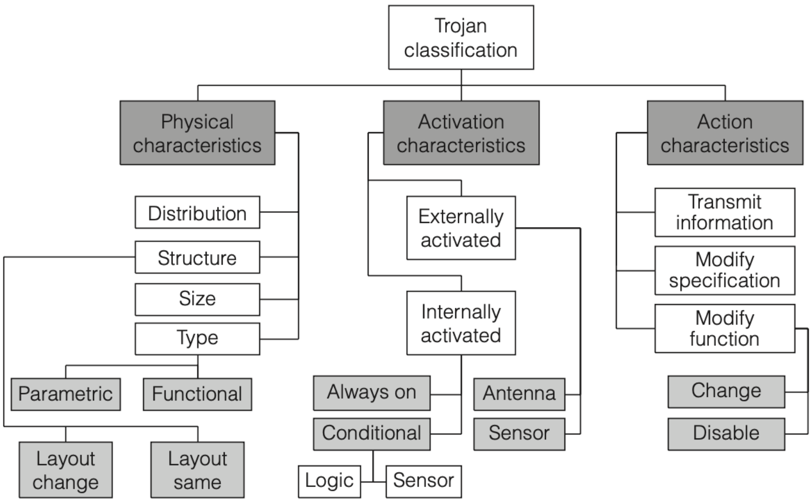


Figure 2. Hardware trojan classification. Source:[13]

Physical characteristics of the Trojan include distribution, structure, size, and type. Distribution refers to the physical location of the Trojan in the design [13]. Trojan placement can have a significant impact on the structure of the designs, which can force other components to be placed differently, thus resulting in a different layout [13]. It can either change the placement of several components or all of them. Size describes the number of added or removed components after injection [13]. Finally, type refers to whether gates and components were added or removed, which is called functional, or whether the Trojan was implemented using already existing gates and wires, called parametric [13].

Activation characteristics describe how a Trojan gets activated. It can happen via an external signal coming from either a sensor or an antenna. On the other hand, it can either be always active, or get activated if certain requirements such as a specific input pattern are met [13]. Always active usually means that some components have a higher failure rate than designed and to avoid detection by random signals this type of activation should be used on the paths that are rarely exercised [13].

Finally, action characteristics define what is the behaviour of the Trojan. Transmit information class describes Trojans that transmit key information to adversaries [13]. Modify specification refers to changing the characteristics of existing wires and gates to modify timings or delays [13]. Finally, a modification function refers to changing a functionality by adding or removing logic, which can result in a significantly different design [13].

### 2.2.1 Attack Methods

Due to the specifics of hardware trojans, they can not be inserted post-production, thus it can only be injected either during the design phase, by utilising third-party designs, which were either modified by the threat actor or were designed with the trojan in mind, and were overlooked during the design phase, or during the production phase in an untrusted foundry.

### 2.2.2 Defence Strategies

Trojan detection can be an issue since they are activated using a very specific signal, often based on external sources, moreover, due to the increasing complexity of designs detecting small malicious circuits in significantly larger designs can be troublesome [13]. Besides using test vectors and observing the output of the chip to verify the correctness of the design, side channel analysis could be used to detect Trojans.

Automatic test pattern generation is a standard tool in VLSI design, it uses the netlist of the circuit to generate input patterns and their corresponding outputs. Theoretically, it should be able to detect every modification to the circuit that is propagated to the output, nevertheless, in practice due to the large size of test vectors it is impractical [14]. Nevertheless, since most Trojans require changing the number of gates they should have more impact on the overall circuit than process variation and should be detectable with side channel analysis by applying random patterns at the input and measuring the power consumption [13].

### 2.2.3 Metrics

Several metrics exist to measure the effectiveness of injected Trojan. Specifically, the probability of detection, which is the relation between the number of detected trojans divided by the total number of trojans. Moreover, the ratio of falsely identified trojans over the number of trojan-free designs can also have useful information for researchers [1].

## 2.3 Counterfeiting and IP piracy

The goals of the adversary in IP piracy can range from identifying trade secrets to stealing the design [1]. In case of identifying trade secrets, adversaries might have to employ various reverse engineering techniques, which are discussed later in the chapter. Accordingly, in this section we will focus on stealing the design and using it as a black box, to produce and sell illegal copies of it. As we see from figure 1 adversary can be located either at the foundry or at SoC Integrator, in both cases adversary has access to the design, accordingly, defence strategies focus on either obfuscating the design to throw off the adversary or mark the design so that ownership can be claimed after production [1].

### 2.3.1 Attack Methods

As already mentioned, due to the adversary's location he has easy access to the design, nevertheless, if defence strategies such as split manufacturing or obfuscation are used, the adversary will have to use reverse engineering attack strategies to recover trade secrets.

### 2.3.2 Defence Strategies

Split manufacturing and logic locking are key defence strategies against IP piracy and IC overproduction [1]. Split manufacturing limits the access of the untrusted foundry by providing only partial design, which will be later assembled at a trusted facility. Accordingly, the adversary will have to put extra effort even if he is not interested in trade secrets. Similarly, with logic locking foundry is not provided with the activation keys, without which the IC does not exhibit desired properties. In both cases, the design can not be used as a black box system, and can not be illegally produced without extra effort [1]. Both of these strategies are discussed in detail later.

Unlike split manufacturing and logic locking, the goals of fingerprinting and watermarking strategies are proving the ownership of the design [1]. Both techniques can be applied during high-level, logic and physical synthesis [1]. Watermark is a uniquely encoded signature, which can be embedded into the IC during the design and implementation phases, by introducing additional constraints [15]. Imposed constraints may force the grouping of certain nodes and the formation of unique structures, which can be easily and unambiguously identified [15]. Nevertheless, additional constraints might alter the behaviour of the chip, accordingly, special pre- and post-processing steps should be performed on inputs and outputs to ensure that it does not interfere with the rest of the design [15]. Moreover, watermarks should be robust, meaning it should be impossible to remove the watermark without knowing the complete functionality of the circuit [1]. In addition to the watermarking, an additional unique identifier, fingerprint, can be inserted for each produced instance to identify the source of piracy [1].

### 2.3.3 Metrics

Split manufacturing and logic locking metrics include the hamming distance between the original netlist and the one predicted by the adversary. Metrics of the watermarking and fingerprinting are the probability of the generation of the same identifier for two different signatures, and the degradation of the quality of the solution [1].

## 2.4 Split manufacturing

One of the most efficient defence strategies against IP piracy, counterfeiting, and Trojan insertion is split manufacturing. To protect sensitive information from untrusted foundries, design can be split into two parts. The bottom layer where the transistors are built is called the Front end of the Line (FEOL), and the top layer, where the metal layers are built for routing, is called the Back end of the Line (BEOL) [16]. Figure 3 illustrates the anatomy of the circuit in terms of FEOL and BEOL. Note that BEOL can contain multiple layers of wiring which are denoted as MX, where X stands for the layer level. The FEOL layer contains transistors, as a result frequently require the use of a high-end foundry, whether it is trusted or not [16]. On the other hand, BEOL layers can be much simpler than the FEOL, thus they can be manufactured in the lower end but trusted foundries [16]. By splitting the design in two, the untrusted foundry has limited access to the design, and thus can not infer the complete functionality of the design, or inject trojans, moreover, it also limits their ability to counterfeit and sell illegal copies of the design. Nevertheless, an experienced adversary might still attempt to gain knowledge of BEOL layers if he possesses FEOL. It can be achieved with proximity, network flow, and SAT attacks [16].

### 2.4.1 Attack Methods

Modern EDA tools often place connected elements close to each other to reduce power and area, thus providing several hints for the adversary. Proximity attack assumes that input-output pairs of different cells are placed in close proximity to one another [16]. For each input pin on the FEOL layer, a set of candidate pins are selected. Candidate pins should satisfy several conditions. Specifically, inputs should be only connected to a single output, thus effectively eliminating all input-input connections and one to many connections [16]. Moreover, since combinational loops are only used by specific structures

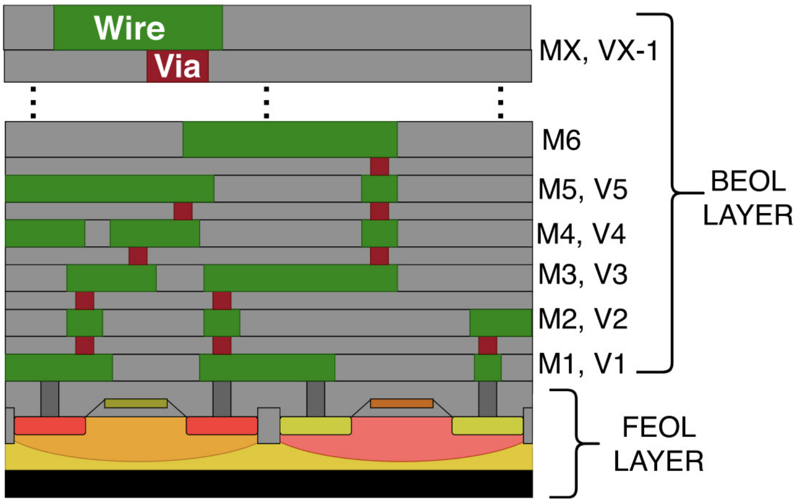


Figure 3. Anatomy of an integrated circuit. Source: [16]

that can be easily identified, they can also be excluded [16]. Accordingly, it is assumed that connecting input pins to the closest output pins that satisfy the aforementioned constraints should be the correct connection. While this attack strategy works perfectly for small designs, consisting of several thousand gates, it becomes extremely unreliable once the design size increases [16].

Shortcomings of the proximity attack are overcome by a network flow attack. In addition to constraints utilised in the proximity attack, the network flow attack also considers the load capacitance, meaning, specific cells have a certain maximum load capacitance, thus only connections satisfying the constraint can be considered as the candidate. Besides load capacitance, timing constraints, which can be guessed from the clock speed, for each connection are also considered to further reduce the number of candidate pins [16]. Finally, the common practice of connecting the source pin to the sink node along the direction of the sink node is taken into account and all other connections are excluded from the list [16]. However, unlike proximity attacks, input pins are not directly connected to the closest output pins, instead directed graphs are constructed and the min-cost network flow problem is solved with the Edmons-karp algorithm [16]. While the network flow attack can retrieve the correct BEOL layer from larger circuits than the proximity attack, it is still unable to retrieve correct connections for large circuits [16].

Finally, the boolean satisfiability (SAT) based attack has been developed. This, unlike previous methods, places key-based multiplexers which allow connection between every node of the FEOL layer and does not result in the cyclic path. As a result, a key-based locked circuit is generated, which can be solved with SAT attack [16]. Thus, an equivalent circuit of the original design can be retrieved.

### 2.4.2 Defence Strategies

As we already saw in the previous section, most of the attack strategies use proximity information to retrieve the BEOL. Accordingly, proximity perturbation and wire lifting strategies have been developed to challenge these types of attacks.

Proximity perturbation is the group of defence strategies that aim to decrease the number of hints left from the EDA tools. Common strategies include swapping pins so that the hamming distance between the outputs of the original netlist and the modified one is 50% [16]. This way enough distinction is introduced in the circuit to throw off the adversary utilising proximity attacks [16]. Besides pin swapping, placement and routing perturbation can be used, which modifies the netlist to introduce randomisation in the design [16]. It involves randomly swapping input and output pins, placing and routing the modified design, and finally correcting the swapped pins in the BEOL layer [16]. Netlist randomisation offers a great defence against proximity attacks, nevertheless, it is still vulnerable to SAT based attacks.

