INTEGRATION OF A MAZE SHIELD INTO DIGITAL ASIC DESIGN FLOW FOR HARDWARE ATTACK RESISTANT ICs

by

Can Kurt

B.S., Electronics and Communication Engineering, Istanbul Technical University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2023

ACKNOWLEDGEMENTS

First and foremost, I express my gratitude to Assist Prof. Dr. İsmail Faik Başkaya, Raşit Rıdvan Turgut, and Prof. Dr. Müştak Erhan Yalçın for their invaluable support and guidance throughout my M.Sc. thesis journey.

Secondly, I want to thank my colleagues from TÜBİTAK and Analog Devices for their technical and mental support, which enhanced this work.

Finally, I am grateful to my family and my girlfriend for their continuous love and support.

A part of this work has been realized within the TÜBİTAK research project 5210073 framework.

ABSTRACT

INTEGRATION OF A MAZE SHIELD INTO DIGITAL ASIC DESIGN FLOW FOR HARDWARE ATTACK RESISTANT ICs

With the increasing use of Integrated Circuits (ICs) in various sectors such as automotive and banking, it has become inevitable to use them for storing critical data. As software security has strengthened, attackers have shifted to direct physical attacks on chips. One such attack involves using Microprobes and Focused Ion Beam (FIB) for structured intrusion. In this type of attack, the goal is to mill the outer layers of the chip with an ion beam and touch the internal data with a microprobe, gaining access to the signals inside the chip without the need for any software-based system. The objective of this thesis is to develop a flow that implements all the stages required to produce a chip resilient to attacks using Microprobes and Focused Ion Beams. The chip to be produced, containing critical information, will have a shield structure above the location of critical data and a control circuit below to detect any external attacks on the shield. The first step involves creating the shield to be placed on the chip. To prevent easy tracking of the generated shield, a complex maze algorithm called AFSA is employed. The generated maze is traversed by a function that converts the maze into a Hamiltonian cycle, transforming it into a shield structure. The produced shield is made examinable through a 3D visualization function. This function is used to check the complexity and accuracy of the generated maze. The generated shield is then transformed into the Library Exchange Format (LEF) file format, a language understood by digital design tools. The converted shield, along with the control circuit and the circuit to be protected, undergoes the digital chip design flow. At the end of the flow, a production-ready secure chip is created and sent for tape-out.

ÖZET

LABİRENT TABANLI KALKAN TASARIMLARININ DONANIMSAL SALDIRILARA DAYANIKLI ENTEGRE DEVRELER İÇİN OLAN ÇİP AKIŞINA EKLENMESİ

Günümüzde otomotivden bankacılığa kadar pek çok sektörde entegre devrelerin kritik verileri saklaması ve işlemesi kaçınılmaz hale gelmiştir. Yazılım güvenliği güç kazandıkça, saldırganlar direkt olarak bu entegre devreleri içeren çiplere fiziksel yöntemler ile saldırı yapmaya başlamışlardır. Bunlardan şu an en bilinenleri Mikroprobe ve Odaklanmış İyon Işını kullanılarak yapılan saldırılardır. Bu saldırıda amaç, çipin dış katmanlarını iyon ışını ile oyup içerideki verilere mikroprobe ile dokunarak, herhangi yazılımsal bir sisteme ihtiyaç duymadan, çipin içinde akan sinyallere erişmektir. Bu tezin amacı Mikroprobe ve Odaklanmış Iyon Işını ile yapılacak olan saldırılara dayanıklı bir çip üretebilmek için gerekli olan tüm aşamaları içeren bir akışı geliştirmek ve 65nm TSMC LP teknolojisi üzerinde uygulamaktır. Çalışma ilk olarak kritik verilerin çip üzerinde bulunduğu bölgeyi koruyabilmek için otomatik kalkan üretmeyi ve üretilen kalkanın zaman içinde saldırıya uğrayıp uğramadığını tespit edecek kontrol devresini ve bu devrenin kalkan ile bağlantısını içerir. Üretilen kalkanın kolay bir şekilde takip edilmesini engellemek için karmaşık bir labirent algoritması olan AFSA baz alınmıstır. Üretilen labirent, labirenti kalkan yapısına dönüştüren bir fonksiyon tarafından etrafı taranarak Hamiltonian döngüye dönüştürülür. Üretilen kalkan, 3 boyutlu görsel fonksiyonu sayesinde incelenebilir hale getirilir. Bu fonksiyon sayesinde üretilen labirentin karmaşıklığı ve doğruluğu kontrol edilir. Üretilen kalkan, sayısal tasarım araçlarının anlayabileceği bir dil olan LEF dosya formatına dönüştürülür. Dönüştürülen kalkan, kontrol devresi ve korunacak devre ile beraber sayısal çip tasarım akışından geçirilir. Akış sonunda üretime hazır güvenli çip oluşturulmuş olur. Aynı zamanda tasarlanmış olan çip başarılı bir şekilde üretim için gönderilmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xi
LIST OF ACRONYMS/ABBREVIATIONS . . . xii
1. INTRODUCTION . . . 1
1.1. Organization of Thesis . . . 2
2. LITERATURE REVIEW . . . 3
3. BACKGROUND . . . 5
3.1. Hardware Attacks . . . 5
3.1.1. Focused Ion Beam(FIB) Attacks . . . 6
3.2. Active Shields . . . 8
3.2.1. Active Shield Structure . . . 9
3.3. Artificial Fish Swarm Algorithm . . . 9
3.4. RTL to GDSII Flow . . . 10
3.5. LEF File Format . . . 14
3.6. Hamiltonian Cycle . . . 15
4. METHODOLOGY . . . 17
4.1. Using Maze as Random Shield . . . 17
4.1.1. Artificial Fish Swarm Algorithm for IC Design . . . 18
4.1.2. Converting Maze into Shield . . . 20
4.2. Verifying Shield with 3D Visualization . . . 22
4.3. Converting Shield into Layout . . . 24
4.4. Implementing Maze into an ASIC Design with Its Checker Circuitry . . . 31
5. CONCLUSION . . . 37
REFERENCES . . . 39

APPENDIX A: SIMPLIFIED LEF FILE EXAMPLE 41

LIST OF FIGURES

Figure 3.1. Hardware attack types. 5

Figure 3.2. FIB and probing attack. 7

Figure 3.3. RTL to GDSII flow. 13

Figure 3.4. Hamiltonian graph and its Hamiltonian cycle examples. 16

Figure 4.1. Artificial fish swarm algorithm flow diagram. 19

Figure 4.2. Shields of various sizes. 21

Figure 4.3. Shields of various sizes with blockage. 22

Figure 4.4. 19x19 two layer shield with different angles. 24

Figure 4.5. Design rules. 25

Figure 4.6. LEF header example. 27

Figure 4.7. Layout view of the generated 40x40 maze with 2 layers. 28

Figure 4.8. 3D Layout view of the generated 40x40 maze with 2 layers. 28

Figure 4.9. Layers of generated 40x40 shield. 29

Figure 4.10. Shield of 100 2 by 2 mazes. 29

Figure 4.11. Shield of 16 4 by 4 mazes. 30

Figure 4.12. Shield of 1 5 by 5 mazes. 30

Figure 4.13. Synthesized schematic of top module. 32

Figure 4.14. Shield of 100 5 by 5 mazes. 33

Figure 4.15. Place and Route layout for TSMC 65 LP Process. 34

Figure 4.16. Place and Route layout for TSMC 65 LP Process. 35

Figure 4.17. GDSII output of design for TSMC 65GP Process. 36

Figure 5.1. GDSII output of design for TSMC 65LP Process. 38

Figure A.1. LEF file example 41

LIST OF TABLES

Table 4.1. Place and Route tool durations. 22

LIST OF SYMBOLS

d_x	Size of maze in axis X
d_y	Size of maze in axis y
$d_spacing$	Spacing between two metal nets
d_width	Width of metal nets
IN_U	User input
R_{MW}	Minimum width rule
R_{MS}	Minimum spacing rule
R_{MVE}	Minimum via enclosure rule
R_{MVW}	Minimum via width rule

LIST OF ACRONYMS/ABBREVIATIONS

3D	Three Dimensional
AFSA	Artificial Fish Swarm Algorithm
ASIC	Application Specific Integrated Circuit
CTS	Clock Tree Synthesis
DRC	Design Rule Check
EDA	Electronic Design Automation
FIB	Focused Ion Beam
GDS	Graphic Data System
IC	Integrated Circuit
LEF	Library Exchange Format
LVS	Layout vs Schematic
PNR	Place and Route
RAM	Random Access Memory
RTL	Register Transfer Level
TSMC	Taiwan Semiconductor Manufacturing Company

1. INTRODUCTION

In the 21st century, cybersecurity stands as a cornerstone of our safety and stability. As our lives become increasingly digitized, from personal interactions to critical infrastructure, the stakes of cybersecurity have never been higher. Cyberattacks threaten not only financial assets but also privacy, democracy, and national security. With emerging technologies expanding the attack surface, investing in robust cybersecurity measures is imperative.

With more complex software security algorithms, breaking into ICs became harder. This improvement led attackers to find new ways to penetrate into chips. One of their solution, instead of using software to hack the system, they use hardware vulnerabilities to access critical information. Hardware attack vulnerabilities present significant risks to the security and integrity of systems, necessitating a comprehensive understanding of these threats and the development of effective countermeasures. Various types of attacks exploit vulnerabilities in hardware components, leading to potential leakage of sensitive information and compromise of system integrity [1].

Various methods have been studied to exploit physical weaknesses of the ICs. One form of hardware attack on ICs involves manipulating the electrical properties of chips through techniques like voltage glitching or laser fault injection. These invasive methods disrupt normal operation or reveal sensitive data, allowing attackers to bypass software security measures and directly manipulate the chip's functioning. Another method, known as side-channel attacks, exploits unintended signals emitted by ICs during operation, such as variations in power consumption or electromagnetic radiation, to infer secret information like encryption keys. Lastly, attackers may resort to microprobing, a physical intrusion technique where they gain access to the circuitry of ICs to intercept or modify data flows. This invasive approach circumvents software security measures and grants attackers direct access to manipulate the chip's operations.

To prevent hardware probing, new way of defense mechanisms are introduced. Proposed system in this thesis includes a physical shield structure and control circuitry of that shield. With this method, probing attacks can be detected if there is any invasive action made and protected circuit can be alarmed. By leveraging internal shielding mechanisms, cryptographically secure shields, and other active defense measures, organizations can strengthen the security of their hardware systems and mitigate the risks associated with probing attacks.

1.1. Organization of Thesis

Throughout the entire thesis, a flow design has been discussed to create a circuit with a maze-based shield resistant to hardware probing attacks and a circuit controlling that shield.

In the first section, the concept is introduced, and the objectives have been outlined. The second section involves a literature review, discussing the innovations in the design. In the third section, applications and developed algorithms in the design process from start to finish is detailed. The fourth section concludes the work done.

2. LITERATURE REVIEW

In the contemporary landscape, the increasing significance of secure integrated circuits has prompted a surge in research endeavors focusing on the intricate aspects of this field. Using Focused Ion Beam (FIB) in nanofabrication technology introduces exploitable effects in an attack scenario, such as implantation, sputtering, surface smoothing, cleaning, and thin film formation [2]. Notably, various studies have delved into the challenges associated with producing circuits equipped with active shields to be protected from these kinds of attacks.

This necessitates shields with a heightened level of detail, thereby escalating both Random Access Memory (RAM) usage and production time. Xin et al. made a commendable effort to address this challenge. Their work proposed an innovative shield production algorithm utilizing a labyrinthine approach grounded in the "Artificial Fish Swarm Algorithm." This methodology reduces production time exponentially as the shield dimensions increase [3]. It forms the foundational framework for the modifications introduced in this thesis, tailored for an optimized digital design flow.

On a related note, Wang et al. offered an alternative algorithmic approach. Their proposal involved strategically arranging numerous simple shield cells in tandem, initially minimizing the time invested in shield production. The ensuing complexity is achieved by rotating the pathways created in the process [4]. However, despite its effectiveness in generating complex shields, this algorithm is deemed less suitable as a foundational framework due to its utilization of significant processor power for validating the completeness of the generated shield loop.

Meanwhile, Wang et al. took a distinctive path by focusing on the practicality of shield detection. This researcher aimed to simplify the process by employing a two-layer simple shield instead of a complex one. The goal was to enhance the detection of openings created by attacking FIB devices [5]. While their work is the sole instance

in the literature where the shield is seamlessly integrated with the control circuit, it comes with the caveat of deploying a relatively straightforward shield susceptible to observation, potentially leaving it vulnerable to targeted attacks.

Another research work evaluated the impact of a cryptographically secure shield on security IP protection, including its effectiveness against side-channel attacks [6]. By assessing the active shield's impact on side-channel attacks, this study contributes to understanding the efficacy of cryptographic shields in defending against probing attacks and other security threats.

