The American University in Cairo

CENTER OF NANOELECTRONICS & DEVICES

CND 221: Advanced Full Custom

Final Project Submission

"Simplified-Data Encryption standard (S-DES)"

Section: 18 & 16
Group: 2

Submitted by:

Student Name	ID
Mohamed Yasser Mohamed	V23010367
Karim Emad El-Din	V23010546
Zyad Ahmed Fawzy	V23010332
Youssef Ahmed Hassan	V23009895
Taha Zaki Mohamed	V23010462

Submitted to:

Dr. Esraa Swillam

Eng. Mostafa Mahmoud

May 2024

Acknowledgement

We would like to extend our deepest gratitude to **Dr. Esraa Swillam** for her exceptional oversight and facilitation of this training program. Her unwavering support and commitment to nurturing our skills in Full Custom design have been invaluable.

Additionally, we would like to express our sincere thanks to **Eng. Mostafa Mahmoud** for his invaluable assistance throughout the project. His dedication and support in helping us overcome challenges were greatly appreciated.

We also want to thank all the **CND team members** for their contributions to this outstanding training program. Your collective efforts have provided us with a comprehensive and enriching learning experience. Thank you all for your dedication and support.

Abstract

This technical report presents the design, implementation, and verification of a Simplified Data Encryption Standard (S-DES) system. S-DES, a reduced-complexity variant of the Data Encryption Standard (DES), retains the core cryptographic principles while operating on smaller parameters. The project focuses on encrypting 8-bit plaintext blocks using a 10-bit key, following a sequence of permutations and substitutions like those in DES. The report details the system's architecture, subsystem design, and implementation phases, including key generation, initial and inverse permutations, the Feistel function, and data switching.

Comprehensive simulations ensure the correctness of individual blocks and the integrated system. Layout, routing, and placement are meticulously performed, with parasitic extraction and re-simulation conducted to assess performance impacts. The project highlights the importance of accurate initial specifications and incremental development while addressing challenges such as the complexity of permutations and parasitic effects. This implementation of S-DES serves as an educational tool, reinforcing fundamental cryptographic concepts and practical digital design methodologies.

Table of Content

Chapter 1: Introduction

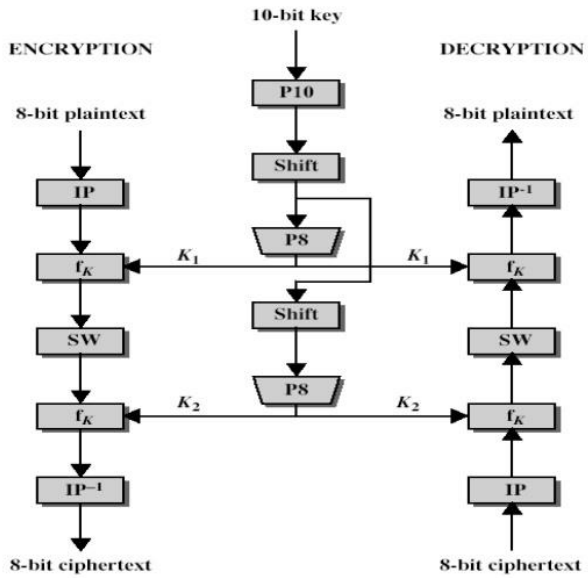
Background

In today's interconnected world, data security has become a paramount concern across various sectors including military, business, and personal communications. The need to protect sensitive information from unauthorized access and ensure privacy is critical in maintaining the integrity and trustworthiness of data systems. Encryption, the process of converting readable data into an encoded format, plays a vital role in securing data, making it accessible only to authorized users.

The Data Encryption Standard (DES) is a significant milestone in the field of cryptography. Developed by IBM cryptographer Horst Feistel in the 1970s and standardized by the National Institute of Standards and Technology (NIST) in 1976, DES has been extensively used for secure data transmission. Despite its historical importance, the 56-bit key length of DES is now considered inadequate against modern computational power, rendering it insecure for contemporary applications. Nonetheless, the principles of DES have laid the foundation for subsequent cryptographic algorithms.

Project Objective

The primary objective of this project is to implement a Simplified Data Encryption Standard (S-DES). S-DES is a pedagogical version of DES that retains the essential characteristics and steps of DES but operates on smaller parameters. Specifically, S-DES encrypts 8-bit blocks of plaintext using a 10-bit key to produce 8-bit ciphertext. This simplified model allows for an easier understanding of the underlying cryptographic processes, making it an excellent educational tool for studying encryption techniques.