Another type of defence strategy is wire lifting. The goal of wire lifting is to reduce the number of connections on the FEOL layer and move them to the BEOL layer, thus making it harder for the adversary to find correct connections [16]. An important constraint of this strategy is manufacturability. To achieve a high yield, via and wire density should be considered. Accordingly, lifting every connection from FEOL to BEOL is impossible, nevertheless, by prioritising and lifting wires that have significant logic differences from the neighbouring cells, and the wires that would easily produce wrong outputs if misidentified, enough connections can be lifted to make proximity attacks impractical, thus boosting the security [16].

### 2.4.3 Metrics

No single metric is developed to assess the effectiveness, of a split manufacturing defence strategy, nevertheless, the most common metrics include the number of correctly predicted BEOL connections and the hamming distance, between the original netlist and the one predicted by the adversary [1].

## 2.5 Reverse engineering

As we can see from figure 1 reverse engineers might have several intentions. Accordingly, the desired abstraction level will vary. Assuming the adversary wants to learn the full functionality of the circuit, the reverse engineering process will consist of two primary stages. First, delayering of the IC and netlist extraction. Second, netlist analysis and functional unit extraction. During the first phase, the physical IC is delayered and an image of each layer is taken, which is used to extract the netlist [1]. Since we can not stop the adversary from delayering the IC, defence strategies focus on obfuscating the physical characteristics of the transistors, in order to make cells look like each other and infer different functionality. These types of defence strategies are called physical design obfuscation. Nevertheless, if the adversary is located at the untrusted foundry, he already possesses the netlist and thus can entirely skip the first phase. Nonetheless, during the second phase, the netlist is analysed to learn the functionality of the IC. Defence strategies targeting the second phase are known as structural design obfuscation.

### 2.5.1 Physical Design obfuscation

As already mentioned, physical design obfuscation focuses on the netlist extraction phase, accordingly, it concentrates on circuit element modifications that are difficult or impossible for an adversary to detect. Common steps for these types of attacks are delayering, imaging, and netlist extraction. Delayering refers to the process of removing each metal and dielectric layer one by one. It can be achieved either chemically, where special chemicals in precise dosage are applied to each of the layers to diffuse it, or mechanically, which involves rubbing tools to smoothen and remove excess surface. During the delayering process, various microscopy techniques, such as focused ion beams, can be used to image the specifics of the layer [17]. Finally, functional analysis can be used to extract the functional units from the gate-level netlist, which can be easily recovered from images [6]. Common obfuscation strategies focus on transistor property alterations in order to create stuck-at faults, delay faults, or stealthy signalling, to confuse the adversary during transistor-level functional analysis [18].

Stuck-at faults, which are transistors that are either always closed or always open, and delay faults, which refer to the nodes switching faster or slower to change the sequential behaviour of the chip, can be achieved by modifying the doping strategy. Figure 4 shows a normal PMOS transistor on the left and a transistor with a different type of dopant on the right, which creates a short circuit between the source and the drain thus producing a stuck-at fault, hence effectively allowing us to mask a certain component and imply different functionality [18]. For instance, Figure 5 shows that by utilising stuck-at faults, specifically, if the Q2 transistor is always off and the Q4 transistor is always on, the B input of the NAND gate will always be high, thus it will be the equivalent of the inverter. The same strategy can be employed to make larger, more complicated circuits, and perform simple operations, thus making a great defence strategy due to the difficulty of dopant transistor detection. Moreover, by manipulating source/drain doping or channel doping we can also make transistors switch faster or slower, thus creating delay faults [18].

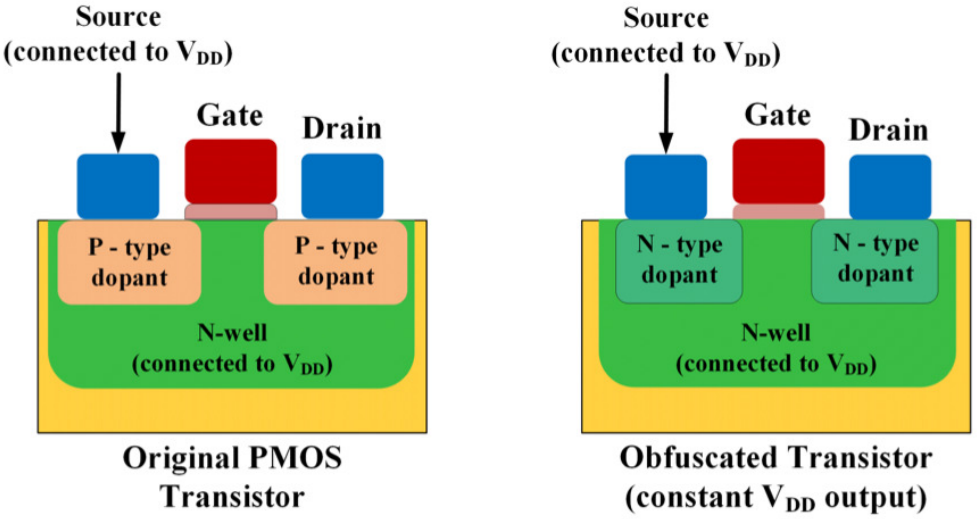


Figure 4. Use of atypical doping to make apparent PMOS transistor realise a constant $V_{DD}$ output. Source: [18]

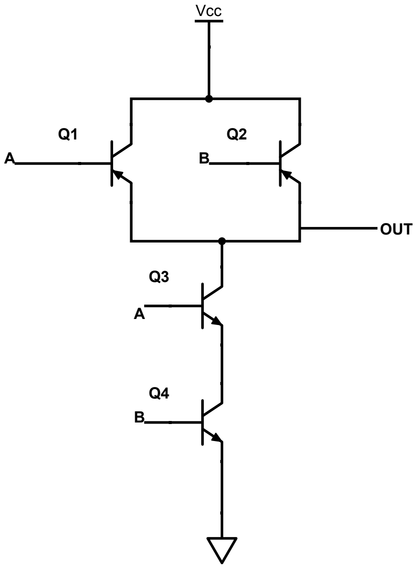


Figure 5. Transistor level diagram of NAND gate.

Even though these faults are hard to detect, it is not impossible. The adversary can use passive voltage contrast [17], or picosecond imaging circuit analysis [19], to detect dopant manipulation. However, due to the technicalities of these approaches, detecting every atypically doped transistor becomes slow and expensive, making it harder for an adversary to reverse engineer the circuit [18].

Finally, stealthy signalling refers to utilising cross talk to send signals. Usually, cross talk is undesirable, and various techniques have been developed to avoid it. Most notably, connecting metal fills to either VCC or the ground, nevertheless if instead it is connected to a clock or any other controlled signal, cross talk can be predicted and utilised to signal between neighbouring interconnects [18]. To further improve the signal strength, a thin interlayer dielectric can be used to increase the capacitive coupling between layers, resulting in increased crosstalk [18]. Similar to stuck-at and delay faults, stealthy signalling detection is extremely hard without a thorough examination of electrical properties, which is time-consuming when the ICs contain millions of transistors [18].

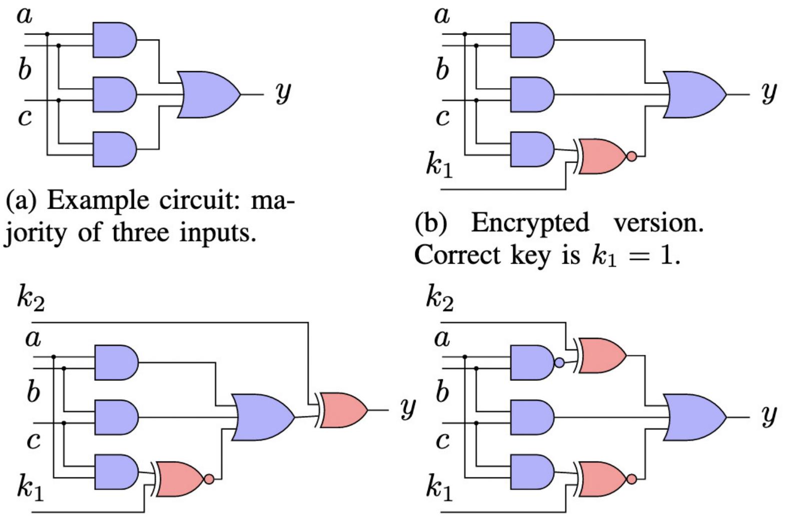
Since, physical design obfuscation requires the insertion of additional transistors or the metal fills, obfuscating every cell would result in significant size overhead and would tip the adversary. On the other hand, by obfuscating only the cells that easily propagate to the output, the adversary would first have to detect which cells were obfuscated [20].

Considering physical design obfuscation's goal is to hide some functionality from the adversary, the skills and resources of the adversary will play a major role during the reverse engineering process. Accordingly, quantifying the obfuscation would also require taking into account the skills of the adversary, which is impossible, thus there is a lack of quantitative approach to classifying the level of obfuscation. However, assuming obfuscated cells were detected but unidentified or partially unidentified, the number of brute force attempts required to recover the functionality can be used as a metric [20]. Additionally, the hamming distance between original and obfuscated designs can be used to quantify the level of obfuscation [20].

### 2.5.2 Structural Design obfuscation

While physical design obfuscation concentrates on making netlist extraction harder, structural obfuscation focuses on complicating the analysis phase of extracted design. Popular techniques include EPIC and its successor smart logic obfuscation. Both techniques add additional logic gates and memory elements to the combinational circuit, which are activated only when a certain key is used, accordingly, the circuit produces desired output only when the correct key is applied [21, 22]. A similar approach can be taken with sequential circuits, however, additional states are inserted in a finite state machine (FSM). FSM can have modified state transitions, invalid transitions from one state into another, duplicated states, or even black hole states, from which it will be unable to recover [1]. In all these scenarios, only the correct key produces the output, hence they are called key-based obfuscation strategies.

Key-based obfuscation techniques are mostly applied in the post-synthesis stage of circuit design. During which, XOR and XNOR gates can be randomly inserted throughout the design [21]. Figure 6 shows logic locking by adding XOR and XNOR gates. Even though randomly placed cells can be effective, they might be easily detectable by the adversary [23]. Accordingly, the gates should be placed constructively. Specifically, by assigning specific weights to the interconnect and placing them to maximise the summation of weights [23]. Although gate insertion is fast and effective, MUX-based logic locking also offers data flow path obfuscation [23]. By inserting multiplexers and connecting its wrong outputs to dummy logic [23]. On the other hand, during the pre-synthesis stage, control and data flow graphs, as well as binary decision diagrams, can be obfuscated and locked [23].



(a) Example circuit: majority of three inputs.

(b) Encrypted version. Correct key is $k_1 = 1$.

(c) Another encrypted version. Correct key is $(k_1, k_2) = (1, 0)$.

(d) Yet another encrypted version with key $(k_1, k_2) = (1, 1)$.

Figure 6. Simple example of logic locking. Source: [24]

Even though logic locking is very effective, if the adversary has access to the netlist of the circuit and an unlocked version of the IC, which in most cases can be obtained through the open market, key extraction becomes feasible with SAT attacks [24]. During SAT attack boolean satisfiability test is performed on the IC, to find the distinguishing inputs, which can rule out at least one wrong key, where the satisfiability is verified by the unlocked version of the IC utilised [24].

Nonetheless, even if one cannot obtain an unlocked version of the IC or the IC is designed to counter SAT attacks, by employing the ANTI-SAT circuit, which makes key extraction exponentially harder depending on the key size, keys can still be extracted with high accuracy using novel machine learning algorithms [25, 26].