Cioranesco et al. propose a protection method against attacks from the active layer by introducing a structure named Through Silicon Vias, which requires two dies bonded back to back [7]. However, this process is costly and time-consuming because it requires the consecutive bonding of a second die under the chip. Producing it is not straightforward with today's technology due to the necessity of aligning the two chips perfectly.

In essence, this thesis's primary objective remains centered around the comprehensive creation of a workflow. This encompasses the production of a complex active shield tailored for a digital design in need of protection, the integration of a circuit to control the active shield and the finalization of the entire system in a production-ready format. This holistic approach strives to furnish a robust and efficient solution for fortifying digital designs against potential threats, thereby contributing invaluable insights to the evolving domain of Secure Integrated Circuits.

3. BACKGROUND

3.1. Hardware Attacks

Hardware attacks are types of security threats that exploit vulnerabilities at the physical level of a computer system. They target hardware components such as memories, processors, and integrated circuits, as opposed to software attacks that target flaws in operating systems or programs. These types of attacks aim to undermine the confidentiality, integrity, and availability of the systems.

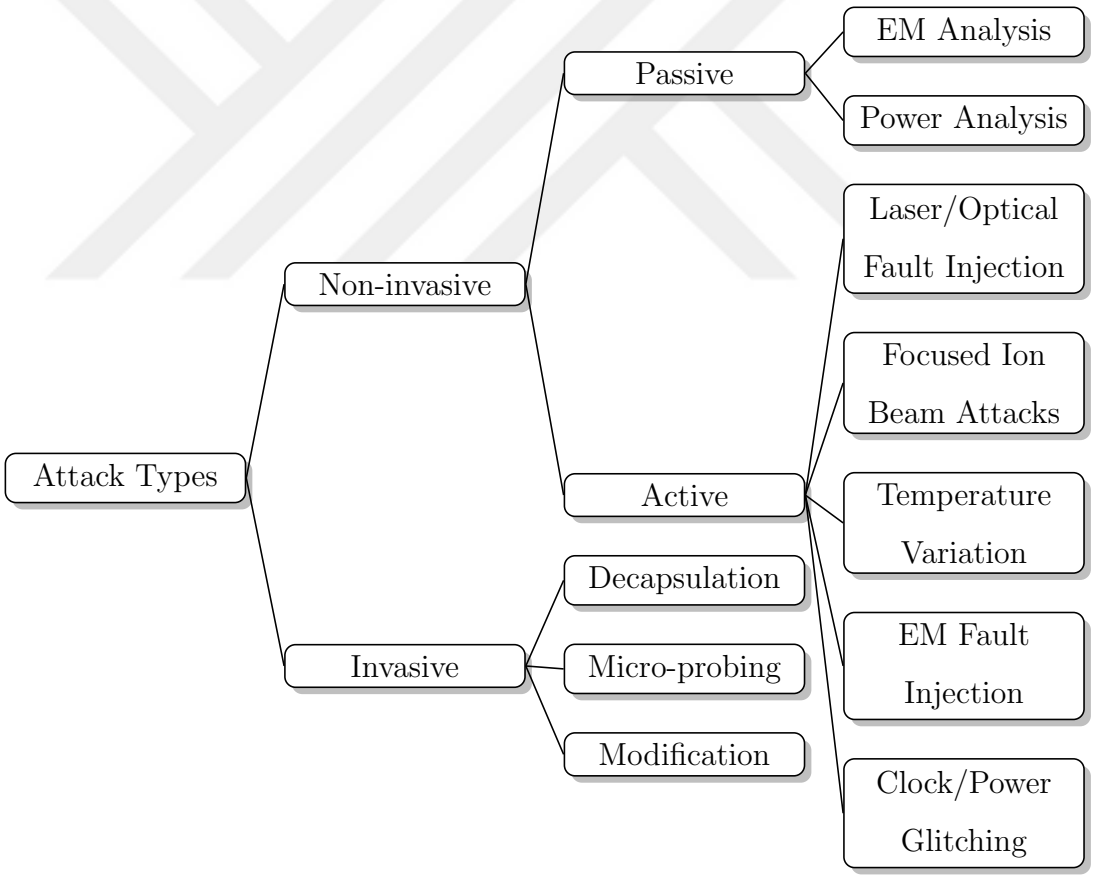


Figure 3.1. Hardware attack types.

There are various types of hardware attacks, each with specific objectives and methods. Probing attacks involve accessing internal components to extract sensitive information using tools such as lasers, probes, or needles. Side-channel attacks take

advantage of unintentional information leakage during device operation, such as analyzing power consumption, electromagnetic emissions, or timing patterns. Fault injection attacks manipulate system behaviors by intentionally inducing faults while tampering attacks modify hardware to weaken security features. Reverse engineering involves extracting hardware design or chip layout information by analyzing physical structures.

Examples of hardware attacks include Focused Ion Beam (FIB) attacks, Differential Power Analysis (DPA), voltage or clock glitching, tampering with components to create backdoors and reverse engineering using methods like scanning electron microscopy (SEM). Details of attack types can be seen in figure 3.1 [8].

Countermeasures against hardware attacks involve implementing robust cryptographic algorithms, designing secure hardware architectures, and incorporating physical security measures to protect against probing and tampering attempts. These measures enhance the overall security of computer systems by addressing vulnerabilities at the hardware level.

3.1.1. Focused Ion Beam(FIB) Attacks

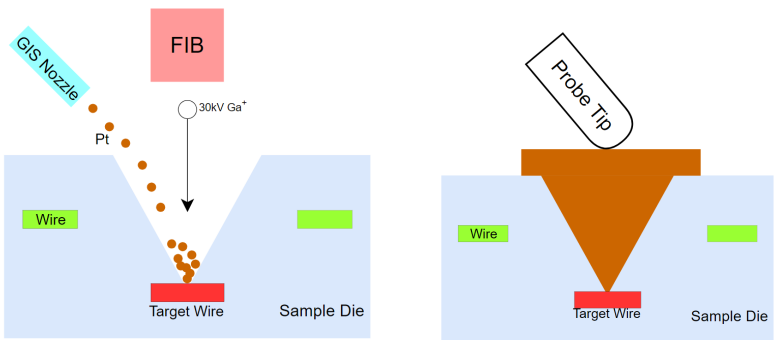
Focused Ion Beam (FIB) attacks represent a sophisticated form of physical security threat targeting integrated circuits (ICs) and semiconductor devices. These attacks leverage the precision of a focused beam of ions, typically gallium ions, to manipulate the surface and internal structure of the IC at the nanoscale level. FIB systems are equipped with a finely focused ion beam that can be precisely directed onto the surface of the semiconductor device as seen in figure 3.2(a) [5].

One of the primary applications of FIB attacks is reverse engineering, where attackers seek to understand the design and functionality of proprietary ICs by removing layers of material and probing internal circuitry. By selectively removing metal or polysilicon layers, attackers can gain access to sensitive components of the IC, such as memory cells or logic gates, allowing them to extract information or introduce malicious

modifications.

Moreover, FIB attacks can be used to alter the behavior of the IC by creating or modifying electrical pathways within the device. Attackers can use the ion beam to create conductive bridges between circuit elements or to disrupt existing connections, effectively bypassing security features such as encryption keys or memory access controls. This manipulation can enable attackers to gain unauthorized access to sensitive data or compromise the functionality of the device.

FIB attacks are often conducted in conjunction with other probing techniques (Figure 3.2(b)) [5], such as scanning electron microscopy (SEM) or focused ion beam microscopy (FIB-SEM), to precisely target specific areas of interest within the IC. These techniques allow attackers to navigate the device's complex internal structure and identify vulnerabilities that can be exploited.



(a) (b)

Figure 3.2. FIB and probing attack (a) Focused ion beam attack, (b) micro probing.

Countermeasures against FIB attacks encompass both physical security measures and design-level strategies. Physical security measures include tamper-evident packaging, shielding, and the use of specialized materials designed to resist ion beam penetration. Design-level countermeasures involve techniques such as layout obfuscation, redundancy, and randomization, which make it more difficult for attackers to

extract useful information or compromise the device through FIB attacks.

3.2. Active Shields

Hardware shields are a countermeasure designed to prevent or at least detect unwanted probings that aim to analyze internal mechanisms or information of integrated circuits. Probing attacks involve physical or electronic probing of the IC, seeking insights into its design, structure, or functionality, with the potential consequence of extracting sensitive information.

Two types of shields can serve as protection against hardware attacks: Passive and Active. A passive shield monitors the electrical characteristics of the shield, like capacitance, to identify any changes made to it from the outside, such as probing attempts. In contrast, an active shield typically introduces energy into the shield as a random signal, which travels through the shield wires and is then compared to an established pattern. If there's an effort to tamper with or breach the shield (like probing), the signal will change, triggering detection by the comparator [9].

Randomization techniques are frequently integrated into active shields to introduce variability into critical parameters like data locations or operation timings. This incorporation of randomness increases the system complexity for attackers, reducing their ability to predict the behavior of the IC accurately and thereby fortifying defenses against probing attempts.

Active shields also involve reactive countermeasures. When an abnormality is detected, these shields might adjust the setup of the ICs, activate extra security measures, or initiate a secure shutdown, thus reducing the impact of potential breaches. In severe situations, active shields might include self-destruct mechanisms that activate when unauthorized probing is detected. These mechanisms intentionally tamper with or harm the protected designs, making them unusable. These responsive measures enhance the flexible and adaptable characteristics of active shields.

Physical protection measures, including secure enclosures or packaging, are incorporated into the active shield system. Such measures heighten the difficulty for potential attackers to physically access and tamper with the IC, further fortifying defenses against probing attacks.

3.2.1. Active Shield Structure

Active Shields consist of two main parts: The shield structure that covers the protected area and the control logic for the shield cover.

Shield controllers can have various design approaches, but they should include two fundamental blocks. The first block is the one that drives the active shield with high or low logic levels in a desired frequency. This value change ensures that the attackers cannot drive the shield externally with a known value. To increase safety, this bit pattern generation can be made by a pseudo or real number generator. The second block required for the active shields is the comparator logic. This block reads the value from the shield cover and compares that with the bit sent by the driver logic. If the values match the required time, the system sends the newly generated bit. But if there is a mismatch between the send and receive data, alarm logic is triggered, and an alarm signal is sent to the protected design. With these blocks security of the protected designs under the shield cover is ensured.

3.3. Artificial Fish Swarm Algorithm

Applying the Artificial Fish Swarm Algorithm (AFSA) to maze generation involves framing the maze creation process as an optimization challenge. Each member of the artificial fish population represents a maze configuration, where their position reflects the maze layout, distinguishing between paths and walls.

Initially, a comprehensive, objective function is devised to assess the quality of a given maze configuration systematically. This function considers criteria like maze con-

nectivity, ensuring a continuous path from start to finish; maze complexity, balancing the number of paths and walls for desired difficulty; and maze solvability, necessitating a unique solution.

After defining the objective function, an initial population of artificial fish is generated, with each fish embodying a randomly created maze configuration. The objective function is then used to evaluate each configuration's quality. Fish movement dynamics within the maze configuration space involve strategically adding or removing walls to improve maze quality.

Individual and collective behavioral paradigms are integral to this algorithm. Individual behaviors include exploring, seeking new maze configurations, and exploiting and leveraging successful configurations. Fish communicate to exchange information on promising maze configurations, while memory mechanisms retain optimal configurations. Learning mechanisms adjust movement strategies based on past success.

The algorithm iteratively refines maze configurations until they meet termination criteria, such as a set number of iterations or satisfactory mazes. The optimal maze configuration, determined by the artificial fish, is extracted upon completion.

This AFSA adaptation for maze generation balances exploration and exploitation, leveraging the algorithm's ability to traverse solution spaces and learn from successful configurations. Fine-tuning the objective function and movement rules aligns the algorithm with specific maze generation needs.

3.4. RTL to GDSII Flow

The flow of RTL-to-GDS (Register Transfer Level to Graphic Design System) is crucial in digital integrated circuit (IC) design. It involves converting a high-level hardware description into a physical layout ready for manufacturing. This complex procedure involves multiple distinct stages which in as a flow diagram in figure 3.3,

with the choice of tools and methods usually depending on the preferences of the design team and the particular technology node being used.

RTL Design: The initial phase of the RTL-to-GDS flow involves the creation of the Register Transfer Level (RTL) description, utilizing hardware description languages such as Verilog or VHDL. Subsequently, synthesis tools are employed to translate the RTL description into a gate-level netlist, providing a representation of the logical structure of the design through standard cells derived from a technology library.

Logic Synthesis: Following RTL design, logic synthesis takes center stage. Specialized tools are crucial in executing the synthesis process. These software applications analyze RTL descriptions and map them to standard cells from technology libraries, optimizing for criteria like speed, area, and power consumption while maintaining functionality. They employ techniques such as logic restructuring and timing optimization, allowing designers to specify constraints like timing requirements and power targets. Integrated with other Electronic Design Automation (EDA) tools, they support hierarchical design methodologies, facilitating the management of large and complex designs. Examples include Synopsys Design Compiler and Cadence Genus, each offering unique features and optimizations for different technology processes.