Scope of the Project

This project encompasses the following phases:

1. System Specification: Define the system's functional requirements and outline the high-level design.

2. Design and Verification: Develop the hierarchical structure of subsystems, ensuring each subsystem is designed, implemented, and verified incrementally.

3. Implementation: Create schematics and cell designs for each subsystem, perform layout, routing, and placement.

4. Simulation and Testing: Simulate individual blocks and the integrated system to ensure correctness and propose comprehensive testing methodologies.

5. Evaluation: Analyze the impact of parasitic elements, calculate performance metrics such as area, power, and throughput, and compare with initial estimates.

Importance of Simplified DES (S-DES)

S-DES serves as a valuable tool for understanding the fundamental concepts of symmetric-key encryption. By working with smaller data blocks and key sizes, students and researchers can gain insights into the complexities of encryption and decryption processes without the overwhelming details present in full-scale DES. This simplified approach aids in demystifying the intricate operations of permutations, substitutions, key generation, and the Feistel structure, providing a clear pathway to mastering advanced cryptographic techniques.

Structure of the Report

The report is structured to provide a comprehensive overview of the S-DES implementation process. It begins with a detailed discussion of the research and background, followed by an exploration of the system's function and architecture. Subsequent sections delve into the design and implementation of subsystems, schematic and cell designs, and meaningful simulation results. The report also covers proposed testing methodologies and reflects on the lessons learned, challenges faced, and unexpected findings encountered during the project.

By systematically detailing each phase of the design and implementation process, this report aims to demonstrate the practical application of cryptographic principles through the development of a simplified yet functional encryption system.

Chapter 2: Research & Background

Historical Context of DES

The Data Encryption Standard (DES) was developed in the early 1970s by IBM, led by cryptographer Horst Feistel. DES was adopted as a federal standard for encrypting sensitive information by the National Institute of Standards and Technology (NIST) in 1976. The algorithm gained widespread acceptance due to its balance of security and efficiency, becoming the de facto encryption standard for numerous applications, including financial transactions and secure communications.

Importance of DES

DES operates on 64-bit blocks of data using a 56-bit key. Its strength lies in its use of the Feistel network, a structure that divides the data block into two halves and processes them through multiple rounds of permutation and substitution. This method introduces confusion and diffusion, key concepts in cryptography that obfuscate the relationship between the plaintext, ciphertext, and key, making it difficult for attackers to decipher the data without the correct key.

Despite its initial success, the advancement of computational power and cryptanalysis techniques has rendered DES insecure. Specifically, the relatively short 56-bit key length makes it vulnerable to brute-force attacks, where an attacker systematically tries all possible keys until the correct one is found. As a result, DES has been largely replaced by more secure algorithms, such as the Advanced Encryption Standard (AES).

Simplified Data Encryption Standard (S-DES)

S-DES is a simplified version of DES designed for educational purposes. It retains the core principles of DES but operates on smaller data blocks and key sizes, making it easier to understand and implement. S-DES encrypts 8-bit blocks of plaintext using a 10-bit key, producing 8-bit ciphertext. This simplification allows students and researchers to explore the mechanics of encryption and decryption without the complexity of full-scale DES.

Key Components of S-DES

- **Initial and Inverse Permutation**

The initial permutation (IP) reorders the bits of the plaintext block to enhance diffusion. The inverse permutation (IP^{-1}) reverses this reordering at the end of the encryption process to produce the final ciphertext.

- **Subkey Generation**

S-DES uses a 10-bit key to generate two 8-bit subkeys, K1 and K2, which are used in the encryption rounds. The subkey generation process involves permutations and left shifts to derive the subkeys from the original key.

- **Feistel Function**

The Feistel function (fK) is the heart of the S-DES encryption process. It involves expanding and permuting the right half of the data block, mixing it with a subkey, and then applying S-box substitutions and a P4 permutation. The result is then XORed with the left half of the data block.

- **Expansion and Substitution**

In the expansion step, the 4-bit right half of the data block is expanded to 8 bits. This expanded block is then XORed with a subkey, and the result is passed through two S-boxes, which perform non-linear substitutions. The output of the S-boxes is permuted using a P4 permutation.

- **S-Boxes**

S-Boxes are substitution boxes that introduce non-linearity into the encryption process. They take a 4-bit input and produce a 2-bit output, which helps in creating a complex relationship between the plaintext and ciphertext.