Since key-based obfuscations aim at increasing the reverse engineering effort, required to recover keys, one of the main metrics of such type of obfuscation is the number of brute force attempts needed to recover secret keys and the hamming distance between correct and incorrect outputs of the circuit [1].

Another type of structural obfuscation is key-less obfuscation. Unlike key-based techniques, if the adversary possesses a netlist of the IC, he can still use it as a "black box" and sell illegal copies. Still, extracting functional units from the netlist should be harder. Such strategies include the insertion of cells that are connected to the rest of the circuit but do not have an impact on the output, thus creating more data the adversary has to analyse before extracting functional logic [18]. However, there is no quantitative method to measure the effectiveness of such strategies since it depends on the skills and the equipment of the adversary.

# 3. Methodology

There are multiple examples of designs that repeatedly instantiate the same module. Such as in the hardware implementation of neural networks, neurons that contain the multiply-and-accumulate type of functions are instantiated hundreds to thousands of times [27]. This common design style is also seen in cryptographic hardware accelerators that are round-based, such as the AES [28]. A generic representation of such a type of system is shown in Fig. 7 (top panel), where a notion of a shared bus that connects all the repeated elements is also introduced.

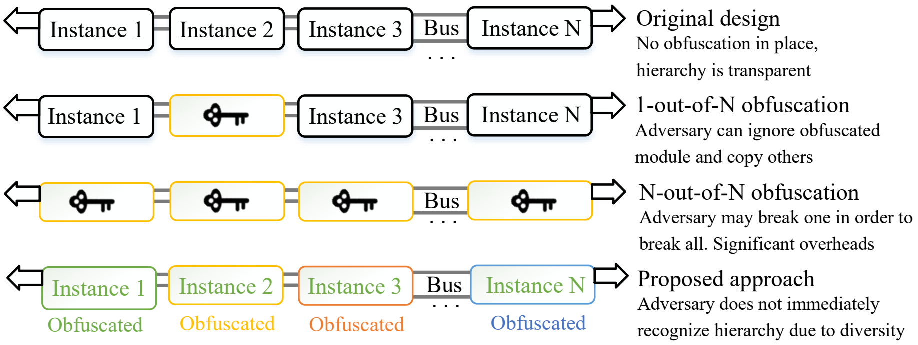

Instance 1
Instance 2
Instance 3
Bus
. . .
Instance N
Original design
No obfuscation in place, hierarchy is transparent
1-out-of-N obfuscation
Adversary can ignore obfuscated module and copy others
N-out-of-N obfuscation
Adversary may break one in order to break all. Significant overheads
Obfuscated
Proposed approach
Adversary does not immediately recognize hierarchy due to diversity

Figure 7. Approaches to obfuscating a hierarchical design, from locking to design diversity.

Next, assuming the system is an IP that is worth protecting against reverse engineering threats, one could take a state-of-the-art locking approach [29] and apply it to a single module (second panel). While this approach seems interesting at first – it would withstand known attacks such as SAT – a capable adversary would bypass the problem entirely by replacing the obfuscated module with one of the transparent ones. It follows then that all instances have to be obfuscated under a key-based approach (third panel). However, even if the approach appears to have merit, once a single module is broken, they may all be broken. It is also important to note that logic locking approaches are not overhead-free, the cost to obfuscate all $N$ modules can be rather large [30].

The illustrative example depicted in Fig. 7 is an attempt to demonstrate that current obfuscation practices have not sufficiently tried to hide the design hierarchy. The different colours on the bottom panel of the image try to convey this concept of design diversity. In the next subsection, we briefly discuss the architecture of the GPS correlator, a fundamental part of the GPS module. Afterwards, we introduce a synthesis-based approach to achieve slightly modified designs in a way that would make it harder for an adversary to notice the repeated instances.

## 3.1 GPS Correlator Architecture

This thesis is based on a case study of the GPS correlator hardware obfuscation. The objective is to develop a key-less structural obfuscation methodology that will be applicable to circuits that utilise the same module multiple times. One of such ICs are neural network accelerators which have multiple processing units. And due to the rise in popularity of AI accelerated applications, their security and IP protection becomes prevalent. GPS correlator is one of the integral parts of the GPS module. It receives a signal from the satellite and continuously auto correlates it. Since GPS uses data from at least 4 satellites, multiple correlators are needed. Moreover, an increased number of correlators can result in a faster signal acquisition, which can be crucial in some scenarios. The correlator used in this case study consists of combinational and sequential parts. The combinational part uses XOR gates to perform correlation, meanwhile, the control unit keeps track of incoming signals, offsets them, and stores them in registers to perform calculations. A state diagram of the control unit is presented in figure 8. Data from sensors are read during setting and loading states, then it is processed during deciding, dividing, offsetting, and running states, and finally outputted during idle, locked, and read rank states.

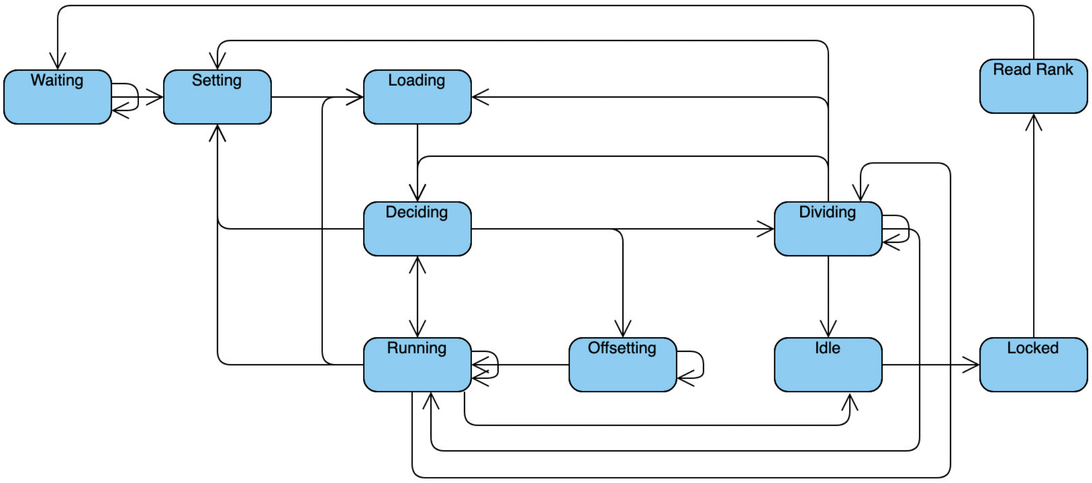


Figure 8. State diagram of GPS correlator.

## 3.2 Proposed synthesis based approach

ASIC design flow can be divided into two phases: logic synthesis and physical synthesis. During logic synthesis, the RTL description of the circuit is mapped into the netlist of standard library cells. Whereas during physical synthesis netlist is further optimised according to the placement, routing, and timing requirements. There are various optimisation techniques applicable during both phases of the design flow, which can have a significant impact on the overall design. Modern EDA tools like Cadence and Xilinx have integrated state of the art optimisation techniques, and allow the user to apply them without the need to manually implement them [31, 32]. Applying optimisation strategies during logic synthesis results in a different layout of the circuit and as already mentioned the goal of this thesis is to generate a circuit with multiple copies of an entity with minor differences in order to increase the effort needed to learn the complete function of the circuit. Accordingly, we used Cadence Genus and a Nangate 15nm open cell library to synthesise and optimise the correlator. Genus is highly configurable and offers state of the art optimization techniques. A total of 9 optimisation strategies were used:

1. Clock Gating
2. Ungrouping
3. Datapath Analytical
4. Bubble Pushing
5. Tighten Max Transition
6. Retiming for Delay
7. Retiming for Area
8. Clock Gating + Retiming for Delay
9. Bubble Pushing + Retiming for Area

Besides optimisation strategies, to generate different designs, we selectively excluded elements from the design. Specifically, the Nangate library contains 67 elements. And for each run, a single element was excluded from the design. For instance, in the first run, we excluded the AND2_X1 element, which is an AND gate with 2 inputs and drive strength 1. The simplest way the synthesiser can replace AND2_X1 is the use of an element with different drive strength or a different number of inputs, in both cases at least area and power consumption will be different from the baseline version. However, depending on the synthesiser it can also use any other element to implement the same functionality. In the second run, we returned the AND2_X1 but instead emitted the AND2_X2 element which is also an AND gate with 2 inputs, but it has a different drive strength. This way, each design will be marginally different from the previous one. First, this process was done without any optimisations, then it was repeated for the previously mentioned optimisations and their combinations. Ideally, it would produce:

$$(Numberofelements + 1) * (Numberofoptimisations) = 612$$

however, in some cases this difference might not be enough to alter the design significantly enough, accordingly, we have to check each design and verify that their characteristics, such as area, power, and critical path delay are different and eliminate the duplicate designs.

Fig. 9 illustrates our methodology that exploits the aforementioned techniques during the logic synthesis to evaluate the obfuscation of the design's hierarchy. The complete process is fully automated and scripted to enable a push-button analysis. We provide RTL description (i.e., Verilog or VHDL), timing constraint, and standard cell library of the targeted technology. We use the Nangate 15nm library of standard cells throughout the evaluation.

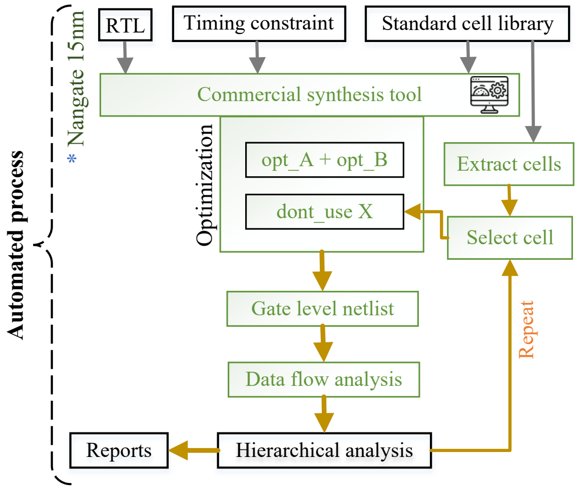


Figure 9. The methodology to evaluate the hierarchy of design in the context of reverse engineering.

## 3.3 Optimisation strategies

### 3.3.1 Clock gating

Clock gating is a popular power-saving technique. It refers to the activation of the clock signal in a specific part of the circuit only when there is work to be done [31]. Clock power can consume a significant portion of the total power usage, thus its optimisation can have a substantial impact on the overall power usage [31]. Clock power consumption depends on capacitance, voltage, and frequency, and clock gating helps us reduce switching capacitance, by avoiding unnecessary state updates to the components [31]. The simplest clock gating strategies include utilising logic gates such as AND or NOR gates [33]. By connecting one input of the NOR gate to the clock and the other to the enable signal we can effectively pass the signal only when both of them are zero and create a clock gating which is especially useful for positive edge triggering circuits [33]. Another popular clock gating strategy is the RTL clock gating, which identifies flip-flops with a common enable signal and uses it to control the clock enable signal of the flip-flops [31].

### 3.3.2 Ungrouping

Ungrouping is the process of merging sub designs into the parent design [34]. It is especially helpful for floor-planning [35]. Figure 10 brightly demonstrates the result of ungrouping. One of the ungrouping algorithms is MB*-tree. It has two main stages, clustering and declustering [36]. During the clustering phase, it groups modules based on area utilisation and connectivity with other modules [36]. After this comes the declustering phase, during which newly clustered groups are expanded and additional logic is shared between modules [36, 35]. It helps us reduce the area by sharing logic and reducing timing [34].