Physical Planning: The flow proceeds to the physical planning stage, where the dimensions of the chip are defined and a preliminary floor plan is established. This includes allocating space for various functional blocks and the initial placement of cells. Specialized floorplanning tools play a vital role in coordinating this essential stage.

Placement and Clock Tree Synthesis (CTS): Placement follows, where the synthesized netlist is placed on the chip floorplan to optimize space and enhance performance and power efficiency. Concurrently, CTS is executed to design a clock distribution network, ensuring precise clocking across the chip with minimal skew and jitter. Automated or guided placement and CTS tools are used for these respective tasks.

Routing and Design Rule Checking (DRC): Routing is a critical phase in the physical design process of integrated circuits, where interconnections between different logic elements are established to create a physical layout. During routing, routing tools navigate through the chip's layout grid, determining the paths for wires connecting components such as gates, flip-flops, and other functional blocks. Routing aims to optimize the layout for factors like signal integrity, timing, power consumption, and manufacturability while adhering to design rules and constraints. This involves resolving conflicts, avoiding congestion, minimizing wire lengths, and ensuring proper spacing between wires to prevent crosstalk and other signal interference issues. Once routing is complete, a Design Rule Check (DRC) is performed to verify that the routed connections meet manufacturing requirements and do not violate design rules. Routing is crucial in translating the logical design into a physical layout that meets performance, power, and manufacturability targets. Cadence Innovus and Synopsys IC Compiler are the two major tools for these steps.

Physical Verification and Timing/Power Analysis: Physical verification proceeds, involving critical checks such as Layout vs. Schematic (LVS) verification to guarantee alignment between the physical layout and the intended schematic representation of the design. This verification ensures that the layout accurately reflects the intended circuit functionality. Concurrently, timing analysis is conducted to verify that the design meets required performance specifications, including timing constraints such as setup and hold times. Timing analysis tools assess signal propagation delays across the design, identifying critical paths and potential timing violations. Additionally, power analysis may be performed to evaluate power consumption across the design and ensure it aligns with specified power budgets. These verification steps are crucial for validating the integrated circuit design's correctness, performance, and reliability before fabrication.

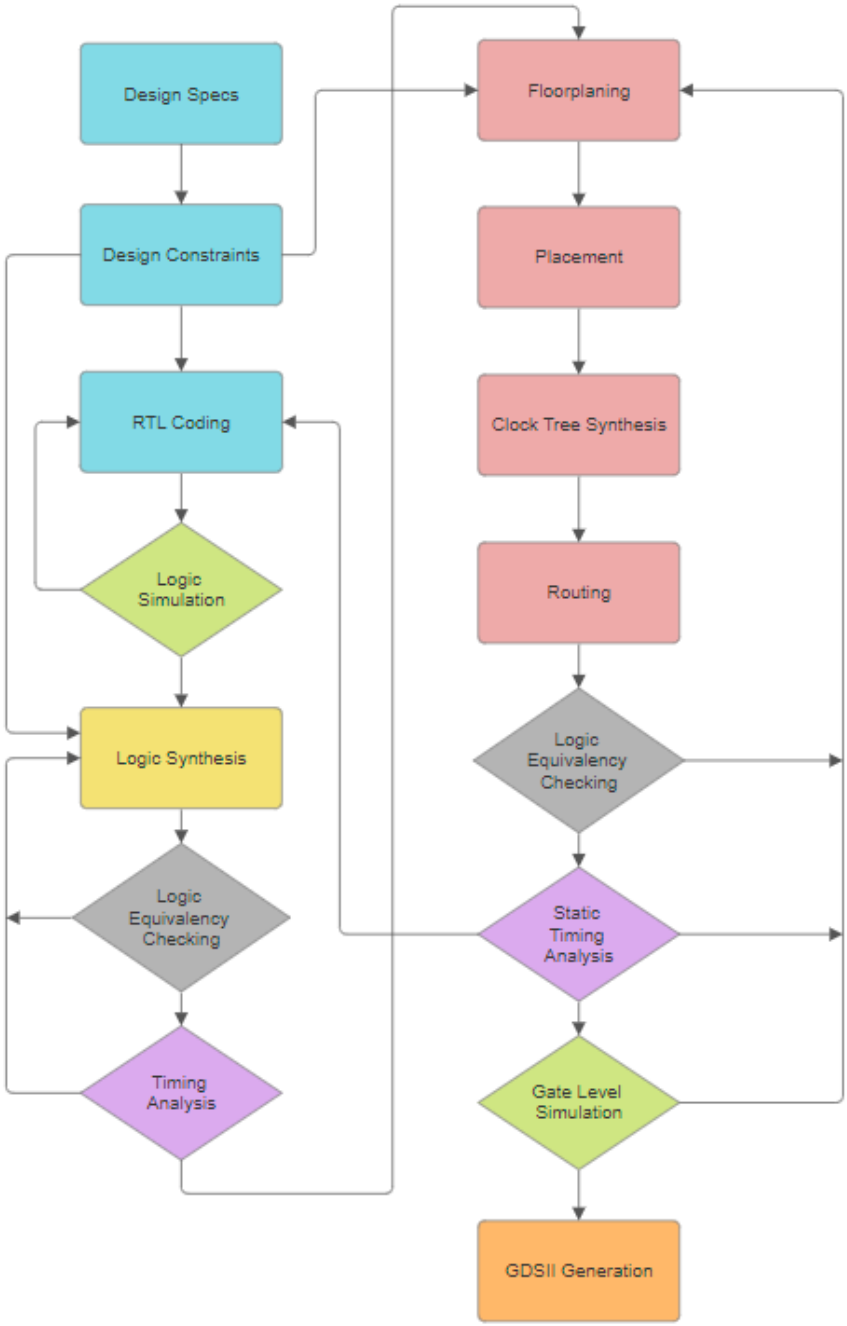


Figure 3.3. RTL to GDSII flow.

Final Output: The concluding steps involve generating final data, including the GDSII file, which encapsulates the information requisite for fabricating the integrated circuit. This involves tools dedicated to GDSII generation and the tape-out process.

Overall, the RTL-to-GDS flow is a methodical and iterative procedure, requiring the smooth coordination of diverse tools and approaches to effectively transform high-level design ideas into tangible physical outcomes.

Electronic Design Automation (EDA) tools offer significant advantages in developing integrated circuits but also have several disadvantages. Firstly, their cost can be prohibitive, particularly for small companies or individual designers, as the initial investment and ongoing support contracts can be substantial. Secondly, the complexity of EDA tools poses a challenge, requiring extensive training and expertise to utilize them effectively, which may lead to a shortage of skilled personnel. These tools often demand powerful computing resources, making them resource-intensive and potentially adding to hardware infrastructure costs. Despite these drawbacks, EDA tools remain indispensable in the semiconductor industry for their role in facilitating the design and fabrication of integrated circuits.

3.5. LEF File Format

The LEF constitutes a standardized file format employed within the realm of electronic design automation (EDA) for articulating the physical and electrical attributes of components within an integrated circuit (IC) library [10]. LEF files encapsulate critical details regarding library elements, including standard cells, macros, and other fundamental building blocks integral to IC design. The format consists of different components:

LEF files commence with a header section containing metadata, denoting the version of the LEF format in use and the units of measurement (e.g., microns, nanometers).

The main part of the LEF file outlines library elements, including macros, pins, and other components. Macros represent significant functional blocks or instances within the IC and provide details such as size, origin, and linked pins.

Pins indicate input and output links for library elements, providing information like pin names, directions (whether they are inputs, outputs, or bidirectional), associated layers, and other relevant attributes.

Layers are essential components that define the physical properties of various metal or dielectric layers utilized in the manufacturing process of integrated circuits. Details about layers include their names, types (such as metal or via), and specific characteristics such as thickness and spacing.

Spacing rules define the minimum spatial requirements between different layers, ensuring correct manufacturing and preventing potential issues like short circuits.

Obstacles correspond to designated areas on layers unsuitable for routing. They are often specified to prevent interference with other components or adhere to design rules.

LEF files are crucial for communication of design data across various EDA tools and stages of IC design. They provide a standardized format that simplifies the representation of physical characteristics essential for creating and verifying integrated circuit layouts.

3.6. Hamiltonian Cycle

In graph theory, a Hamiltonian cycle is a concept denoting a particular type of cycle within a graph. Named after the 19th-century mathematician William Rowan Hamilton, a Hamiltonian cycle in a graph G is formally defined as a closed walk that traverses each vertex of G exactly once and returns to its origin. A Hamiltonian cycle's defining characteristic is its ability to visit each vertex precisely once.

The structure of a Hamiltonian cycle is delineated by a sequence of vertices, wherein an edge connects each consecutive pair. The cycle concludes by linking the

terminal vertex back to the initial one. This definition encapsulates the fundamental property of visiting each vertex exactly once.

Determining the existence of a Hamiltonian cycle in a given graph and identifying such a cycle pose computational challenges. The Hamiltonian cycle problem is classified as NP-complete, indicating the absence of a known polynomial-time algorithm applicable to all instances. The computational complexity of this problem renders it particularly challenging, especially for large graphs.

The significance of Hamiltonian cycles extends to diverse applications, including network design, logistics, and optimization problems, underscoring their relevance in practical problem-solving and algorithmic considerations within various domains of study. An example of a Hamiltonian graph can be seen in figure 3.4(a) and its Hamiltonian cycle examples can be seen in figures 3.4(b), 3.4(c), 3.4(d).

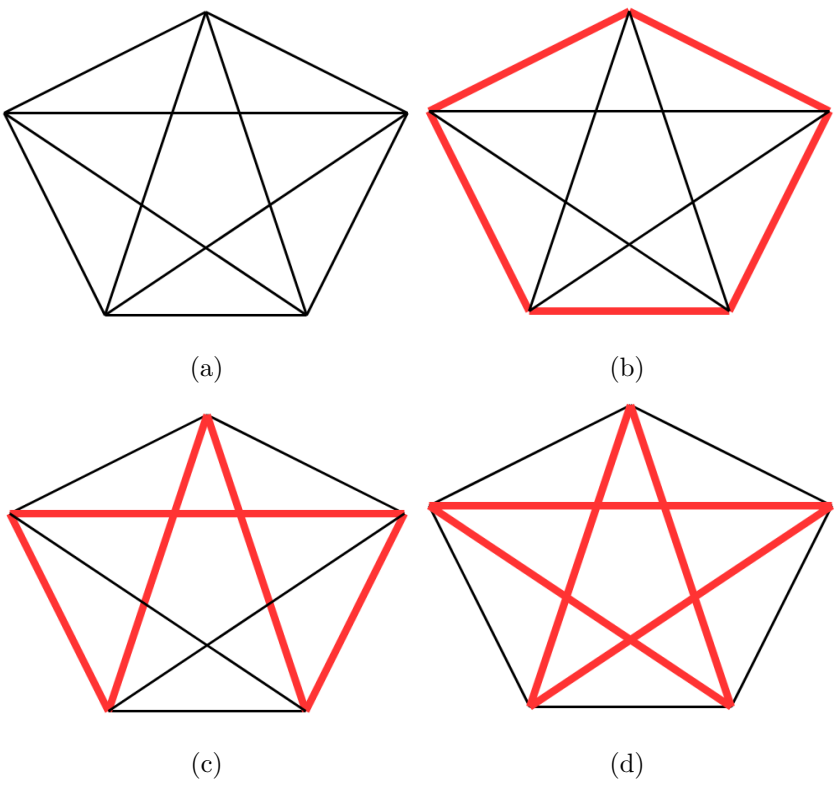


Figure 3.4. Hamiltonian graph (a) and its hamiltonian cycle examples (b)(c)(d).

4. METHODOLOGY

4.1. Using Maze as Random Shield

Active Shield architecture consists of digital control logic and shield block, as stated in chapter 3.2.1. Fundamentally, a shield block is one very long metal line with input and output pins. As it is stated in the section 3.6, Hamiltonian cycles also have one input and one output. Therefore shields can be considered as Hamiltonian cycle.

One major vulnerability of shields is being bypassed with metal deposition if they are too basic to be reverse-engineered. To reduce the risk of being bypassed, more complex shield designs are required. The effectiveness of the shields produced relies significantly on their level of complexity. Also, shields should have one input and one output connected to them, which satisfies the Hamiltonian Path condition.

There are various software algorithms to create paths. Maze algorithms are one of the best candidates to generate these shield layers because they can have 1 input and 1 output when constrained, and they generate paths that are more complex than simple lines, which come with the nature of a maze.

This flow has been designed to work with any type of maze generation algorithm. Depending on the complexity requirements, routing rules, or even the maze generation time constraints, any suitable algorithm can be included in the flow. To demonstrate the flow's capability, a previously used maze algorithm, the Artificial Fish Swarm Algorithm, was chosen. It is modified for the software integration and IC design implementation requirements.