Relevance of S-DES in Cryptographic Education

S-DES is an invaluable tool for teaching and understanding the basics of symmetric-key encryption. Its simplified design allows students to focus on the core concepts without getting overwhelmed by the complexities of more advanced algorithms. By implementing S-DES, students gain hands-on experience with key cryptographic principles such as permutations, substitutions, key generation, and the Feistel structure. This foundational knowledge is crucial for understanding more complex encryption schemes and their applications in securing data.

Chapter 3: The Function of the System

Overview

The Simplified Data Encryption Standard (S-DES) system is designed to provide a clear and practical demonstration of symmetric-key encryption principles. It encrypts 8-bit blocks of plaintext using a 10-bit key, producing 8-bit blocks of ciphertext. This chapter outlines the detailed functional process of the S-DES system, including its encryption and decryption operations.

Encryption Process

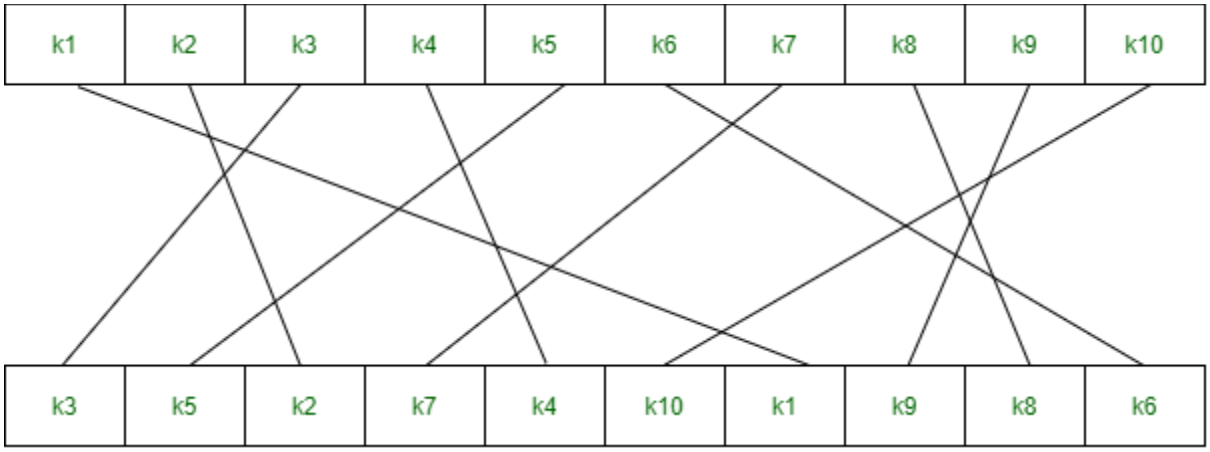
The S-DES encryption process consists of the following steps:

Initial Permutation (IP):

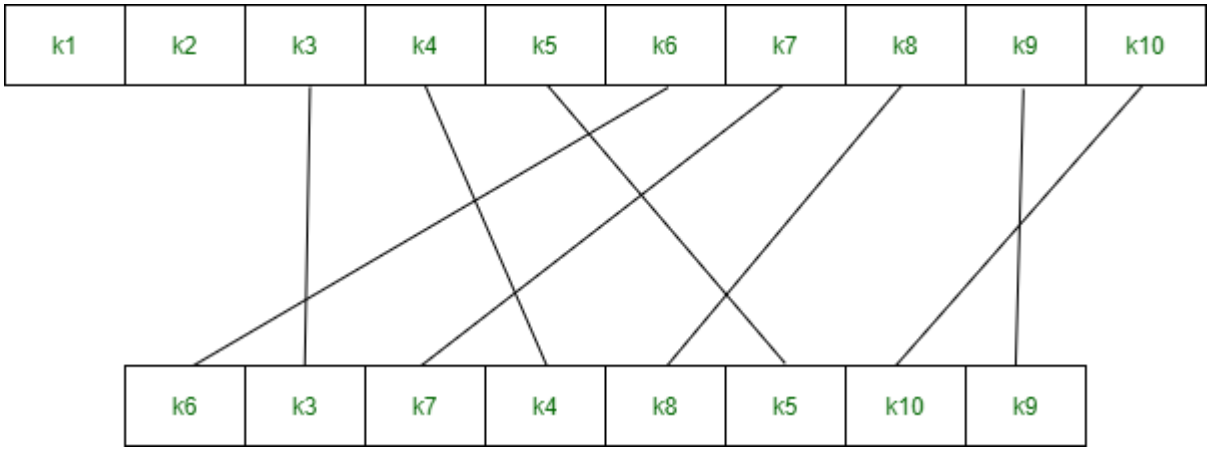
The 8-bit plaintext block undergoes an initial permutation (IP) that rearranges its bits according to a predefined permutation table. This step enhances diffusion by spreading the plaintext bits across the block.