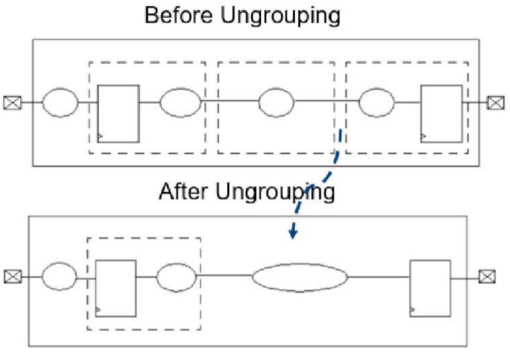


Figure 10. Ungrouping. Source:[34]

### 3.3.3 Datapath Analytical

Datapath structure can have a significant impact on the performance, due to which lots of high-performance VLSI designs datapaths are handcrafted [37]. However, due to the constantly increasing complexity of designs, manually optimising everything becomes infeasible. Thus, various analytical datapath optimisation techniques, such as HPWL have been developed [38, 39, 40]. Since HPWL works during the logic synthesis phase, its wirelength estimations are not accurate, especially in larger designs [38, 39]. If the whole optimisation process is divided into multiple parts, datapath performance can be significantly increased. Specifically, if the datapath circuit's connectivity regularity is extracted and used to evenly distribute them during the placement of logic blocks, then an optimal placement strategy can be achieved [38]. Nonetheless, many other datapath optimisation strategies exist which might work better in other circumstances, for instance they might specialise in latch placements. However, optimisation strategies utilised by

EDA tools are often proprietary [41].

### 3.3.4 Bubble Pushing

The practice of applying DeMorgan's law and duplicating logic to remove trapped inversions is called bubble pushing [42].

### 3.3.5 Tighten Max Transition

In larger systems where delay timings vary between different logic blocks, a single switching event can be propagated at different speeds, thus causing multiple switches which can cause the wrong value to be captured by a latch [43, 44]. Thus, constraining max transition times can have a significant impact on reliability and power consumption [43, 45]. One of the most common and efficient delay reduction techniques is buffer insertion [46]. An optimal number of inverters can be calculated based on the total resistance and capacitance of the interconnect line [46].

### 3.3.6 Retiming

Retiming is the process of reorganisation of memory elements in synchronous circuits [47, 48]. It can be focused either on minimising the delay, which is called retiming for delay or minimising the number of registers and thus achieving area savings, which is called retiming for area [47]. It was first introduced by Leiserson and Saxe in [49], and various papers are still published regarding numerous optimisations to the original algorithm [50].

## 3.4 Evaluation

Finally, a reverse engineering tool, DANA [51, 52, 8], was used to analyse the designs and evaluate the effectiveness of the method. As we can see from figure 11 depicting the architecture of the tool. There are three main stages: preprocessing, processing and evaluation. Processing and evaluation phases are run at least twice [8].

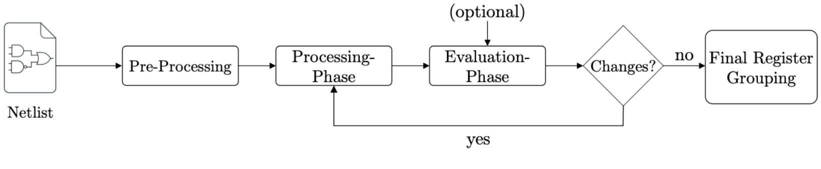


Figure 11. Architectural overview on DANA. Source:[8]

First, in the preprocessing phase, it parses through the netlist and identifies all flip-flops, and traces them until the next memory element is detected [8]. Thus, the flow of data between flip-flops is also identified. Next, each identified flip-flop is assigned to a unique group, which during the processing phase gets refined [8]. Each subsequent run of the processing phase takes the output of the previous evaluation phase as an input. Finally, during the evaluation phase, all possible groupings generated during the processing phase are analysed by taking into account the number of occurrences of the single group in the refined groups as well as the number of small groups, and a single register group is chosen [8]. If additional information on the register sizes is provided, later referred to as steered mode, a higher priority is assigned to the groups of specified size. As we can see from figure 11 output of the evaluation phase is used as an input to the processing phase and final grouping is generated once there are no changes detected between previous and current runs. The final output of the tool contains information about detected register groups, their predecessors, and successors. The number of detected registers and their sizes, as well as their predecessors and successors, should provide a good indication of the similarity between different designs.

# 4. Results

This section reports the results of our proposed methodology, based on the case study of a GPS correlator module. We used the RTL description of the GPS correlator and generated the results for a single design. We did not change the RTL of the design throughout the analysis for fairness. Recalling again, the objective was to develop a key-less and structural obfuscation methodology that would apply to circuits that have modules instantiated multiple times. Our proposed obfuscation is key-less and infers a little overhead, or almost zero. The performance does not impact the optimisation techniques. But, the area and power vary therefore it should be investigated. We have used a very relaxed clock frequency in order to allow the synthesis tool to make less constrained decisions.

## 4.1 Power-Performance-Area evaluations

A total of 509 unique designs were generated. Without any optimisations enabled it was able to generate 55 designs, the rest 12 cases generated one out of 3 already existing designs. Nevertheless, the number of duplicate designs varies depending on the optimisation strategy. Most of the unique designs were generated by Tightening Max Transition, Retiming for Delay, and its combination with Clock gating, however, it should be noted that ungrouping was not able to generate a single unique design. The detailed number of unique designs generated per optimisation technique can be seen in table 1

In table 2 we can see the minimum and maximum values of the area, number of cells,

| **Optimisation Strategies** | **Unique Designs** |
|---|---|
| **None** | 55 |
| **Clock Gating** | 50 |
| **Ungrouping** | 0 |
| **Datapath Analytical** | 56 |
| **Bubble Pushing** | 53 |
| **Tighten Max Transition** | 62 |
| **Retiming for Delay** | 61 |
| **Retiming for Area** | 53 |
| **Clock Gating + Retiming for Delay** | 62 |
| **Bubble Pushing + Retiming for Area** | 57 |

Table 1. Number of unique designs generated by an optimisation strategy.

and dynamic and leakage powers of the generated designs, as well as the optimisation technique used. Note that min leakage power was achieved by all optimisations except ungrouping, datapath analytical, and tightening max transition. Together with the normal distribution graphs 12, 13, 14, 15 below, where red dot represents the location of baseline design, gives us an idea about the overall results. Figure 12 shows that the baseline design is closer to the mean value. Almost half of the designs have less area as compared to the baseline design. This is the same for the number of cells as seen in figure 13. Similarly, the leakage power of the baseline design, figures 14, 15, are closer to the mean value. More than half of the designs consume more leakage power as compared to the baseline design. Regarding the dynamic power, the baseline design is far from the mean value and a large number of designs consume higher power as compared to the baseline design. It is noteworthy that we observe the change in the hierarchy of the structure, and the effect of the variation is reflected in the area, number of cells, leakage power, and dynamic power.

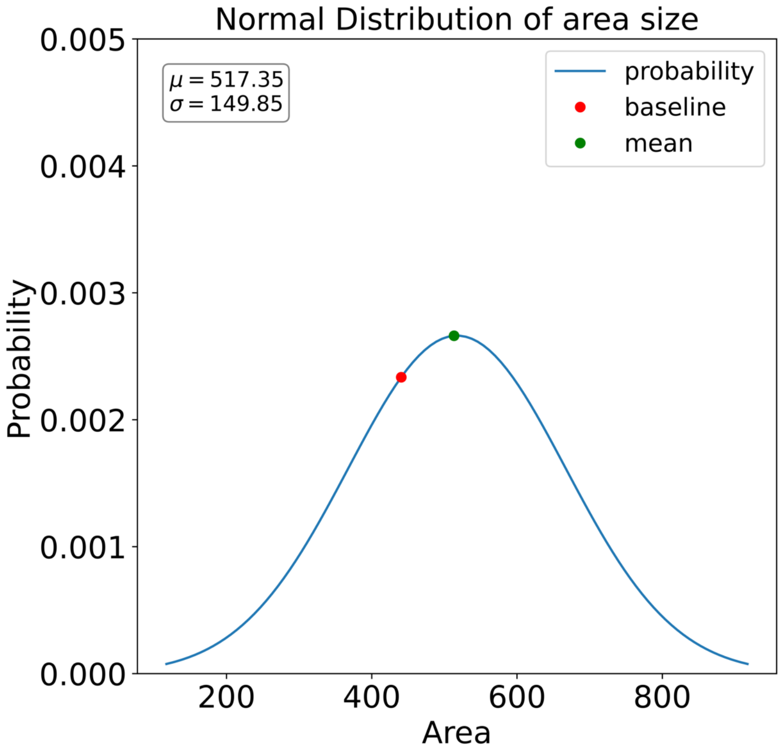


Figure 12. PDFs of the area ($\mu m^2$)

Next, we are going to observe the percentage increase and decrease of the area, number of cells, leakage power, and dynamic power. Table 3 lists the analysis of different overheads for their corresponding techniques. The first column lists the optimisation technique, the second column shows the percentage increase/decrease in the area, the third column shows the percentage increase/decrease in the number of cells, and the last two columns represent the leakage and dynamic power.

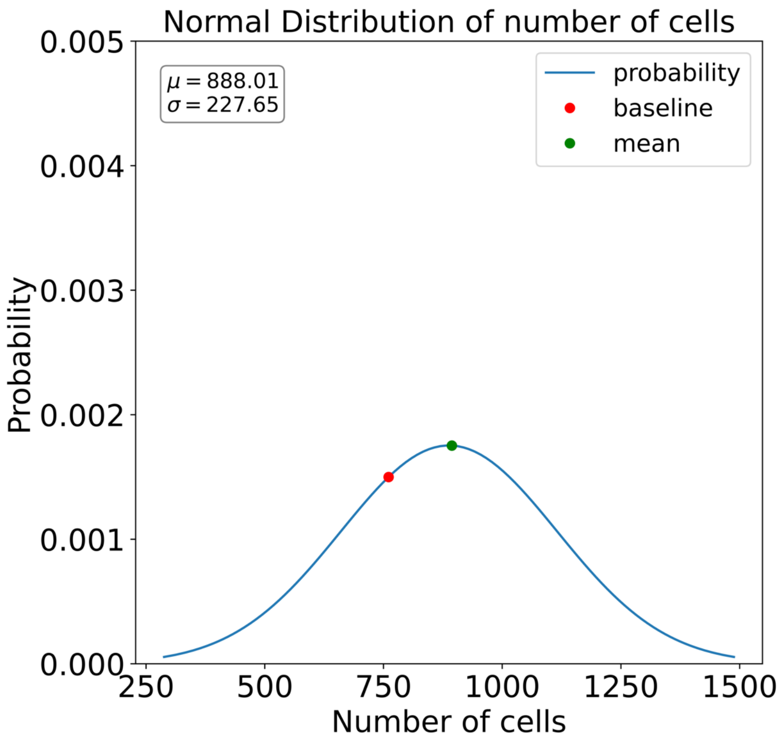


Figure 13. PDFs of the number of cells

| **Optimisation technique** | **Area** ($\mu m^2$) | | **Cells** | | **Leakage power** ($mW$) | | **Dynamic power** ($mW$) | |
|---|---|---|---|---|---|---|---|---|
| | **Min** | **Max** | **Min** | **Max** | **Min** | **Max** | **Min** | **Max** |
| **Baseline** | 432.9 | 461.6 | 762 | 845 | 0.012 | 0.013 | 2.368 | 2.414 |
| **Clock Gating** | 432.7 | 462.2 | 750 | 810 | 0.012 | 0.013 | 0.600 | 0.935 |
| **Ungrouping** | 432.9 | 461.6 | 762 | 845 | 0.012 | 0.013 | 2.368 | 2.414 |
| **Datapath Analytical** | 433.2 | 458.8 | 750 | 821 | 0.012 | 0.013 | 2.368 | 2.431 |
| **Bubble Pushing** | 437.1 | 459.1 | 758 | 840 | 0.012 | 0.013 | 2.293 | 2.469 |
| **Tightening max transition** | 598.8 | 990.3 | 1168 | 1644 | 0.022 | 0.062 | 0.706 | 2.091 |
| **Retiming for Delay** | 440.9 | 600.9 | 737 | 1047 | 0.012 | 0.018 | 2.944 | 4.398 |
| **Retiming for Area** | 434.7 | 458.8 | 767 | 840 | 0.012 | 0.013 | 2.399 | 2.460 |
| **Clock Gating + Retiming for Delay** | 425.2 | 560.9 | 711 | 966 | 0.012 | 0.018 | 1.084 | 1.562 |
| **Bubble Pushing + Retiming for Area** | 440.0 | 460.3 | 755 | 842 | 0.012 | 0.013 | 2.350 | 2.537 |