4.1.1. Artificial Fish Swarm Algorithm for IC Design

The AFSA is a widely recognized algorithm previously utilized in research for its rapid capabilities in designing shields with the required complexity. In this study, AFSA has been chosen as the basis and adjusted to enhance the shield's adaptability, extending its functionality beyond the surface layer to include lower layers. This adaptation allows for the creation of intricate shapes by introducing obstacles strategically within the maze. These obstacles serve a dual purpose, potentially facilitating the passage of power VIAs at a later stage if needed.

The algorithm, written in Python, follows a series of well-defined steps. Initially, the dimensions of the labyrinth to be created are input in terms of unit squares in both axes. A fish matrix with these dimensions is created to track which fish are used and which are not during the algorithm's execution. Additionally, areas where the labyrinth is not intended to form are recorded in this matrix. Similarly, a connection matrix is created to contain information about which box in the finished labyrinth will connect to which box and from which direction. Finally, swarm and tracking stacks, to be used in the later stages of the algorithm, are also generated.

Once the necessary variables and matrices are created, an unobstructed box is randomly selected. The boxes surrounding the chosen box (fewer for corner boxes) are added to the tracking stack if there is no obstacle. This tracking stack is then used to select one box randomly. The feasible ways in which the selected box can be connected to the main labyrinth are then determined, and if there are multiple possible connections, one is randomly chosen among them. The connected box is then recorded in the connection matrix along with the connection direction. These steps continue until all connectable boxes in the desired area are connected.

In summary, the use of the AFSA algorithm in this study has enabled the creation of a sophisticated shield design that is adaptable to both the top layer and lower layers. The algorithm's step-by-step process ensures that all connectable boxes in the desired

area are connected, and the modified algorithm allows for the introduction of obstacles to facilitate the creation of complex shapes. Detailed flow steps are given in figure 4.1.

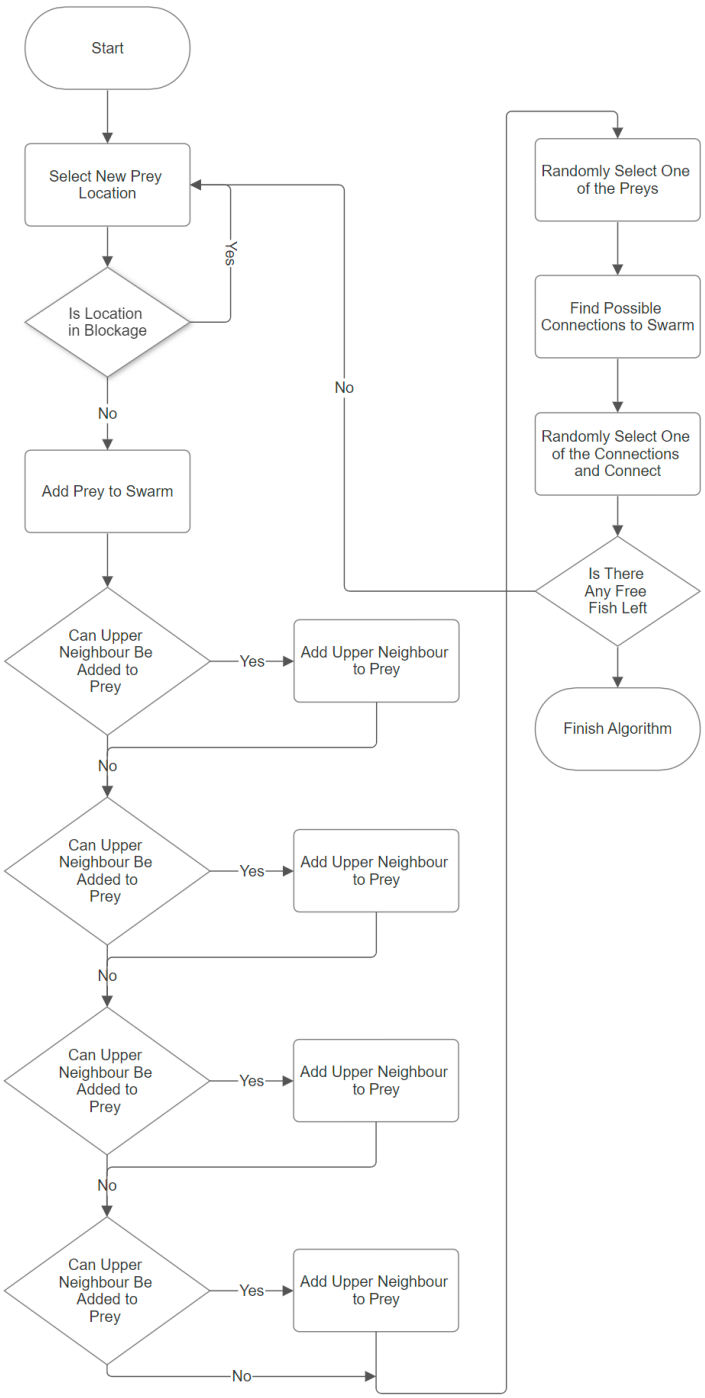


Figure 4.1. Artificial fish swarm algorithm flow diagram.

4.1.2. Converting Maze into Shield

Based on the outlined principles, implementing active shields requires a proportional number of inputs and outputs, mandating at least one input and one output. To achieve this, a specialized algorithm is crucial to derive a structure from the intricate maze generated in the preceding section. This algorithm systematically traverses all maze walls, starting from the initial point and concluding with the extraction of these walls into a new matrix, resulting in a Hamiltonian cycle.

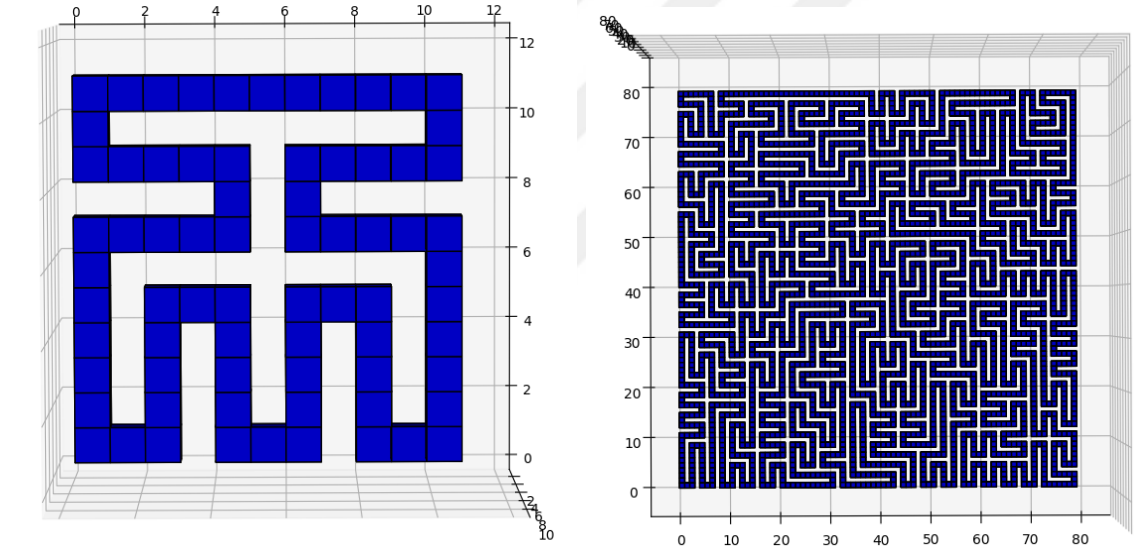
To accomplish this, the four corners of each box generated in the previous phase are transformed into path nodes. These nodes are interconnected based on the maze's directional connections, forming column and beam matrices that establish a detailed connection map of the nodes. The nodes' placement within these matrices depends on the connection type: vertically linked boxes are recorded in the column matrix, while horizontally linked ones are noted in the beam matrix.

Upon finalizing all assignments, the shield structure is fully constructed and ready for comprehensive visual inspection. The process of converting corners into path nodes and linking them according to labyrinth connections enables the creation of matrices accurately representing the structure's connectivity. This systematic approach ensures the resulting shield structure maintains the necessary input-output equilibrium crucial for active shields. As a result, the completed shield structure is well-suited for meticulous visual scrutiny, enhancing the efficacy of the design verification process overall. Example-generated mazes with 11 by 11 and 79 by 79 island sizes are given in figures 4.14(a) and 4.14(d), respectively. Also, a maze with power VIA blockages and analog block blockages can be seen in figures 4.3(a) and 4.3(b) respectively.

To increase the complexity of the shield, two layers of mazes can be used. They can be connected together with a special algorithm to combine them with vias. These mazes can be identical or completely different. Algorithm search for the best locations to add vias to bind them. As a result, multi-dimensional mazes can be generated.

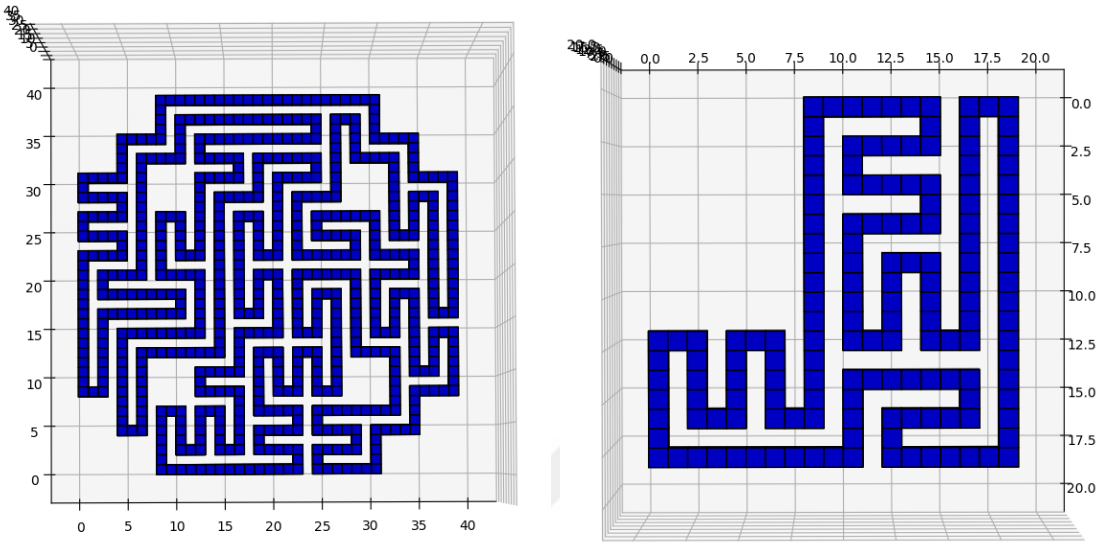
Raşit Rıdvan Turgut also stated in his mails that these mazes can be converted into multi-layer maze based shields.

Also, to reduce the shield generation duration, instead of creating one big maze, multiple small mazes can be used. These mazes can be two or three-dimensional. They can be combined with an algorithm. In another e-mail from Raşit Rıdvan, Turgut confirms that, with this algorithm, maze generation can be divided into multiple CPUs, and RAM usage can reduced.



(a) (b)

Figure 4.2. Shields of various sizes. (a) 11x11 shield, (b) 79x79 shield.



(a) (b)

Figure 4.3. Shields of various sizes with blockage. (a) 39x39 shield, (b) 19x19 shield.

4.2. Verifying Shield with 3D Visualization

EDA tools are crucial for the design flow of the active shields, but they have a few disadvantages. Firstly, long booting times which are given in table **??** can hinder productivity, especially in larger design projects requiring frequent software restarts. This delay can result in wasted time and reduced efficiency for design teams. Secondly, licensing for EDA tools often involves complex agreements and high costs. Licenses may be required for each individual user or machine, and additional fees may apply for accessing advanced features or updates. This licensing structure can be burdensome for small companies or individual designers with limited budgets.

Table 4.1. Place and Route tool durations.

Method	**Duration**
Only opening PnR Tool	37 seconds
Running Full Flow	410 seconds

Crafting complex shields can pose challenges in specific scenarios where relying solely on digital design tools may not be practical. Therefore, alternative methods without using any process node-related file are necessary to facilitate quick and effective visual validation of maze-based shield designs, identification of connection errors, and confirmation of the presence of a complete cycle.

A new Python function has been developed to produce an interactive three-dimensional model to meet this need. This model is a tool for assessing the intricacy of maze-based shield designs, identifying possible connection errors, and verifying the presence of a complete cycle. The main aim of this function is to simplify the visual inspection process, particularly in cases where starting the design program is time-consuming or impractical.

By embracing this comprehensive method, the Python function accelerates the verification process, allowing designers to evaluate complex maze-based shield designs quickly. This efficacy is especially beneficial when traditional design programs necessitate initialization scripts or files to commence. The interactive three-dimensional model produced by this function acts as a visual tool to confirm the accuracy and entirety of the shield design, enhancing the efficiency and dependability of the design verification process. Example generated 19 by 19 maze can be viewed in python program as given in figure 4.4.