Subkey Generation:

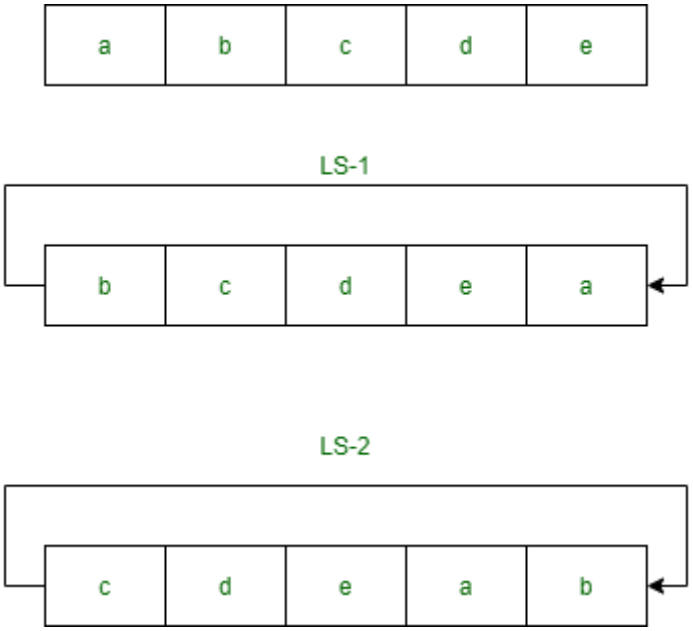
- A 10-bit key is used to generate two 8-bit subkeys, K1 and K2, through a series of permutations and left shifts.
- P10 Permutation: The 10-bit key is permuted using the P10 table.



- Left Shifts: The permuted key is split into two 5-bit halves, each of which is subjected to left circular shifts.
- P8 Permutation: The shifted halves are recombined and permuted using the P8 table to generate K1.

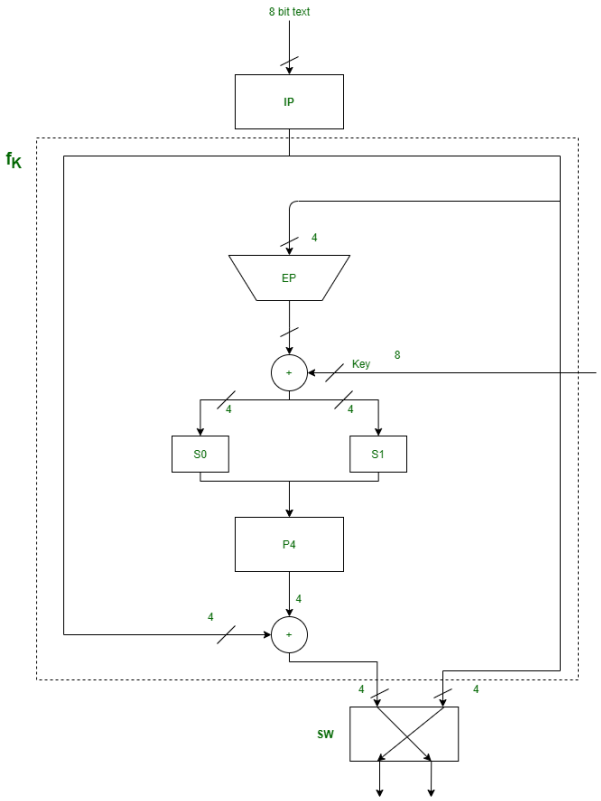


- Further Shifts: The halves are shifted again to produce K2 using another P8 permutation.