Table 2. Minimum and Maximum values of area, number of cells, leakage and dynamic power of the generated designs

We note that clock gating offers a significant decrease in dynamic power. Datapath analytical lowers area, cells, and dynamic power between 1-3%. The same is happening for the bubble pushing. Tightening max transition has a significant impact on the area, cells, and leakage power. But it shows a remarkable decrease in the dynamic power (84.3%). We should note that a large number of distinct designs were generated from this technique. The retiming for delay also has a similar behaviour for area and cells (20.18% and 12.71% increase) but it also shows an increase in the leakage and dynamic power. The combination of clock gating and retiming for delay shows an increase in every parameter except dynamic power, analogous to tightening max transition. The combination of bubble pushing and retiming for the area also shows a little increase/decrease in the parameters. In a nutshell, all the techniques have little impact on the area, cells, and power consumption

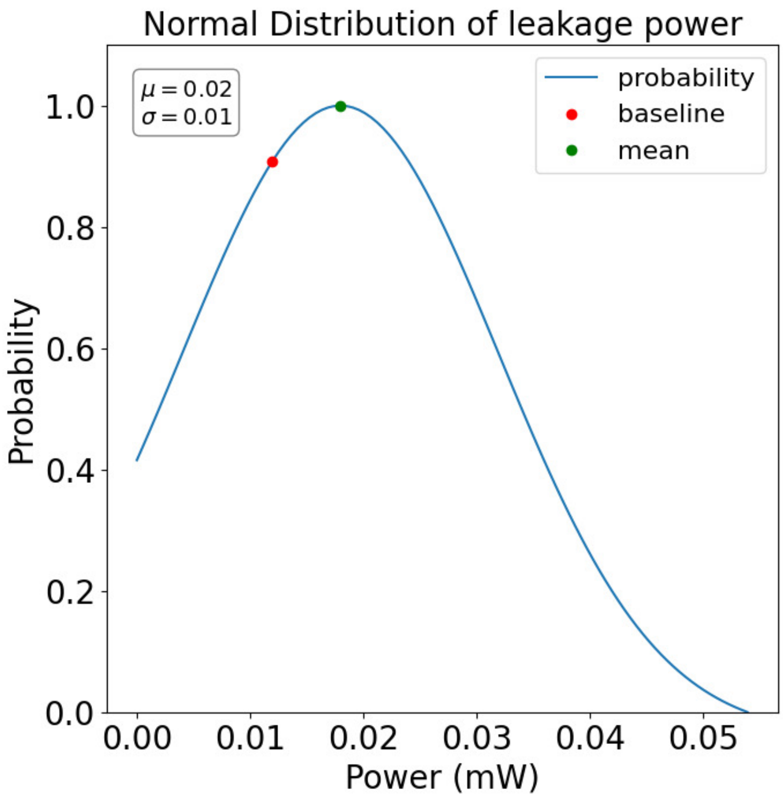


Figure 14. PDFs of the leakage power ($mW$)

except tightening max transition, retiming for delay, and a combination of clock gating & retiming for delay.

## 4.2 Dana security analysis

Dana was used in both steered and normal mode. In steered mode, register sizes of 10 and 1115 were specified. Pyvis [53] library was used to generate graphs from the results generated by Dana. It allows us to clearly see distinctions between different groupings. Each bubble on the graph represents a register group. The size of the bubble shows how many bits are in the register, the larger the size of the bubble the larger the size of the register, straight lines show the interconnection between different registers, and circular lines are connections from the register to itself. If we take graphs in figures 16, 17, 18, 19, based on normal mode 4 major clusters were identified by Dana. Based on this, we can conclude that there are 4 major types of designs to which all other designs converge to. Figure 16 shows the baseline design that includes different sized registers, but it still contains two registers with size 10, which can be a straightforward clue for an adversary to reconstruct the registers and their connected circuitry. Nevertheless, if we examine figures with optimisations enabled we can see that in figure 17, where we had enabled clock gating, we can see different 10-bit registers, however, a large number of registers are

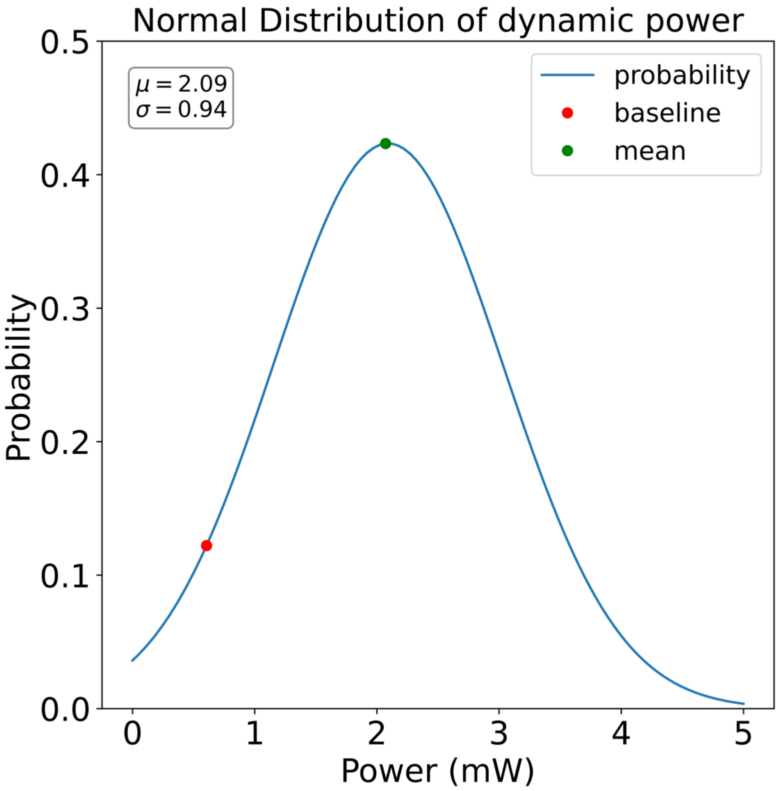


Figure 15. PDFs of the dynamic power $(mW)$

diminished and only a few of them with varying sizes are left. It allows us to clearly see distinctions between different groupings. In these unrolled designs, the sizes of the register do not correspond to the correct sizes declared in the RTL code. Fig. 18 shows the graph for the synthesised design with retiming for delay. Here, we can analyse that the complete graph consists of a large number of registers and we still can see a 10-bit register. This also offers a unique structure of the design and DANA is unable to map it in the same way. In the next example, shown in Fig. 19, we exploit two different optimisations (clock-gating and retiming for delay) at the same time. Again, we obtain a distinct graph. To summarise these results, we can confidently state that a design composed of many instances of the same module but each instance is synthesised differently, will present itself as a challenge to a reverse engineering adversary.

All these experiments presented so far were executed in the normal mode of DANA. Now, we exploit the steer mode of DANA with the register size of 10-bits as shown in Fig. 20. It is a fair assumption that from a non-steered mode, an adversary might reach the conclusion that 10-bit registers are present. We can see that the structure of the design is explicitly different from the previous ones. DANA still is unable to highlight clues even in steering mode. This implied a high level of obfuscation for the design. The adversary makes use of the different reverse engineering tools along with high skills but still, it requires an additional effort to correctly identify the design. Our applied optimisation

| **Optimisation technique** | **Area (%)** | **Cells (%)** | **Dynamic Power (%)** | **Leakage Power** |
|---|---|---|---|---|
| **Clock gating** | -0.3 | -0.7 | -119.4 | 0 |
| **Ungrouping** | 0 | 0 | 0 | 0 |
| **Datapath analytical** | -1.4 | -2.9 | -0.4 | 0 |
| **Bubble pushing** | -0.5 | -0.7 | -2.7 | 0 |
| **Tighten max transition** | +69.8 | +60.1 | -84.3 | +131.4 |
| **Retiming for delay** | +20.1 | +12.7 | +51.3 | +28.5 |
| **Retiming for area** | -0.5 | -0.3 | +0.7 | 0 |
| **Clock gating + Retiming for delay** | +18.9 | +13.6 | -66.5 | +34.4 |
| **Bubble pushing + Retiming for area** | +0.4 | -2.1 | +4.5 | 0 |

Table 3. Percent increase/decrease in the baseline design and a variants generated with the corresponding optimisation technique

techniques perfectly modify the structure of the design. This places barriers on DANA's clustering algorithm which incorrectly identifies the register group. This is the case for every optimisation technique.

## 4.3 Discussion

As already mentioned we can not prevent anyone from reverse engineering the circuit, nevertheless, we can delay it and make it an unattractive target. Moreover, since the skills and equipment of adversaries are diverse, we can not objectively and quantitatively measure how long it would take anyone to reverse engineer the design. As we saw in the previous section, leveraging logic synthesis and different optimisation strategies had a significant impact on the hierarchy of the design. Thus, the optimisation techniques are contributing towards obfuscation, to confuse the adversary to understand the architecture of design. Based on this we can definitely say that due to the high divergence of the generated designs adversary would have to analyse each of them separately, thus increasing the effort needed. Moreover, since our flow for the obfuscation is completely automated and does not incur high overheads, nor RTL changes, it becomes a highly attractive solution for circuit designers.

Although, it will vary from design to design, in our case study difference between designs' areas and cell counts difference were up to 80%, in terms of dynamic power up to 150%, and approximately 135% in terms of leakage power. Nevertheless, the average overhead in terms of area is 16.58%, in terms of cell count 15%, in terms of dynamic power 109%, and in terms of leakage power 40%. Even though average dynamic power is more than doubled than the baseline version, in applications where power consumption is a priority, tightening max transition and retiming for delay can be excluded from the list to meet the design goals and keep overheads under control. However, many other optimisations still remain attractive solutions.