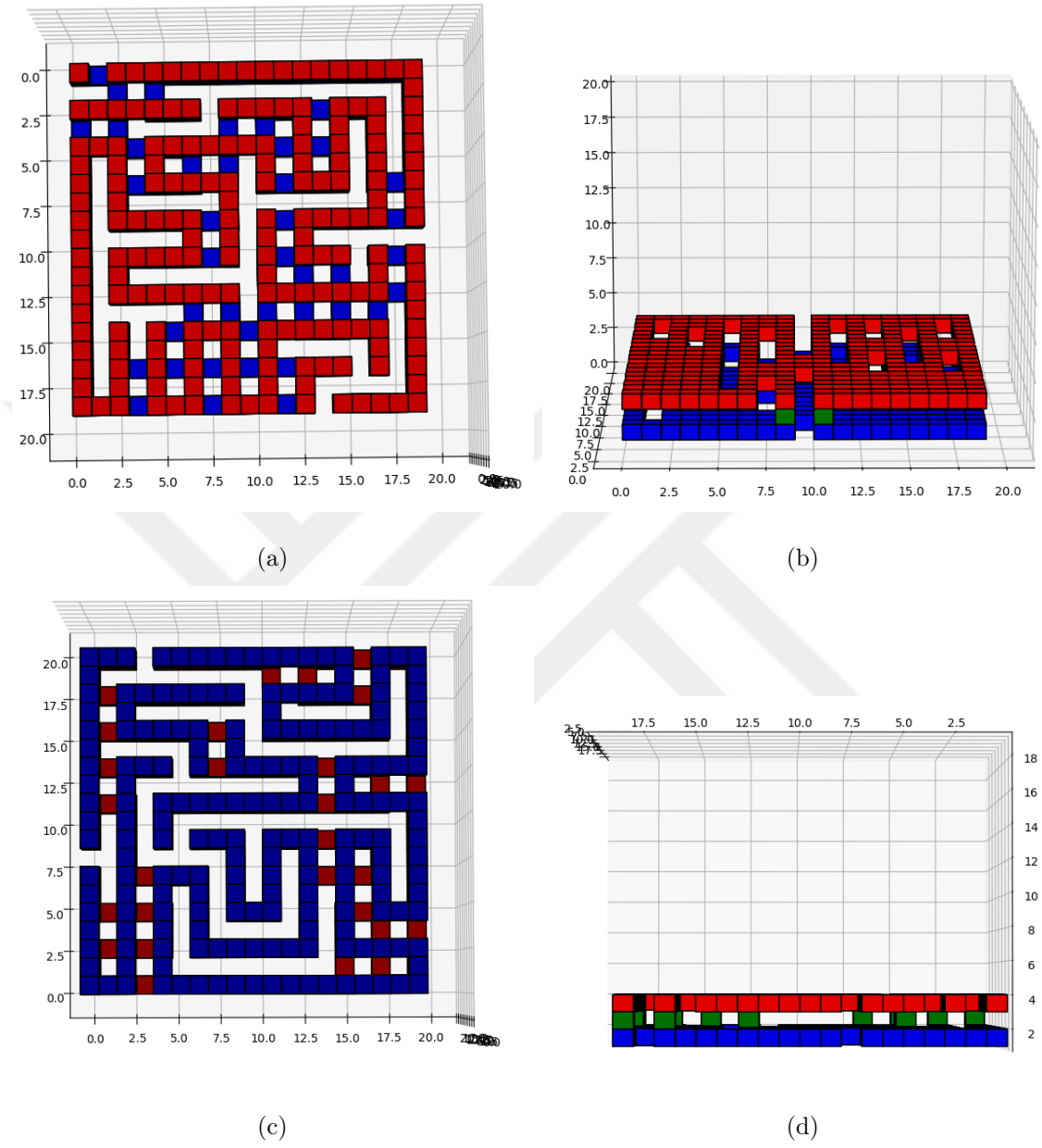


Figure 4.4. 19x19 two layer shield with different angles. (a) Top view, (b) angled view, (c) bottom view, (d) via view.

4.3. Converting Shield into Layout

The final stage of manufacturing the shield design involves converting the Python output into a file format compatible with Application Specific Integrated Circuit (ASIC) design tools. This file format must contain pertinent information such as dimensions,

layer details, and input-output locations. The LEF file format is chosen due to its universal compatibility with all design programs. Macro designs read from this file can be seamlessly integrated into the design without requiring further processing.

Before creating the LEF file, it is necessary to read the Tech LEF file on manufacturing technology to obtain minimum distance and width information for the desired layer(s) and VIAs. The minimum rules will be employed if the user-specified widths and distances are smaller than the minimum rules.

The initial step in creating the LEF file is to create the Header section, which contains information such as file version, length units, etc. Following this, the macro block is created with a name chosen by the user. Subsequently, the COVER type is selected. COVER represents a macro consisting solely of metals with no active layer. It is generally used for pad locations where wire bonding will be done, and it was chosen for this application because it satisfies the stipulated requirements. The rule definitions for the process can be seen in figure 4.5.

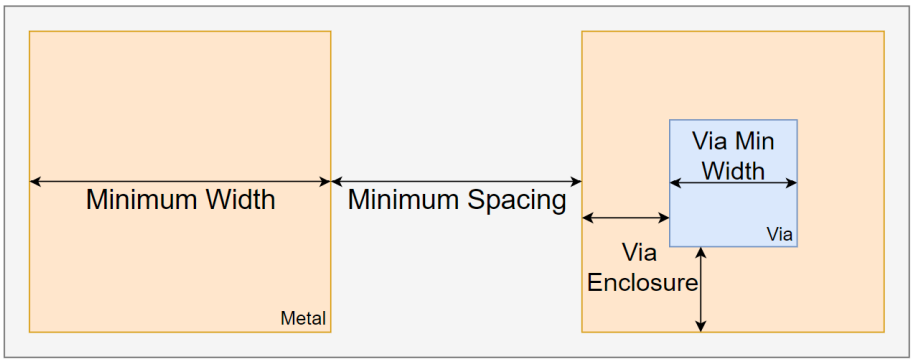


Figure 4.5. Design rules.

Equation 4.1 defines the spacing rule that defines the distance between two metal routing nets. The spacing distance cannot exceed the rule in the technology LEF file. Therefore, the process rule is used if the desired spacing rule is less than the minimum distance rule. In other cases, user input is assigned to the rule variable.

The same algorithm with one more input given in equation 4.2 applies to the minimum width rule, which defines the thickness of the metal routing nets. Metal layers must enclose the vias by an amount defined in the tech LEF file in every direction. Hence, metals should be thick as two times via enclosure and minimum via width. Also, there is another rule that minimum thickness of the metal routings. The algorithm compares these two constraints with the user input and selects the maximum of them to assign the variable.

The total size information of the macro is calculated with the rules given as

$$d_{spacing} = max(IN_U, R_{MS}) \tag{4.1}$$

where user input and minimum spacing rule are denoted as IN_U and R_{MS}, respectively and metal width of the routings are calculated as

$$d_{width} = max(IN_U, R_{MW}, 2 * R_{MVE} + R_{MVW}) \tag{4.2}$$

where user input, minimum width, minimum via enclosure and minimum via width rules denoted as IN_U, R_{MW}, R_{MVE} and R_{MVW}, respectively . The total size for each axis is calculated by multiplying spacing with island amount minus one and width by island amount, then adding them as in equation for x dimension

$$d_x = d_{spacing} * (Island\ Size_x - 1) + d_{width} * Island\ Size_x \tag{4.3}$$

and for y dimension

$$d_y = d_{spacing} * (Island\ Size_y - 1) + d_{width} * Island\ Size_y \tag{4.4}$$

Finally, the origin point is specified.

After generating the macro, the input-output pins of the macro are defined. Information on the pins' direction, type, layer, and area is provided at this point. The Figure 4.6 gives an example pin definition.

The final stage involves converting the information of the shield design generated in Python into routing information and, if there are multiple layers, into VIA information. Firstly, as explained in the previous section, islands are placed according

to distance and width rules. After this placement, if the gaps between them appear to be filled according to the matrix generated by Python, they are filled. VIAs are placed atop the islands' positions in the VIA matrix according to the VIA width rules. Example generated 100 by 100 maze can be seen in figures 4.7, 4.8 and 4.9. Also, complex shields with more than one maze can be generated, and examples can be seen in figures 4.10, 4.11, and 4.12.

```
VERSION 5.8 ;

BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

MACRO MAZE
        CLASS COVER ;
        SIZE 150.500000 BY 150.500000 ;
        FOREIGN enc 0.000000 0.000000 ;
        ORIGIN 0 0 ;
        PIN in[0]
                DIRECTION INPUT ;
                USE SIGNAL ;
                PORT
                        LAYER M9 ;
                                RECT 0.000000 0.000000 0.400000 0.400000 ;
                END
        END in[0]
        PIN out[0]
                DIRECTION OUTPUT ;
                USE SIGNAL ;
                PORT
                        LAYER M9 ;
                                RECT 1.900000 0.000000 2.300000 0.400000 ;
                END
        END out[0]
```

Figure 4.6. LEF header example.

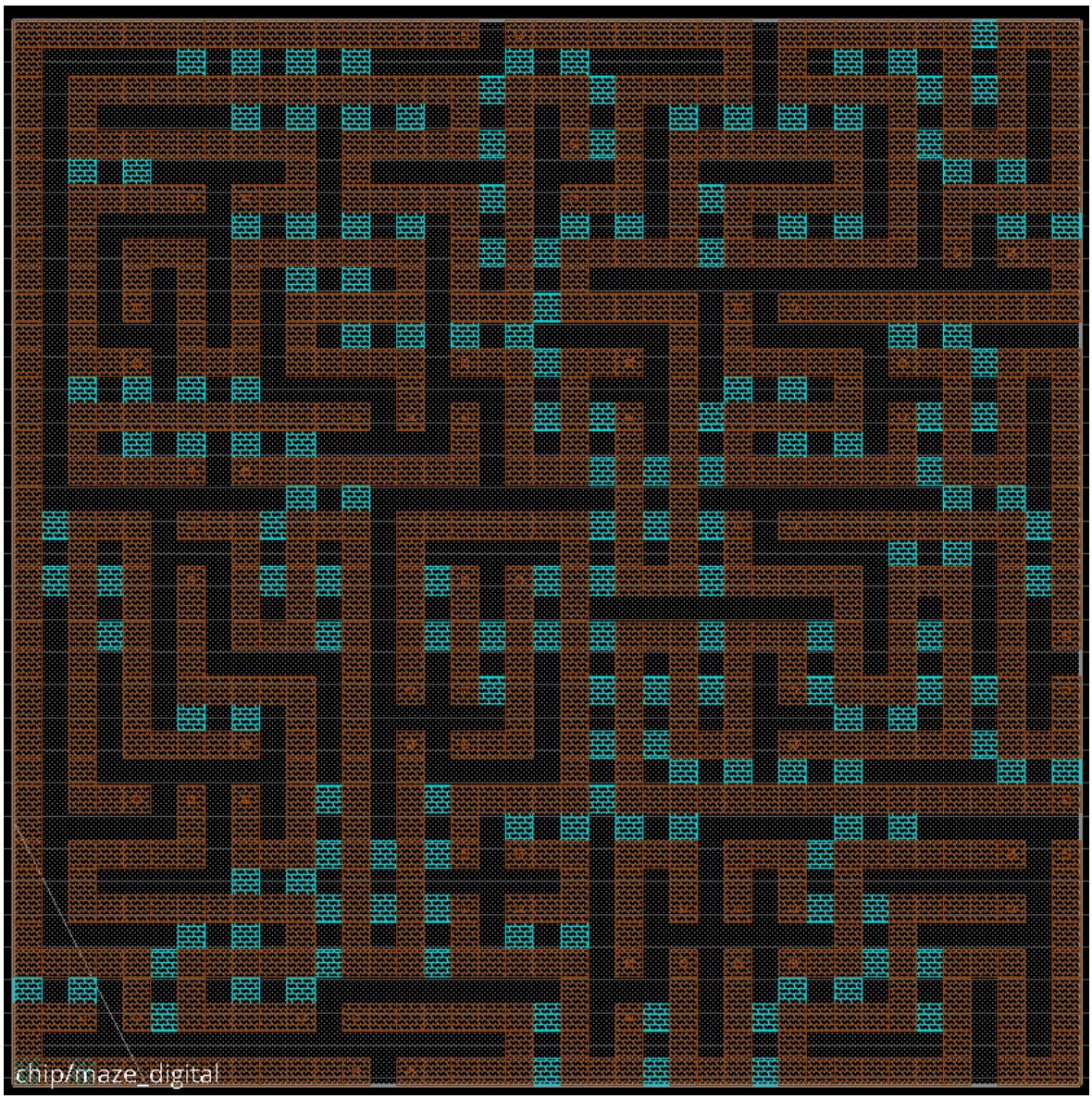


Figure 4.7. Layout view of the generated 40x40 maze with 2 layers.