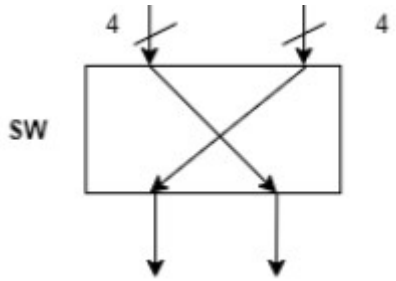
First Round of Feistel Function (fK1):

- The permuted plaintext is split into two 4-bit halves, left (L0) and right (R0).
- The right half (R0) is expanded and permuted using the E/P table, producing an 8-bit block.
- This 8-bit block is XORed with subkey K1.
- The result is then processed through S-boxes (S0 and S1) and a P4 permutation.
- The output of the P4 permutation is XORed with the left half (L0) to produce a new left half (L1).
- The right half (R0) remains unchanged, forming the new right half (R1).



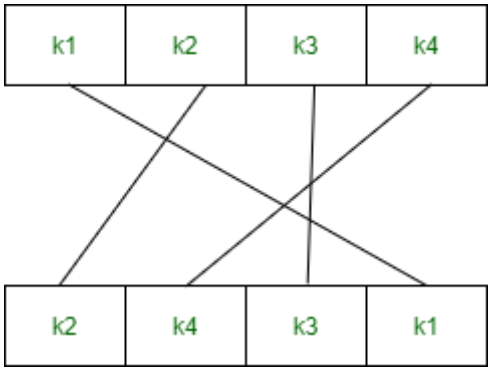
Switching Function (SW):

The left and right halves (L1 and R1) are swapped.



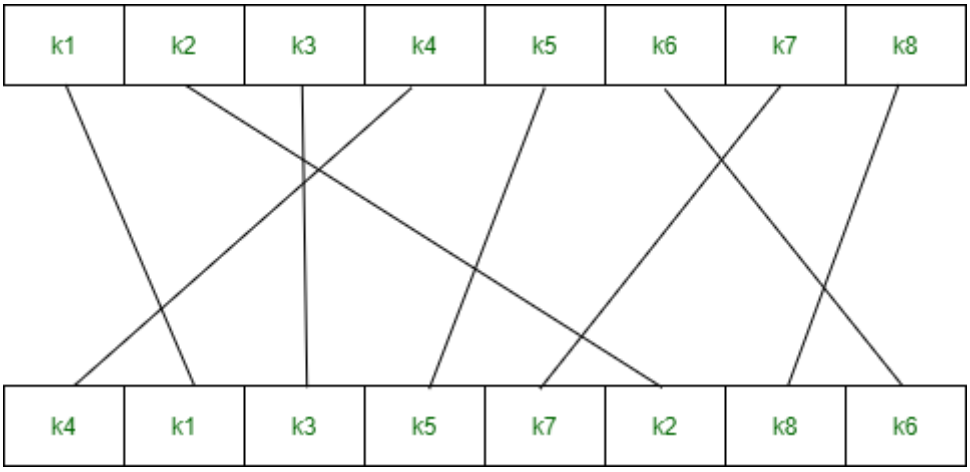
Second Round of Feistel Function (fK2):

- The new right half (R1) is expanded and permuted, then XORed with subkey K2.
- The result passes through S-boxes and a P4 permutation.
- The output of the P4 permutation is XORed with the new left half (R1), producing a new left half (L2).
- The right half (R1) remains unchanged, forming the new right half (R2).



Inverse Initial Permutation (IP^{-1}):

The final left and right halves (L2 and R2) are recombined and subjected to the inverse initial permutation (IP^{-1}) to produce the final 8-bit ciphertext.



Decryption Process

The decryption process of S-DES is essentially the reverse of the encryption process, utilizing the same key schedule but applying the subkeys in reverse order. The steps are as follows:

Initial Permutation (IP):

- The 8-bit ciphertext block undergoes the same initial permutation (IP) as in the encryption process.
- First Round of Feistel Function (fK2):
- The permuted ciphertext is split into two 4-bit halves, left (L2) and right (R2).
- The right half (R2) is expanded and permuted, then XORed with subkey K2.
- The result is processed through S-boxes and a P4 permutation.
- The output of the P4 permutation is XORed with the left half (L2) to produce a new left half (L1).
- The right half (R2) remains unchanged, forming the new right half (R1).

Switching Function (SW):

The left and right halves (L1 and R1) are swapped.

Second Round of Feistel Function (fK1):

- The new right half (R1) is expanded and permuted, then XORed with subkey K1.
- The result passes through S-boxes and a P4 permutation.
- The output of the P4 permutation is XORed with the new left half (R1), producing a new left half (L0).
- The right half (R1) remains unchanged, forming the new right half (R0).