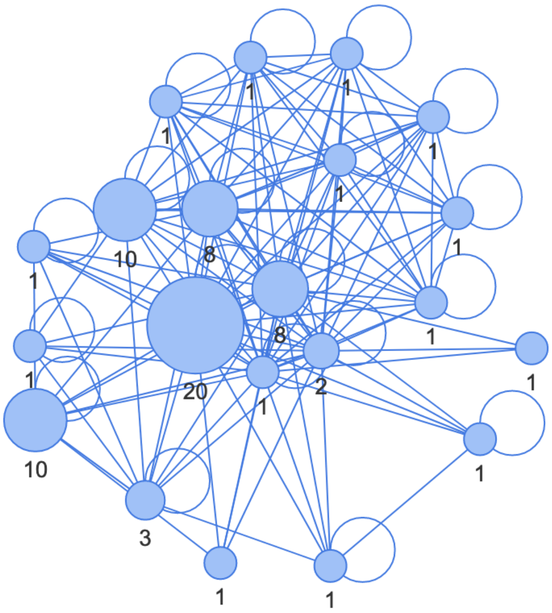


Figure 16. Graph of the register group for baseline design

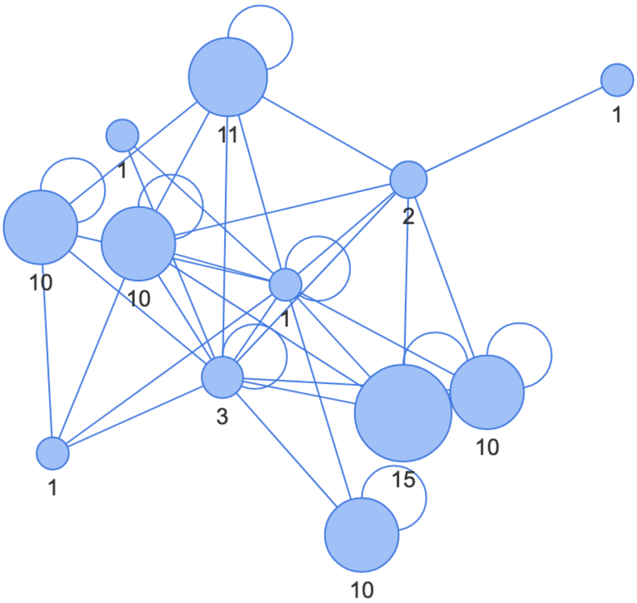


Figure 17. Graph of the register group for the clock-gated design

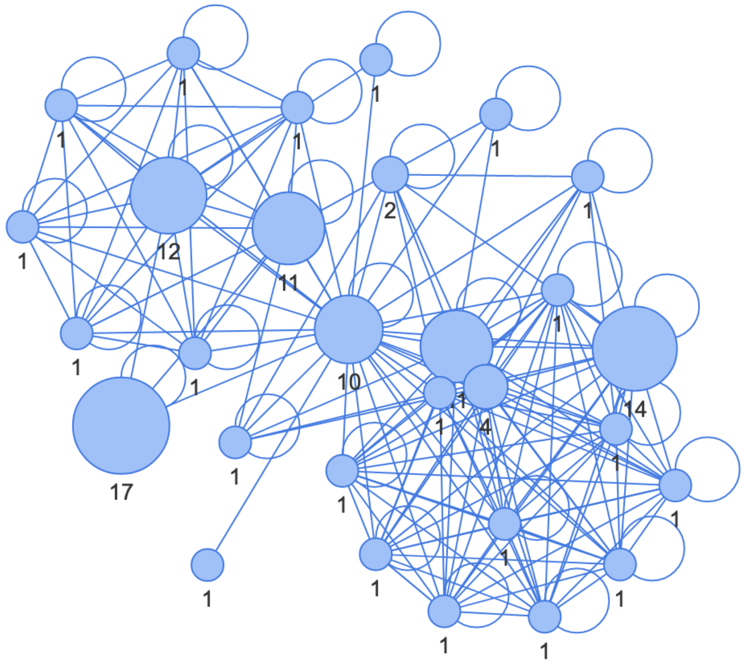


Figure 18. Graph of the register group for the retiming (delay)

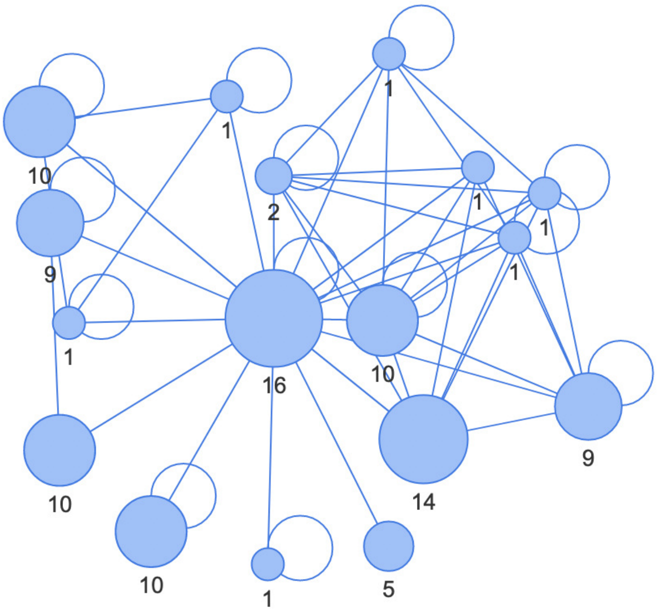


Figure 19. Graph of the register group for the clock-gating and retiming (delay)

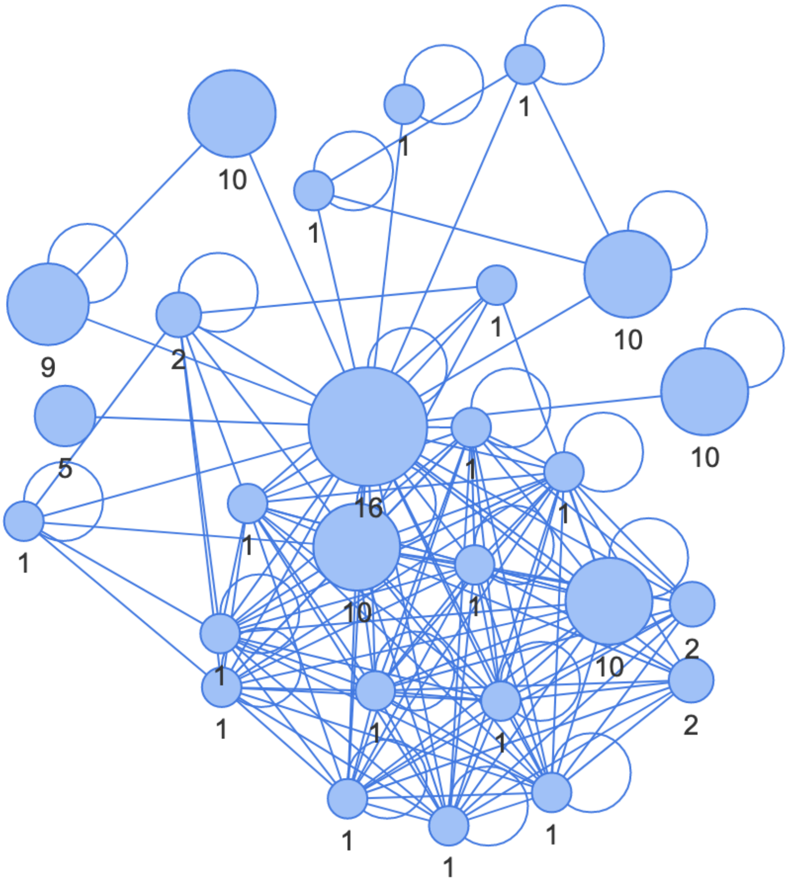


Figure 20. Graph of the register group for the clock-gating and retiming (delay) with steered mode (Register size 10).

# 5. Conclusions

The advancements in IC production and the increasing availability of reverse engineering tools highlighted the need for hardware security. This is why in this thesis we presented a novel approach to generating hundreds of unique designs from a single one, in an attempt to confuse the adversary and delay the reverse engineering of the design. We demonstrated in the case study of a GPS correlator the effectiveness of the approach and analysed the overhead in terms of area and power. We saw that generated designs were clustered in multiple groups by the state of the art reverse engineering tool DANA, which means that the adversary would have to put an extra effort to analyse them separately. Even though in our case study the average dynamic and leakage powers increased significantly, the results will vary from design to design and the synthesis tools utilised. Nevertheless, the approach showed promising results, meaning that it could help protect circuits that instantiate the same module multiple times. Despite promising results, due to the high range of adversary's skills and equipment, it is impossible to quantitatively measure how much harder it is to reverse engineer and the amount of additional effort needed to learn the complete functionality of the circuit.

In the future, another synthesis tool can be used with both academic and commercial cell libraries. Moreover, other reverse engineering tools such as RELIC can be used to analyse the designs. Additionally, the ICs can be manufactured and analysed to examine whether design diversity holds after physical implementation. Moreover, since the synthesis of larger circuits is time consuming, synthesising hundreds of designs when only several of them are needed is inefficient thus a methodology can be developed which can suggest which optimisation strategy to utilise and what would be the approximate divergence from the baseline version.

# Bibliography

[1] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. “A Primer on Hardware Security: Models, Methods, and Metrics”. In: *Proceedings of the IEEE* 102.8 (2014), pp. 1283–1295. DOI: `10.1109/jproc.2014.2335155`.

[2] Ed Sperling. *Fundamental changes in economics of Chip Security*. 2020. URL: `https://semiengineering.com/fundamental-changes-in-economics-of-security/`.

[3] Yiran Chen et al. “A Survey of Accelerator Architectures for Deep Neural Networks”. In: *Engineering* 6.3 (2020), pp. 264–274. ISSN: 2095-8099. DOI: `https://doi.org/10.1016/j.eng.2020.01.007`. URL: `https://www.sciencedirect.com/science/article/pii/S2095809919306356`.

[4] Hadi Esmaeilzadeh et al. “Neural Acceleration for General-Purpose Approximate Programs”. In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 2012, pp. 449–460. DOI: `10.1109/MICRO.2012.48`.

[5] Mark C. Hansen, Hakan Yalcin, and John P. Hayes. “Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering”. In: *IEEE Design & Test of Computers* 16.3 (1999), pp. 72–80. DOI: `10.1109/54.785838`.

[6] Pramod Subramanyan et al. “Reverse Engineering Digital Circuits Using Functional Analysis”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE Conference Publications, 2013. DOI: `10.7873/date.2013.264`.

[7] Wenchao Li, Zach Wasson, and Sanjit A. Seshia. “Reverse engineering circuits using behavioral pattern mining”. In: *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2012. DOI: `10.1109/hst.2012.6224325`.

[8] Nils Albartus et al. “DANA Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 309–336. DOI: `10.46586/tches.v2020.i4.309-336`.

[9] Stjepan Picek et al. “Side-channel analysis and machine learning: A practical perspective”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 4095–4102. DOI: `10.1109/IJCNN.2017.7966373`.

[10] Mark Randolph and William Diehl. “Power side-channel attack analysis: A review of 20 years of study for the layman”. In: *Cryptography* 4.2 (2020), p. 15.

[11] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. “Template attacks”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2002, pp. 13–28.

[12] John Demme et al. “Side-channel vulnerability factor: A metric for measuring information leakage”. In: *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2012, pp. 106–117.

[13] Mohammad Tehranipoor and Farinaz Koushanfar. “A Survey of Hardware Trojan Taxonomy and Detection”. In: *IEEE Design Test of Computers* 27.1 (2010), pp. 10–25. DOI: 10.1109/MDT.2010.7.

[14] Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. “Detecting malicious inclusions in secure hardware: Challenges and solutions”. In: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE. 2008, pp. 15–19.

[15] Andrew B. Kahng et al. “Watermarking techniques for intellectual property protection”. In: *Proceedings of the 35th annual Design Automation Conference*. 1998, pp. 776–781.

[16] Tiago D. Perez and Samuel Pagliarini. “A Survey on Split Manufacturing: Attacks, Defenses, and Challenges”. In: *IEEE Access* 8 (2020), pp. 184013–184035. DOI: 10.1109/ACCESS.2020.3029339.