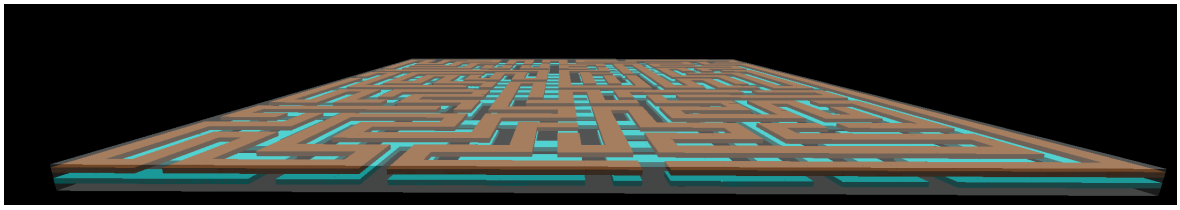
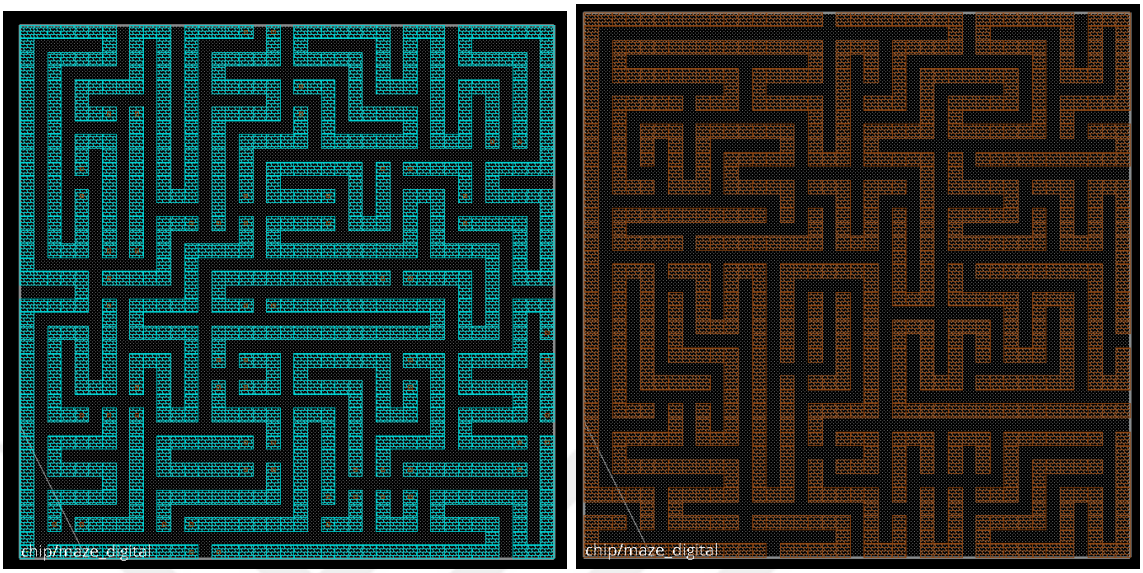
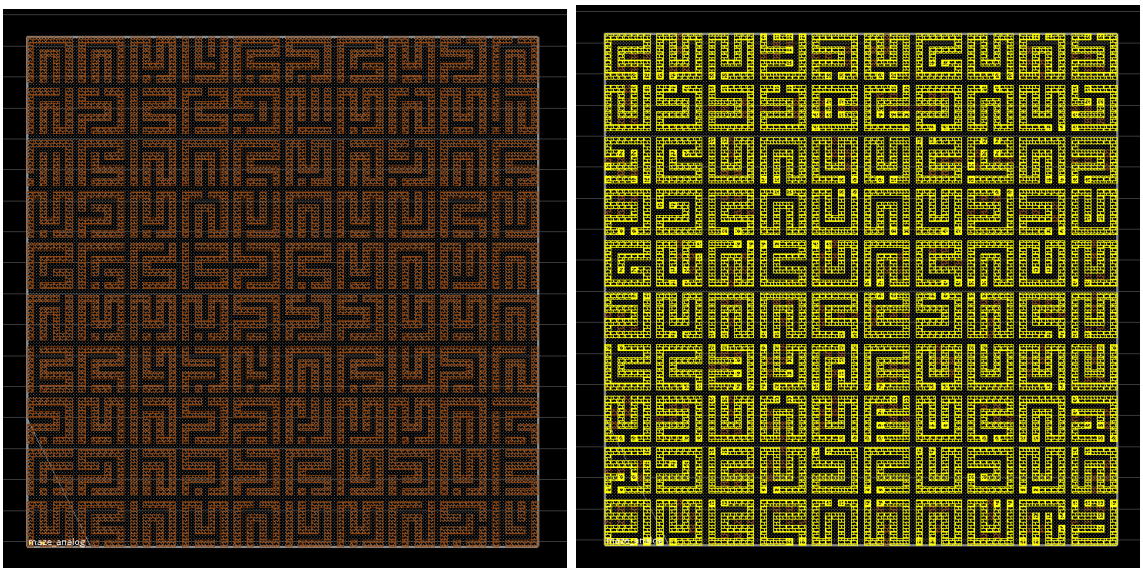


Figure 4.8. 3D Layout view of the generated 40x40 maze with 2 layers.



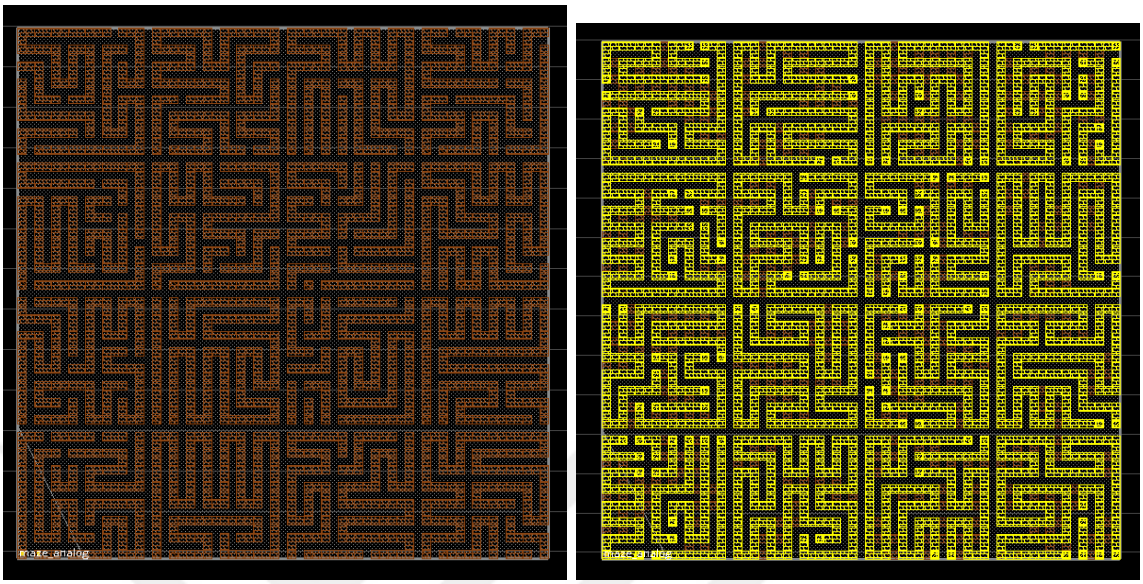
(a) (b)

Figure 4.9. Layers of generated 40x40 shield. (a) Bottom(M7) layer, (b) top(M8) layer.



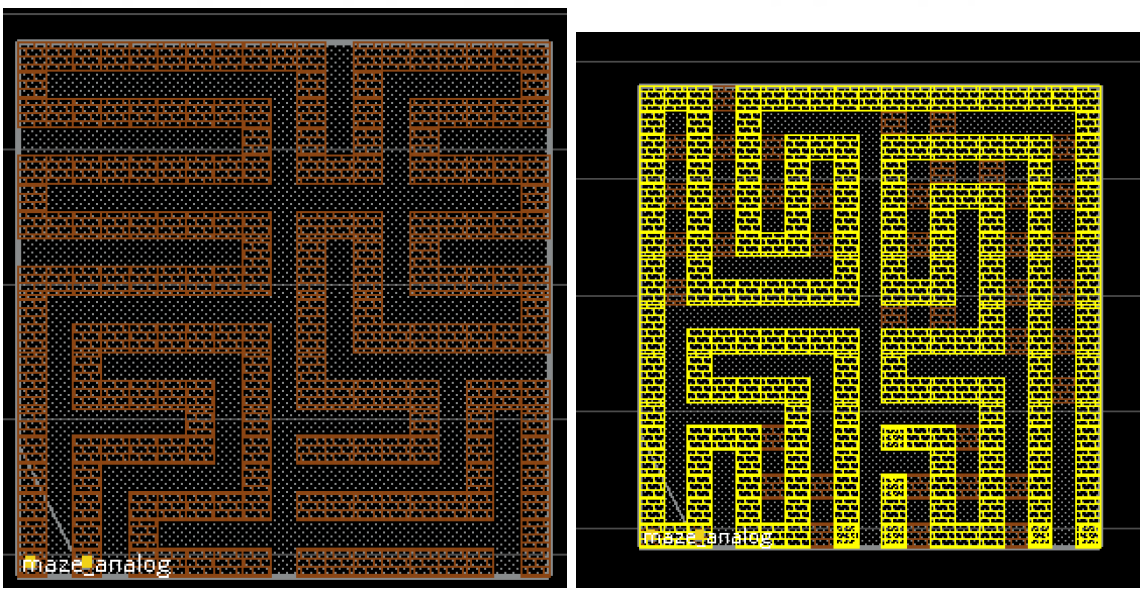
(a) (b)

Figure 4.10. Shield of 100 2 by 2 mazes. (a) Bottom(M8) layer, (b) top(M9) layer.



(a) (b)

Figure 4.11. Shield of 16 4 by 4 mazes. (a) Bottom(M8) layer, (b) top(M9) layer.



(a) (b)

Figure 4.12. Shield of 1 5 by 5 mazes. (a) Bottom(M8) layer, (b) top(M9) layer.

4.4. Implementing Maze into an ASIC Design with Its Checker Circuitry

Generated shield macros can be integrated into designs to be protected with shield control circuits. After the integration of the control circuit, EDA tools can be used to convert the RTL design into a physical design. This process requires several steps to generate the final product.

The initial step of this flow is to determine the timing and area constraints of the design. Timing and area constraints are fundamental aspects of IC design, influencing both performance and physical layout. Timing constraints govern the maximum allowable signal propagation delays within the IC, ensuring signals meet timing specifications such as setup and hold times. Efficient timing management is crucial to prevent issues like setup violations and ensure the reliable operation of the IC. On the other hand, area constraints dictate the maximum chip area available for the design, impacting the physical dimensions of the IC and its manufacturability. These constraints will be used in the individual steps of the flow.

When the constraints of the design are ready, flow can start with synthesis. In this step, the RTL file that contains a protected design and integrated shield control circuit is given as input with the process node-related standard cell timing information. For this step, Genus Synthesis Tool is used with a prepared script. RTL code is converted into a netlist with standard cells of the chosen process. Furthermore, timing reports of the generated netlist are produced. These timing reports are vital parts of the process to ensure that the desired operating frequency is achieved. LEC is conducted with Conformal LEC Tool to verify that the generated netlist completely matches the design.

The synthesized netlist in figure 4.13 is integrated into the top module, which contains the IO pad cells and shield macro block. With this integration, the routing stage guarantees the establishment of metal connections for external die communication and shield routings. Subsequently, the top module and timing and area constraints are

given to the Innovus PnR Tool as input. Also, a "Multi Mode Multi Corner" file containing timing information and the process-related design rules are created. When the initialization step is completed, the tool starts with the placement of the standard cells, IO cells, and shield macro. Next, clock tree synthesis is made for clock signal distribution. Finally, routing is made by the tool, and the final GDSII file is generated with necessary reports and timing information files.