Inverse Initial Permutation (IP^{-1}):

The final left and right halves (L0 and R0) are recombined and subjected to the inverse initial permutation (IP^{-1}) to produce the final 8-bit plaintext.

Chapter 4: System-Level Architecture

Overview

The system-level architecture of the Simplified Data Encryption Standard (S-DES) consists of several core components and functions that work together to achieve encryption and decryption of data. This chapter will explain these components, their interactions, and the role they play in the encryption/decryption process using the provided **C++ implementation** as a reference.

Key Components

1. Initial Permutation (IP) and Inverse Initial Permutation (IP^{-1}):

- Initial Permutation (IP): Rearranges the bits of the input plaintext according to a predefined permutation table to enhance bit diffusion.
- Inverse Initial Permutation (IP^{-1}): Reverses the initial permutation to restore the original bit order in the final ciphertext.

```
void Permutation(int arr[], int index[], int n){
int temp[n];
for (int i = 0; i < n; i++)
temp[i] = arr[index[i] - 1];
for (int i = 0; i < n; i++)
arr[i] = temp[i];
}
```

2. Subkey Generation:

- P10 Permutation: Rearranges the 10-bit key using the P10 permutation table.
- Left Shifts: Splits the key into two 5-bit halves, each of which undergoes left circular shifts.
- P8 Permutation: Recombines the shifted halves and permutes them using the P8 table to generate the subkeys K1 and K2.

```
class KeyGeneration {
    private:
        int P10_rule[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
        int P8_rule[8] = {6, 3, 7, 4, 8, 5, 10, 9};
        int temp_left[5] = {}, temp_right[5] = {};
    public:
        void key(int master_key[], int *k1, int *k2){
            Permutation(master_key, P10_rule, 10);
            Split(master_key, 10, temp_left, temp_right);
            leftRotate(temp_left, 1, 5);
            leftRotate(temp_right, 1, 5);
            combine(temp_left, temp_right, master_key, 10);
            Permutation(master_key, P8_rule, 10);
            for(int i = 0; i < 8; i++) k1[i] = master_key[i];
            leftRotate(temp_left, 2, 5);
            leftRotate(temp_right, 2, 5);
            combine(temp_left, temp_right, master_key, 10);
            Permutation(master_key, P8_rule, 10);
            for(int i = 0; i < 8; i++) k2[i] = master_key[i];
        }
};
```

3. **Feistel Function (fK):**

- Expansion/Permutation (E/P): Expands the right half of the data block from 4 bits to 8 bits.
- XOR with Subkey: XORs the expanded data with the subkey (K1 or K2).
- S-Boxes (S0 and S1): Processes the result through S-Boxes to introduce non-linearity.
- Permutation (P4): Permutes the S-Box output to further mix the bits.
- XOR with Left Half: XORs the permuted result with the left half of the data block.

```
class Roundfunction {
    private:
        int Expanrule[8] = {4, 1, 2, 3, 2, 3, 4, 1};
        int P4_rule[4] = {2, 4, 3, 1};
        int r_arr2[8] = {}, a[4] = {}, b[4] = {};
        int opS0S1[4] = {};
    public:
        void roundfun(int *k1, int *l_arr, int *r_arr, int *fk1){
            ExPermutation(r_arr, Expanrule, r_arr2, 8);
            XOR(k1, r_arr2, 8);
            Split(r_arr2, 8, a, b);
            S_box(a, b, opS0S1);
            Permutation(opS0S1, P4_rule, 4);
            XOR(opS0S1, l_arr, 4);
            combine(l_arr, r_arr, fk1, 8);
        }
};
```

4. Switching Function (SW):

Swaps the left and right halves of the data block after the first round of the Feistel function.

```
void Swap(int *left_array, int *right_array, int n){
    int temp[n];
    for (int i = 0; i < n; i++)
        temp[i] = left_array[i];
    for (int i = 0; i < n; i++)
        left_array[i] = right_array[i];
    for (int i = 0; i < n; i++)
        right_array[i] = temp[i]; }
```

System Operation

1. **Encryption**

Initial Permutation:

The plaintext undergoes the initial permutation.

```
void enc(int arr[], int *key1, int *key2, int *fk1){
    Permutation(arr, IP_rule, 8);
    Split(arr, 8, l_arr, r_arr);
    Roundfunction::roundfun(key1, l_arr, r_arr, fk1);
    Swap(l_arr, r_arr, 4);
    Roundfunction::roundfun(key2, l_arr, r_arr, fk1);
    Permutation(fk1, IP_inv_rule, 8);
}
```

Feistel Rounds:

The data block goes through two rounds of the Feistel function, with the halves swapped in between.