[17] Takeshi Sugawara et al. “Reversing Stealthy Dopant-Level Circuits”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 112–126. DOI: 10.1007/978-3-662-44709-3_7.

[18] Arunkumar Vijayakumar et al. “Physical Design Obfuscation of Hardware: A Comprehensive Investigation of Device and Logic-Level Techniques”. In: *IEEE Transactions on Information Forensics and Security* 12.1 (2017), pp. 64–77. DOI: 10.1109/tifs.2016.2601067.

[19] James C. Tsang, Jeffrey A. Kash, and David P. Vallett. “Picosecond imaging circuit analysis”. In: *IBM Journal of Research and Development* 44.4 (2000), pp. 583–603. DOI: 10.1147/rd.444.0583.

[20] Jeyavijayan Rajendran et al. “Security analysis of integrated circuit camouflaging”. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 709–720.

[21] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. “EPIC: Ending Piracy of Integrated Circuits”. In: *2008 Design, Automation and Test in Europe*. IEEE, 2008. DOI: 10.1109/date.2008.4484823.

[22] Jeyavijayan Rajendran et al. “Security analysis of logic obfuscation”. In: *Proceedings of the 49th Annual Design Automation Conference on - DAC ’12*. ACM Press, 2012. DOI: 10.1145/2228360.2228377.

[23] Sarah Amir et al. “Development and Evaluation of Hardware Obfuscation Benchmarks”. In: *Journal of Hardware and Systems Security* 2.2 (2018), pp. 142–161. DOI: 10.1007/s41635-018-0036-3.

[24] Pramod Subramanyan, Sayak Ray, and Sharad Malik. “Evaluating the security of logic encryption algorithms”. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2015, pp. 137–143.

[25] Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. “SAIL: Machine learning guided structural analysis attack on hardware obfuscation”. In: *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE. 2018, pp. 56–61.

[26] Yang Xie and Ankur Srivastava. “Anti-SAT: Mitigating SAT attack on logic locking”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.2 (2018), pp. 199–207.

[27] Javier M. Duarte et al. “Fast inference of deep neural networks in FPGAs for particle physics”. In: *Journal of Instrumentation* 13.07 (2018).

[28] Homer Hsing. *AES-128*. 2013. URL: https://opencores.org/projects/tiny%5C_aes.

[29] Zhaokun Han, Muhammad Yasin, and Jeyavijayan (JV) Rajendran. “Does logic locking work with EDA tools?” In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 1055–1072. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/han-zhaokun.

[30] Sophie Dupuis and Marie-Lise Flottes. “Logic Locking: A Survey of Proposed Methods and Evaluation Metrics”. In: *Journal of Electronic Testing* 35.3 (2019), pp. 273–291. DOI: 10.1007/s10836-019-05800-4. URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-02128826.

[31] Jitesh Shinde and Suresh S. Salankar. “Clock gating — A power optimizing technique for VLSI circuits”. In: *2011 Annual IEEE India Conference*. 2011, pp. 1–4. DOI: 10.1109/INDCON.2011.6139440.

[32] Prasanth Chatarasi et al. "Vyasa: A High-Performance Vectorizing Compiler for Tensor Convolutions on the Xilinx AI Engine". In: *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020. DOI: `10.1109/hpec43674.2020.9286183`.

[33] Nandita Srinivasan et al. "Power Reduction by Clock Gating Technique". In: *Procedia Technology* 21 (2015), pp. 631–635. DOI: `10.1016/j.protcy.2015.10.075`.

[34] Smitha Iyengar and Lakshmi Shrinivasan. "Power, Performance and Area Optimization of I/O Design". In: *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, 2018. DOI: `10.1109/icirca.2018.8597347`.

[35] Naushad Manzoor Laskar et al. "A survey on VLSI Floorplanning: Its representation and modern approaches of optimization". In: *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. IEEE, 2015. DOI: `10.1109/iciiecs.2015.7192989`.

[36] Hsun-Cheng Lee et al. "Multilevel floorplanning/placement for large-scale modules using $B* - trees$". In: *Proceedings of the 40th conference on Design automation - DAC '03*. ACM Press, 2003. DOI: `10.1145/775832.776037`.

[37] Igor L. Markov, Jin Hu, and Myung-Chul Kim. "Progress and Challenges in VLSI Placement Research". In: *Proceedings of the IEEE* 103.11 (2015), pp. 1985–2003. DOI: `10.1109/jproc.2015.2478963`.

[38] Sheng Chou, Meng-Kai Hsu, and Yao-Wen Chang. "Structure-aware placement for datapath-intensive circuit designs". In: *Proceedings of the 49th Annual Design Automation Conference on - DAC '12*. ACM Press, 2012. DOI: `10.1145/2228360.2228498`.

[39] Minsik Cho et al. "LatchPlanner: Latch placement algorithm for datapath-oriented high-performance VLSI designs". In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013. DOI: `10.1109/iccad.2013.6691141`.

[40] Thomas Kutzschebauch and Leon Stok. "Regularity driven logic synthesis". In: *IEEE/ACM International Conference on Computer Aided Design. ICCAD - 2000. IEEE/ACM Digest of Technical Papers (Cat. No.00CH37140)*. IEEE. DOI: `10.1109/iccad.2000.896511`.

[41] Samuel Pagliarini et al. "Evaluating Architectural, Redundancy, and Implementation Strategies for Radiation Hardening of FinFET Integrated Circuits". In: *IEEE Transactions on Nuclear Science* 68.5 (2021), pp. 1045–1053. DOI: `10.1109/tns.2021.3070643`.

[42] Tyler J. Thorp, Gin S. Yee, and Carl M. Sechen. “Design and synthesis of dynamic circuits”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11.1 (2003), pp. 141–149. DOI: 10.1109/tvlsi.2002.800518.

[43] Chin-Chi Teng, Anthony M. Hill, and Sung-Mo Kang. “Estimation of maximum transition counts at internal nodes in CMOS VLSI circuits”. In: *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*. IEEE Comput. Soc. Press. DOI: 10.1109/iccad.1995.480142.

[44] Yajun Ran et al. “Eliminating false positives in crosstalk noise analysis”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.9 (2005), pp. 1406–1419. DOI: 10.1109/tcad.2005.850829.

[45] Mohamed Chentouf and Alaoui Zine El Abidine. “Evaluating the Impact of Max Transition Constraint Variations on Power Reduction Capabilities in Cell-Based Designs”. In: *Journal of Low Power Electronics and Applications* 7.4 (2017), p. 25. DOI: 10.3390/jlpea7040025.

[46] Jason Cong et al. “Performance optimization of VLSI interconnect layout”. In: *Integration* 21.1-2 (1996), pp. 1–94. DOI: 10.1016/s0167-9260(96)00008-9.

[47] Naresh Maheshwari and Sachin Sapatnekar. “Efficient retiming of large circuits”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 6.1 (1998), pp. 74–83. DOI: 10.1109/92.661250.

[48] Nagesh Shenoy and Richard Rudell. “Efficient Implementation Of Retiming”. In: *IEEE/ACM International Conference on Computer-Aided Design*. IEEE. DOI: 10.1109/iccad.1994.629770.

[49] Charles E. Leiserson, Flavio M. Rose, and James B. Saxe. “Optimizing Synchronous Circuitry by Retiming (Preliminary Version)”. In: *Third Caltech Conference on Very Large Scale Integration*. Springer Berlin Heidelberg, 1983, pp. 87–116. DOI: 10.1007/978-3-642-95432-0_7.

[50] Vinita Pandey, Subhash C. Yadav, and Priya Arora. “Retiming technique for clock period minimization using shortest path algorithm”. In: *2016 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 2016. DOI: 10.1109/ccaa.2016.7813942.

[51] Embedded Security Group. *HAL - The Hardware Analyzer*. https://github.com/emsec/hal. 2019.

[52] Marc Fyrbiak et al. “HAL- The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion”. In: *IEEE Transactions on Dependable and Secure Computing* (2018).

[53] West Health Institute. *Pyvis*. https://github.com/WestHealth/pyvis. 2018.