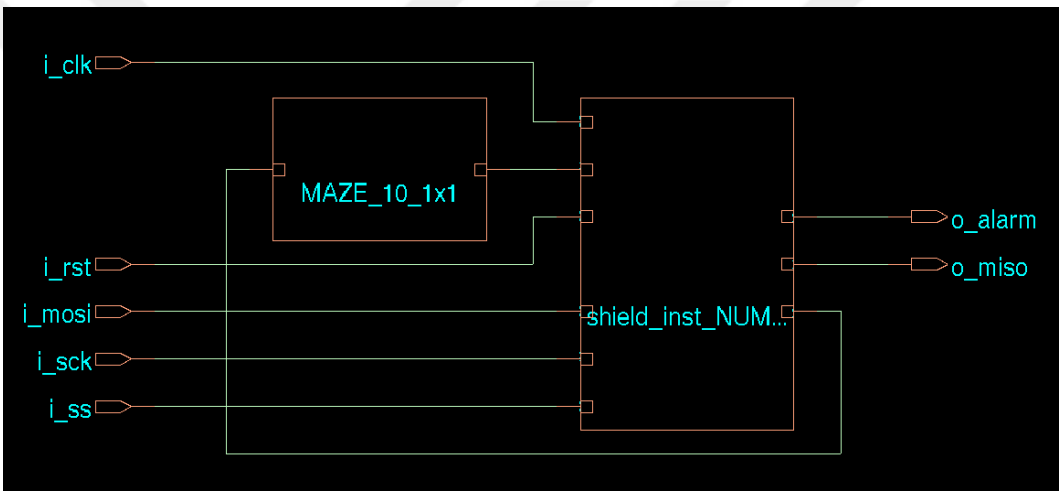
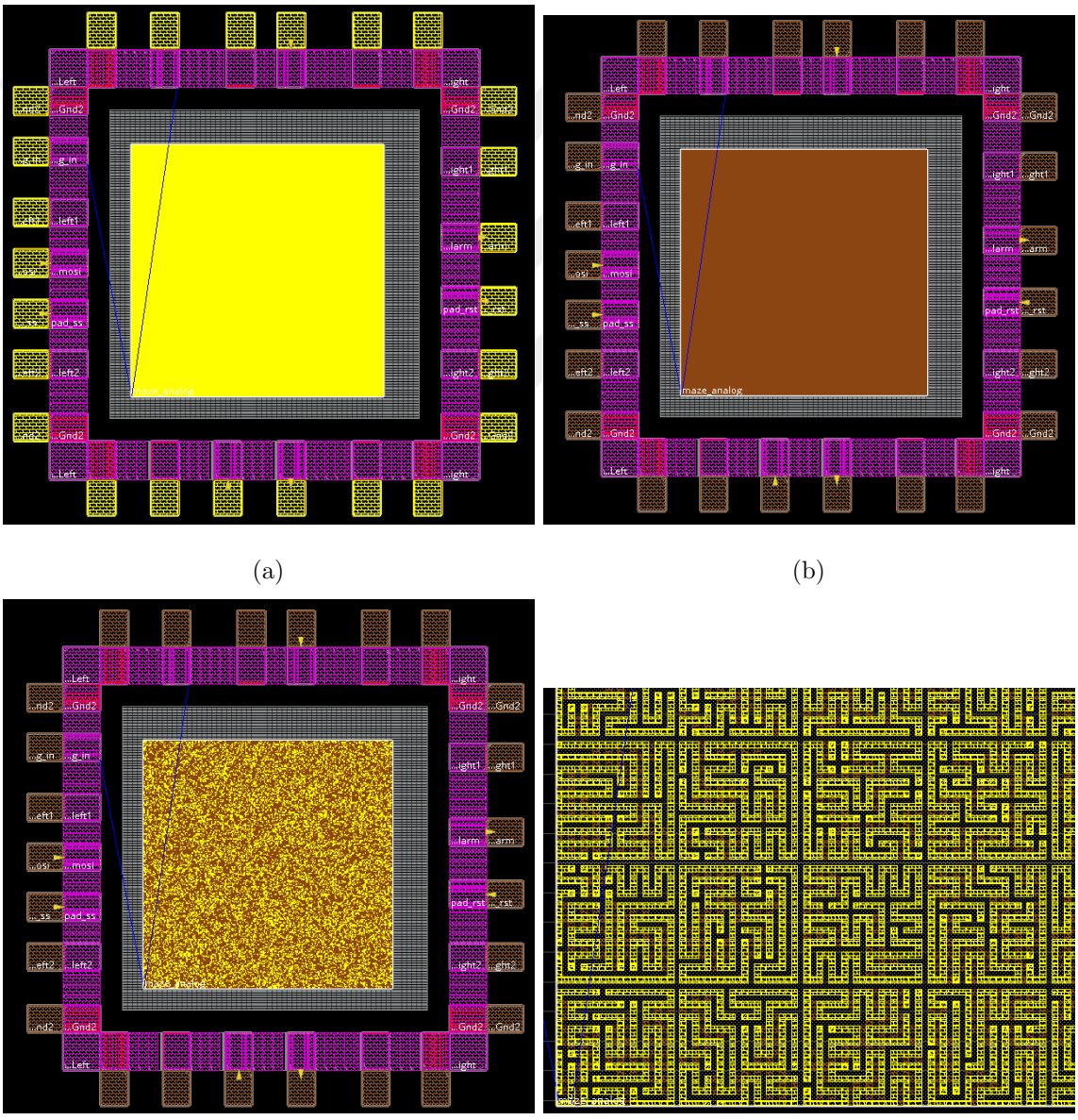


Figure 4.13. Synthesized schematic of top module.

The design undergoes a signoff verification process in the concluding phase to ensure alignment with Design Rule Check (DRC) regulations. The Layout vs. Schematic (LVS) examination confirms the accuracy of the generated netlist, while parasitic capacitance and resistance computations ascertain the digital design's operational suitability at specified frequencies. With the parasitic information gate level, simulation is run, and expected behavior is confirmed with the testbench. If every signoff step succeeds, the design is ready for the tapeout.

An exemplary design for this research thesis is created, including a control circuit based on a random number generator that communicates via the SPI protocol. The design operates at clock frequencies of up to 100 MHz and has a total area of

1mm*1mm, including IO cells and pads. For this flow, two process nodes are tested for process independence. Both TSMC 65LP in figure 4.16 and TSMC 65 GP in figure 4.17 nodes are used in the flow to generate the same design with the same maze layout. The sample design successfully passed all Design Rule Check (DRC) rules and Antenna rules for the process, and the TSMC 65LP node version was sent to TSMC for manufacturing.



(a) (b)

(c) (d)

Figure 4.14. Shield of 100 5 by 5 mazes. (a) Top layer, (b) bottom layer, (c) via layer, (d) close up.

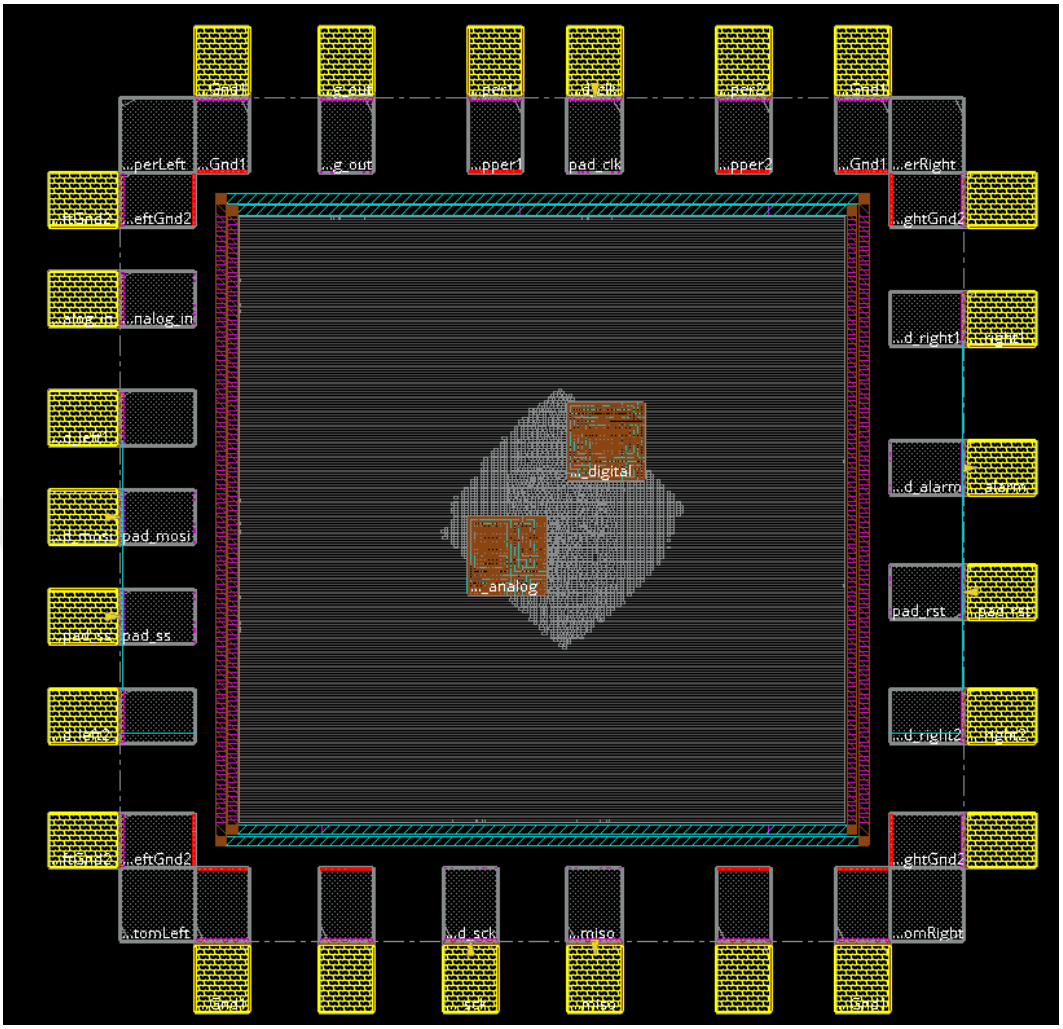


Figure 4.15. Place and Route layout for TSMC 65 LP Process.
(Routing layers removed for shield visibility).

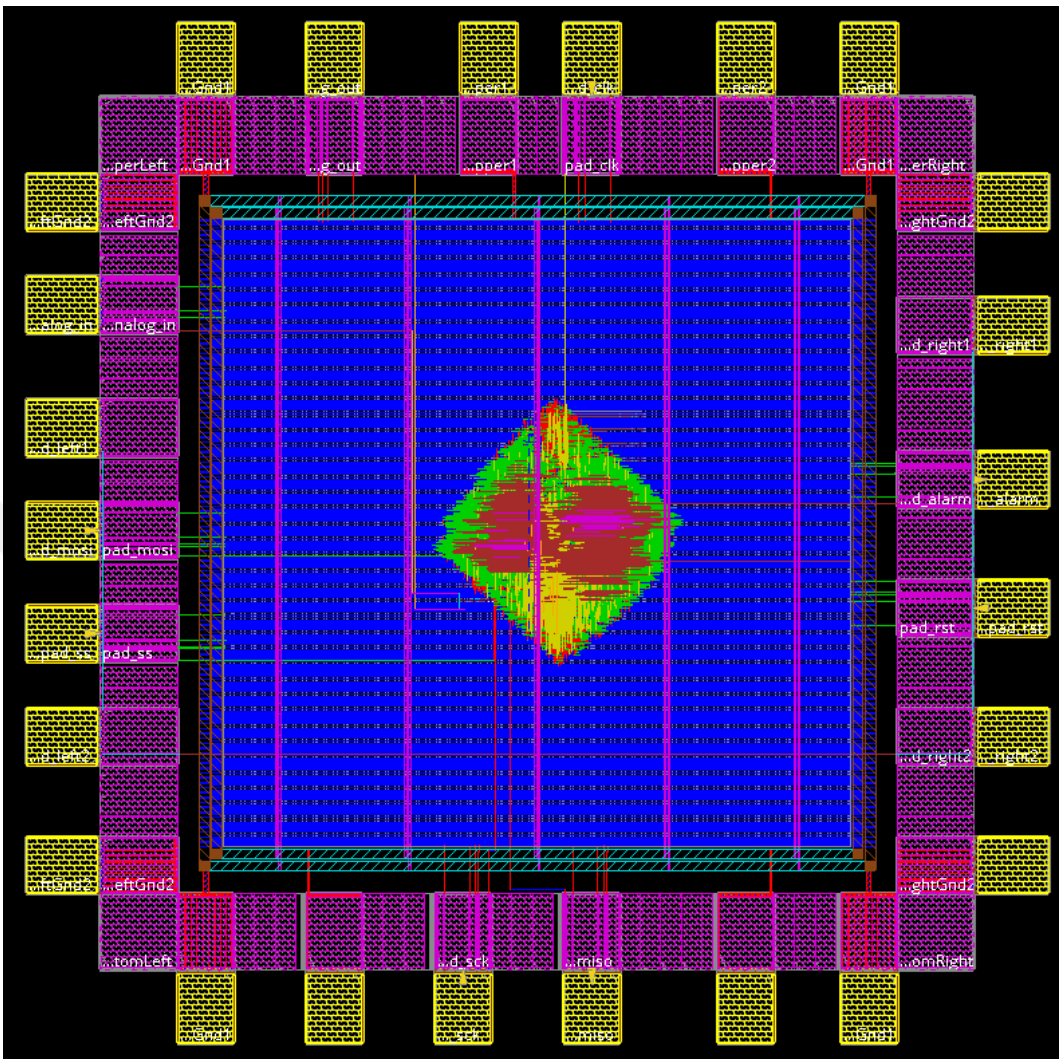


Figure 4.16. Place and Route layout for TSMC 65 LP Process.