# Appendices

# Appendix 1 - GPS Correlator Code

```
module cor_flex (clk, rst_n, enable, data_from_sensor, satellite, offset, key_bits, key_sel, locked, ranking);
input clk;
input rst_n;
input enable;
input data_from_sensor;
input [4:0] satellite; // range is [0..31], so 5 bits are needed
input [9:0] offset; // range is [0..1022], so 10 bits are neede
input [3:0] key_bits;
input [1:0] key_sel;
output reg locked;
output reg signed [10:0] ranking; // best case scenario will have a match on all 1023 bits. I need 10 bits to store
    that. plus one for signal

localparam STATE_SIZE = 4;
localparam WAITING = 4'd0;
localparam DECIDING = 4'd1;
localparam OFFSETTING = 4'd2;
localparam RUNNING = 4'd3;
localparam IDLE = 4'd4;
localparam LOCKED = 4'd5;
localparam READ_RANK = 4'd6;
localparam SETTING  = 4'd7;
localparam DIVIDING = 4'd8;
localparam CRCING = 4'd9;
localparam LOADING= 4'd10;

reg [STATE_SIZE-1:0] state, next_state;

reg [4:0] local_satellite, next_local_satellite;
reg [9:0] local_offset, next_local_offset;
reg [9:0] local_g1_setting, next_g1_setting;
reg [9:0] local_runtime, next_runtime;
reg div_sel, next_div_sel;
reg next_locked;
reg [10:0] next_ranking;

localparam SEQ_SIZE = 1023;
localparam SEQ_SIZE_LIMIT = 10'd1022;
localparam SEED = 10'b1111111111;


reg [9:0] g1, next_g1; // G1 has feedback from position 3 and 10, and it is constant for all satellites
wire tap_for_g1_g6;
wire and_out_g6;
and(and_out_g6, g1[9], local_g1_setting[6]);
xor(tap_for_g1_g6, g1[5], and_out_g6);

wire tap_for_g1_g2;
wire and_out_g2;
and(and_out_g2, g1[9], local_g1_setting[2]);
xor(tap_for_g1_g2, g1[1], and_out_g2);

wire tap_for_g1_g1;
wire and_out_g1;
and(and_out_g1, g1[9], local_g1_setting[1]);
xor(tap_for_g1_g1, g1[0], and_out_g1);

wire tap_for_g1_g3;
wire and_out_g3;
```

```
and(and_out_g3, g1[9], local_g1_setting[3]);
xor(tap_for_g1_g3, g1[2], and_out_g3);

wire tap_for_g1_g4;
wire and_out_g4;
and(and_out_g4, g1[9], local_g1_setting[4]);
xor(tap_for_g1_g4, g1[3], and_out_g4);

wire tap_for_g1_g5;
wire and_out_g5;
and(and_out_g5, g1[9], local_g1_setting[5]);
xor(tap_for_g1_g5, g1[4], and_out_g5);

wire tap_for_g1_g7;
wire and_out_g7;
and(and_out_g7, g1[9], local_g1_setting[7]);
xor(tap_for_g1_g7, g1[6], and_out_g7);

wire tap_for_g1_g8;
wire and_out_g8;
and(and_out_g8, g1[9], local_g1_setting[8]);
xor(tap_for_g1_g8, g1[7], and_out_g8);

wire tap_for_g1_g9;
wire and_out_g9;
and(and_out_g9, g1[9], local_g1_setting[9]);
xor(tap_for_g1_g9, g1[8], and_out_g9);


wire divid_in;
xor (divid_in, g1[9], data_from_sensor);

wire tap_for_g1_g0;
assign tap_for_g1_g0 = div_sel? divid_in : g1[9];



reg [9:0] g2, next_g2;
wire tap_for_g2;
xor(tap_for_g2, g2[1], g2[2], g2[5], g2[7], g2[8], g2[9]); // G2 has feedback from 2,3,6,8,9, and 10. constant for all
        satellites


wire final_sum1;
xor (final_sum1, g1[6], g2[1], g2[5]);

wire final_sum2;
xor (final_sum2, g1[6], g2[2], g2[6]);

wire final_sum3;
xor (final_sum3, g1[6], g2[3], g2[7]);

wire final_sum4;
xor (final_sum4, g1[6], g2[4], g2[8]);

wire final_sum5;
xor (final_sum5, g1[6], g2[0], g2[8]);

wire final_sum6;
xor (final_sum6, g1[6], g2[1], g2[9]);

wire final_sum7;
xor (final_sum7, g1[6], g2[0], g2[7]);

wire final_sum8;
xor (final_sum8, g1[6], g2[1], g2[8]);

wire final_sum9;
xor (final_sum9, g1[6], g2[2], g2[9]);

wire final_sum10;
xor (final_sum10, g1[6], g2[1], g2[2]);

wire final_sum11;
xor (final_sum11, g1[6], g2[2], g2[3]);

wire final_sum12;
xor (final_sum12, g1[6], g2[4], g2[5]);

wire final_sum13;
```

```
xor (final_sum13, g1[6], g2[5], g2[6]);

wire final_sum14;
xor (final_sum14, g1[6], g2[6], g2[7]);

wire final_sum15;
xor (final_sum15, g1[6], g2[7], g2[8]);

wire final_sum16;
xor (final_sum16, g1[6], g2[8], g2[9]);

wire final_sum17;
xor (final_sum17, g1[6], g2[0], g2[3]);

wire final_sum18;
xor (final_sum18, g1[6], g2[1], g2[4]);

wire final_sum19;
xor (final_sum19, g1[6], g2[2], g2[5]);

wire final_sum20;
xor (final_sum20, g1[6], g2[3], g2[6]);

wire final_sum21;
xor (final_sum21, g1[6], g2[4], g2[7]);

wire final_sum22;
xor (final_sum22, g1[6], g2[5], g2[8]);

wire final_sum23;
xor (final_sum23, g1[6], g2[0], g2[2]);

wire final_sum24;
xor (final_sum24, g1[6], g2[3], g2[5]);

wire final_sum25;
xor (final_sum25, g1[6], g2[4], g2[6]);

wire final_sum26;
xor (final_sum26, g1[6], g2[5], g2[7]);

wire final_sum27;
xor (final_sum27, g1[6], g2[6], g2[8]);

wire final_sum28;
xor (final_sum28, g1[6], g2[7], g2[9]);

wire final_sum29;
xor (final_sum29, g1[6], g2[0], g2[5]);

wire final_sum30;
xor (final_sum30, g1[6], g2[1], g2[6]);

wire final_sum31;
xor (final_sum31, g1[6], g2[2], g2[7]);

wire final_sum32;
xor (final_sum32, g1[6], g2[3], g2[8]);

reg use_this_sum;
always @(*) begin
 use_this_sum = 0;

 case (local_satellite)
  5'd0: use_this_sum = final_sum1;
  5'd1: use_this_sum = final_sum2;
  5'd2: use_this_sum = final_sum3;
  5'd3: use_this_sum = final_sum4;
  5'd4: use_this_sum = final_sum5;
  5'd5: use_this_sum = final_sum6;
  5'd6: use_this_sum = final_sum7;
  5'd7: use_this_sum = final_sum8;
  5'd8: use_this_sum = final_sum9;
  5'd9: use_this_sum = final_sum10;
  5'd10: use_this_sum = final_sum11;
  5'd11: use_this_sum = final_sum12;
  5'd12: use_this_sum = final_sum13;
  5'd13: use_this_sum = final_sum14;
  5'd14: use_this_sum = final_sum15;
  5'd15: use_this_sum = final_sum16;
  5'd16: use_this_sum = final_sum17;
  5'd17: use_this_sum = final_sum18;
  5'd18: use_this_sum = final_sum19;
  5'd19: use_this_sum = final_sum20;
  5'd20: use_this_sum = final_sum21;
  5'd21: use_this_sum = final_sum22;
  5'd22: use_this_sum = final_sum23;
  5'd23: use_this_sum = final_sum24;
  5'd24: use_this_sum = final_sum25;
  5'd25: use_this_sum = final_sum26;
  5'd26: use_this_sum = final_sum27;
  5'd27: use_this_sum = final_sum28;
  5'd28: use_this_sum = final_sum29;
  5'd29: use_this_sum = final_sum30;
  5'd30: use_this_sum = final_sum31;
  5'd31: use_this_sum = final_sum32;
 endcase
end

wire partial_score;
reg data_from_sensor_copy;
xor (partial_score, use_this_sum, data_from_sensor_copy);



reg sum1, sub1;
reg next_sum1, next_sub1;

reg temp, next_temp;

always @(posedge clk) begin
 if (rst_n == 1'b0) begin
  state <= WAITING;
  g1 <= SEED;
  g2 <= SEED;
  local_satellite <= 5'd0;
  local_offset <= 10'd0;
        local_g1_setting <= 10'd0;
  locked <= 1'b0;
        local_runtime<= 10'd0;
  ranking <= 10'd0;
  sum1 <= 1'b0;
  sub1 <= 1'b0;
        div_sel <=1'b0;
  temp <= 1'b0;
  data_from_sensor_copy = 0;
 end
 else begin
  state <= next_state;
  g1 <= next_g1;
  g2 <= next_g2;
        div_sel <= next_div_sel;
  local_satellite <= next_local_satellite;
        local_g1_setting <= next_g1_setting;
  local_offset <= next_local_offset;
  locked <= next_locked;
  ranking <= next_ranking;
        local_runtime <= next_runtime;
  sum1 <= next_sum1;
  sub1 <= next_sub1;
  temp <= next_temp;
  data_from_sensor_copy = data_from_sensor;
 end
end


always @(*) begin
 next_state = WAITING;
 next_g1 = g1;
 next_g2 = g2;
    next_div_sel = 1'b0;
    next_runtime = local_runtime;
 next_local_satellite = local_satellite;
 next_local_offset = local_offset;
    next_g1_setting = local_g1_setting;
 next_locked = 1'b0;
 next_sum1 = 1'b0;
 next_sub1 = 1'b0;
 next_ranking = ranking;
 next_temp = temp;
```

```
if (sum1) begin
 next_ranking = ranking + offset[9:5]*offset[4:0];
end
else if (sub1) begin
 next_ranking = ranking - offset[9:5]*offset[4:0];
end


case (state)
 WAITING: begin
  if (enable) begin
   next_local_satellite = satellite;
   next_local_offset = offset; // copies from the input
   next_state = SETTING;
  end
  next_temp = key_bits[key_sel];
 end
       SETTING:  begin
           next_g1_setting = offset;
  next_state = LOADING;
 end
       LOADING:  begin
           next_runtime = offset;
  next_state = DECIDING;
       end
 DECIDING: begin
  if (satellite == 5'b00101) begin
               next_state = SETTING;
  end
           else if (satellite== 5'b01001) begin
               next_state = DIVIDING ;
               next_div_sel  = 1'b1;
           end
  else if (local_offset != 0) begin
   next_state = OFFSETTING;
   next_g1 = {tap_for_g1_g9,tap_for_g1_g8,tap_for_g1_g7,tap_for_g1_g6,tap_for_g1_g5,tap_for_g1_g4,tap_for_g1_g3,
        tap_for_g1_g2,tap_for_g1_g1,tap_for_g1_g0};
   next_g2 = {g2[8:0], tap_for_g2}; // but this doesn't count as offset.
  end
  else begin
   next_state = RUNNING;
   next_g1 = {tap_for_g1_g9,tap_for_g1_g8,tap_for_g1_g7,tap_for_g1_g6,tap_for_g1_g5,tap_for_g1_g4,tap_for_g1_g3,
        tap_for_g1_g2,tap_for_g1_g1,tap_for_g1_g0};
   next_g2 = {g2[8:0], tap_for_g2}; // but this doesn't count as offset.
  end
 end
       DIVIDING:  begin
     next_runtime = local_runtime - 10'd1;
  next_g1 = {tap_for_g1_g9,tap_for_g1_g8,tap_for_g1_g7,tap_for_g1_g6,tap_for_g1_g5,tap_for_g1_g4,tap_for_g1_g3,
       tap_for_g1_g2,tap_for_g1_g1,tap_for_g1_g0};
  if (local_runtime == 10'd0) begin
               next_div_sel  = 1'b0;

               if (satellite == 5'b01101) begin
                next_state = IDLE;
               end
               else if (satellite == 5'b01100) begin
                next_state = SETTING;
               end
               else if (satellite == 5'b01000) begin
                next_state = LOADING;
               end
               else if (satellite == 5'b00001) begin
                next_state = DECIDING;
               end
               next_runtime = offset;
   next_state = RUNNING;
  end
  else begin
   next_state = DIVIDING;
               next_div_sel  = 1'b1;
               next_ranking = {ranking[9:0],g1[9]};
  end
       end
 OFFSETTING: begin
  next_local_offset = local_offset - 10'd1;
  next_g1 = {tap_for_g1_g9,tap_for_g1_g8,tap_for_g1_g7,tap_for_g1_g6,tap_for_g1_g5,tap_for_g1_g4,tap_for_g1_g3,
       tap_for_g1_g2,tap_for_g1_g1,tap_for_g1_g0};
  next_g2 = {g2[8:0], tap_for_g2};
```

```
   if (local_offset != 1) begin
    next_state = OFFSETTING;
   end
   else begin
    next_state = RUNNING;
   end
  end
  RUNNING: begin
   next_runtime = local_runtime - 10'd1;

   if (satellite== 5'b0000) begin
    next_g1 = {tap_for_g1_g9,tap_for_g1_g8,tap_for_g1_g7,tap_for_g1_g6,tap_for_g1_g5,tap_for_g1_g4,tap_for_g1_g3,
         tap_for_g1_g2,tap_for_g1_g1,tap_for_g1_g0};
    next_g2 = {g2[8:0], tap_for_g2};
                if (partial_score == 1'b0) begin // meaning they are the same (remember it is XORed)
     next_sum1 = 1'b1;
    end
    else begin
     next_sub1 = 1'b1;
    end
            end
            else if (satellite == 5'b00101) begin // meaning they are the same (remember it is XORed)
    next_sub1 =1'b1;
   end
   else  begin
    next_sum1 = 1'b1;
   end

          if ( local_runtime ==10'd0 ) begin
    if (satellite == 5'b00010) begin
                    next_state = DIVIDING;
                end
                else if (satellite == 5'b00011) begin
                    next_state = SETTING;
                end
                else if (satellite == 5'b00111) begin
                    next_state = LOADING;
                end
                else if (satellite == 5'b01110) begin
                    next_state = DECIDING;
                    next_runtime = offset;
                end
                else begin
     next_state = IDLE;
    end
            end
   else begin
    next_state = RUNNING;
   end
  end
  IDLE: begin
   // I need this idle state to allow for the ranking to
   // update, it is pipelined
   next_locked = 1'b1;
   next_state = LOCKED;
  end
  LOCKED: begin
   next_state = READ_RANK & {temp, temp, temp};
  end
  READ_RANK: begin
   // I need this state to allow the rankings to be
   // propagated outside
   next_state = WAITING;
   next_ranking = 10'd0;
  end
 endcase
end
endmodule
```