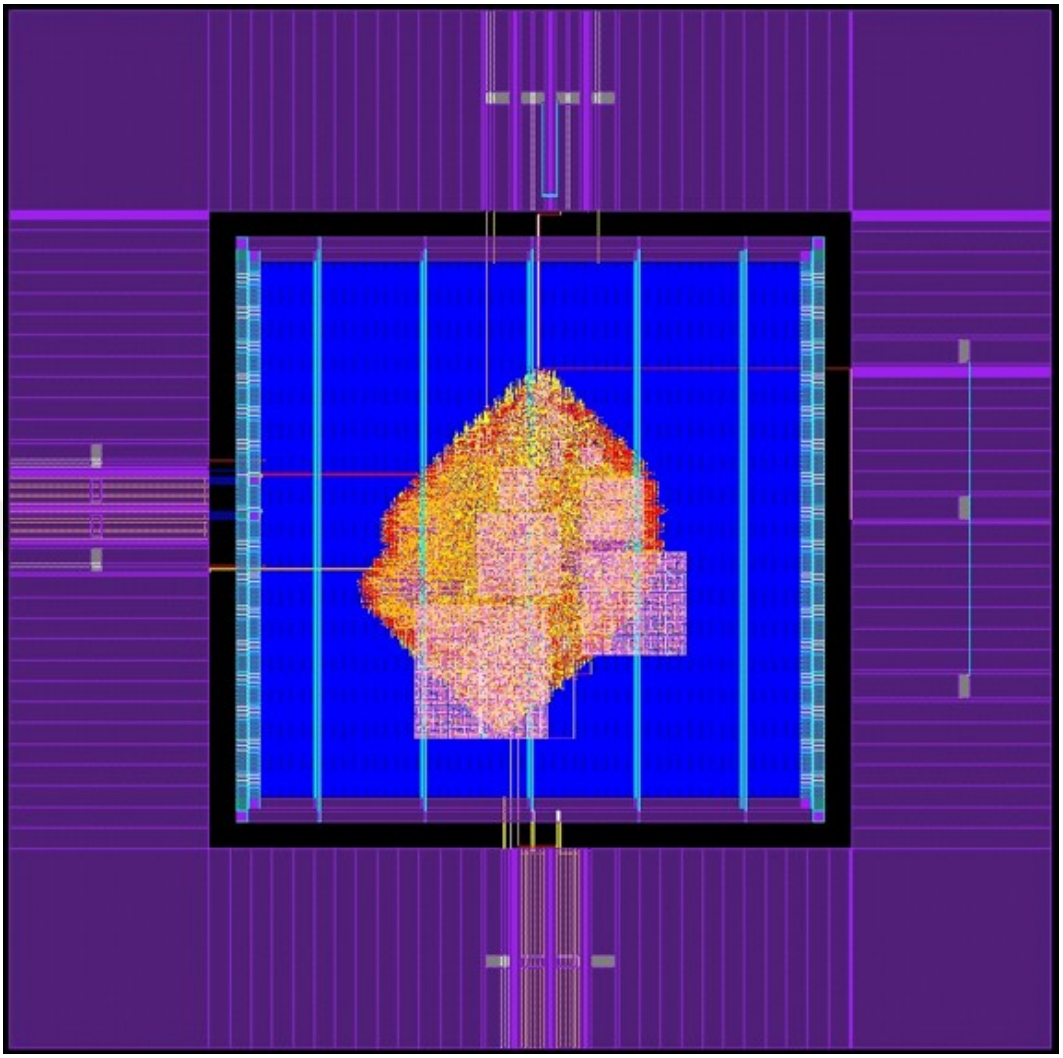


Figure 4.17. GDSII output of design for TSMC 65GP Process.

5. CONCLUSION

Secure IC design has been a focus of research for a long time, as the need for secure chips has increased with the emergence of new attack methods. However, the industry standards for secure chips are not clear yet, and the field is still developing.

To address this issue, this research aims to enhance the AFSA algorithm for generating holes for power vias, which are commonly required in ASIC designs. By modifying the AFSA algorithm, it can generate mazes that fit the circuit's shape instead of conventional square and rectangular shapes. These algorithms can also be used to generate two-layer shields afterward.

The workflow now includes a 3D visualization function, which helps speed up the prototyping process for the generated mazes without requiring any usage of EDA tools and process information. This function validates newly tested algorithms, determines the desired complexity of the mazes, and ensures that shapes beyond squares and rectangles are intentionally produced.

Moreover, the maze transforms into a shield, which is translated into a format suitable for ASIC design tools and integrated into the design process. This comprehensive research effort sets itself apart from previous studies that only addressed specific challenges in digital chip design. Instead, it covers all stages needed to transform an RTL code into a manufacturable product.

In the final step of this thesis, complete the flow to generate a GDSII file for ASIC fabrication. From synthesis to place and route steps, every script for the tools is tailored to integrate the shield and controller circuit with the design to be protected. Furthermore, example design in figure 5.1 is sent for tape-out to TSMC.

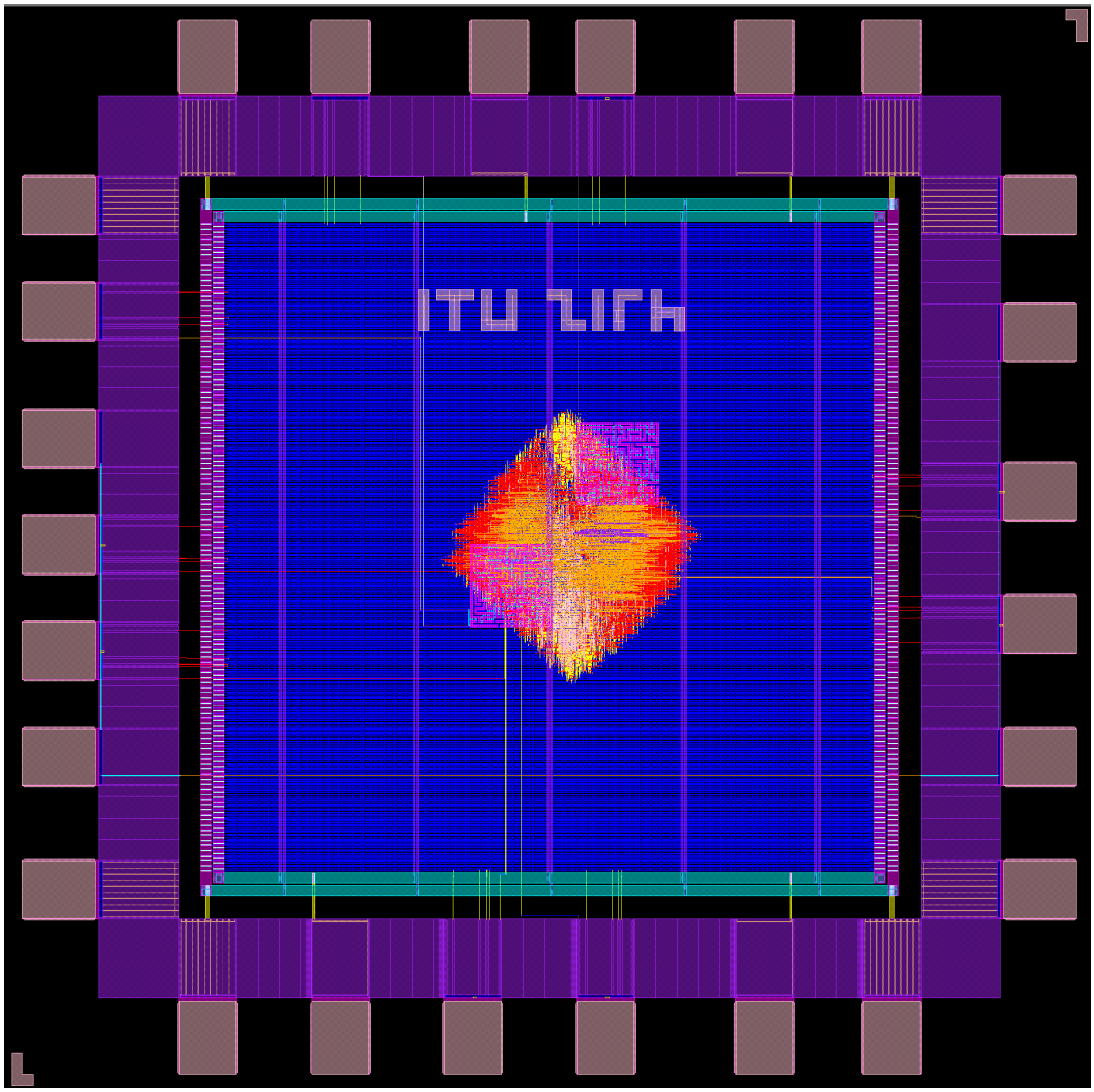


Figure 5.1. GDSII output of design for TSMC 65LP Process.

A process-independent, from scratch to final-chip flow is implemented with the necessary Python and TCL scripts. This research contributes significantly to the Secure IC design domain, promising to integrate larger, more intricate shield structures into lower layers with enhanced speed and efficiency.

REFERENCES

1. Kocher, P., J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution", *2019 IEEE Symposium on Security and Privacy*, pp. 1–19, 2019.

2. Matsuo, J., N. Toyoda and I. Yamada, "Nanofabrication Technology by Gas Cluster Ion Beams", *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena*, Vol. 14, No. 6, pp. 3951–3954, 1996.

3. Xin, R., Y. Yuan, J. He, S. Zhen and Y. Zhao, "Random Active Shield Generation Based on Modified Artificial Fish-swarm Algorithm", *Computers Security*, Vol. 88, p. 101552, 2020.

4. Wang, K., Y. Gu, T. Zhou and H. Chen, "Multi-Pair Active Shielding for Security IC Protection", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 38, No. 12, pp. 2321–2329, 2019.

5. Wang, H., Q. Shi, A. Nahiyan, D. Forte and M. M. Tehranipoor, "A Physical Design Flow Against Front-Side Probing Attacks by Internal Shielding", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 39, No. 10, pp. 2152–2165, 2020.

6. Ngo, X. T., J.-L. Danger, S. Guilley, T. Graba, Y. Mathieu, Z. Najm and S. Bhasin, "Cryptographically Secure Shield for Security IPs Protection", *IEEE Transactions on Computers*, Vol. 66, No. 2, pp. 354–360, 2017.

7. Cioranesco, J.-M., J.-L. Danger, T. Graba, S. Guilley, Y. Mathieu, D. Naccache and X. T. Ngo, "Cryptographically Secure Shields", pp. 25–31, 2014.

8. Vidaković, M. and D. Vinko, "Hardware-Based Methods for Electronic Device Protection against Invasive and Non-Invasive Attacks", *Electronics*, Vol. 12, No. 21, p. 4507, 2023.

9. Katz, E., M. Avital and I. Levi, "Refined Analytical EM Model of IC-Internal Shielding for Hardware-Security and Intra-Device Simulative Framework", *IEEE Access*, p. 1, 2024.

10. Cadence Design Systems, "LEF/DEF Language Reference", `https://www.ispd.cc/contests/18/lefdefref.pdf`, accessed on January 06, 2024.

APPENDIX A: SIMPLIFIED LEF FILE EXAMPLE

Sample LEF file example can be seen in Figure A.1. In this file, small shield with a name "MAZE" is created. with a size 3.5 to 3.5 microns. It has only one input and one output. AP and M9 routing layers and RV via layer selected for this shield generation.

```
VERSION 5.8 ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

MACRO MAZE
    CLASS COVER ;
    SIZE 3.500000 BY 3.500000 ;
    FOREIGN enc 0.000000 0.000000 ;
    ORIGIN 0 0 ;
    PIN in[0]
        DIRECTION INPUT ;
        USE SIGNAL ;
        PORT
            LAYER M9 ;
            RECT 0.000000 0.000000 0.500000 0.500000 ;
        END
    END in[0]
    PIN out[0]
        DIRECTION OUTPUT ;
        USE SIGNAL ;
        PORT
            LAYER M9 ;
            RECT 1.000000 0.000000 1.500000 0.500000 ;
        END
    END out[0]
    OBS
        LAYER AP ;
        RECT 0.000000 0.000000 0.500000 0.500000 ;
        RECT 0.500000 0.000000 1.000000 0.500000 ;
        RECT 1.000000 0.000000 1.500000 0.500000 ;
        LAYER M9 ;
        RECT 0.000000 0.000000 0.500000 0.500000 ;
        RECT 1.000000 0.000000 1.500000 0.500000 ;
        RECT 2.000000 0.000000 2.500000 0.500000 ;
        RECT 2.500000 0.000000 3.000000 0.500000 ;
        LAYER RV ;
        RECT 2.050000 3.050000 2.450000 3.450000 ;
        RECT 3.050000 3.050000 3.450000 3.450000 ;
    END
END MAZE

END LIBRARY
```

Figure A.1. LEF file example