Inverse Permutation:

The result undergoes the inverse initial permutation to produce the ciphertext.

Decryption

Initial Permutation: The ciphertext undergoes the initial permutation.

```
void decryp(int arr[], int *key1, int *key2, int *fk1){
    Permutation(arr, IP_rule, 8);
    Split(arr, 8, l_arr, r_arr);
    Roundfunction::roundfun(key2, l_arr, r_arr, fk1);
    Swap(l_arr, r_arr, 4);
    Roundfunction::roundfun(key1, l_arr, r_arr, fk1);
    Permutation(fk1, IP_inv_rule, 8); }
```

Feistel Rounds: The data block goes through two rounds of the Feistel function, with the subkeys applied in reverse order.

Inverse Permutation: The result undergoes the inverse initial permutation to produce the plaintext.

Chapter 5: Subsystem Design

In this chapter, we delve into the subsystem design of the Simplified Data Encryption Standard (S-DES) implementation. Each subsystem plays a crucial role in the encryption and decryption processes, ensuring data is securely transformed and restored. We'll break down each subsystem and its functions, based on the provided C++ code.

Key Generation Subsystem

The key generation subsystem is responsible for generating two 8-bit subkeys (K1 and K2) from a 10-bit master key. This subsystem includes the following steps:

1. **P10 Permutation:**

A 10-bit input key undergoes permutation using the P10 permutation table, which rearranges the bits.

int P10_rule[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

Code: Permutation(master_key, P10_rule, 10);

2. **Splitting:**

The permuted key is split into two 5-bit halves.

Code: Split(master_key, 10, temp_left, temp_right);

3. **Left Shift:**

Each half undergoes a left circular shift by 1 bit (LS-1).

Code:

leftRotate(temp_left, 1, 5);

leftRotate(temp_right, 1, 5);

4. **P8 Permutation:**

The shifted halves are combined and permuted using the P8 table to generate the first subkey (K1).

Code:

int P8_rule[8] = {6, 3, 7, 4, 8, 5, 10, 9};

combine(temp_left, temp_right, master_key, 10);

Permutation(master_key, P8_rule, 10);

for (int i = 0; i < 8; i++) k1[i] = master_key[i];

5. **Left Shift by 2 (LS-2):**

The halves undergo a left circular shift by 2 bits.

Code:

leftRotate(temp_left, 2, 5);

leftRotate(temp_right, 2, 5);

6. **Second P8 Permutation:**

The shifted halves are combined and permuted using the P8 table to generate the second subkey (K2).

Code:

combine(temp_left, temp_right, master_key, 10);

Permutation(master_key, P8_rule, 10);

for (int i = 0; i < 8; i++) k2[i] = master_key[i];

Round Function Subsystem

The round function is a critical component of the S-DES encryption and decryption processes. It operates on one half of the data block and involves several steps to ensure security through confusion and diffusion.

1. **Expansion/Permutation (E/P):**

The 4-bit right half of the data block is expanded to 8 bits using the E/P table.

Code:

int Expanrule[8] = {4, 1, 2, 3, 2, 3, 4, 1};

ExPermutation(r_arr, Expanrule, r_arr2, 8);

2. **XOR with Subkey:**

The expanded right half is XORed with the subkey (K1 or K2).

Code:

XOR(k1, r_arr2, 8);

3. **Splitting:**

The 8-bit result is split into two 4-bit halves.

Code: Split(r_arr2, 8, a, b);

4. **S-Box Substitution:**

Each 4-bit half is substituted using two S-Boxes (S0 and S1) to introduce non-linearity.

Code:

S_box(a, b, opS0S1);

5. **P4 Permutation:**

The 4-bit output from the S-Boxes is permuted using the P4 table.

Code:

int P4_rule[4] = {2, 4, 3, 1};

Permutation(opS0S1, P4_rule, 4);

6. **XOR with Left Half:**

The permuted result is XORed with the left half of the original data block.

Code: XOR(opS0S1, l_arr, 4);

7. **Combination:**

The result is combined with the unchanged right half to form the new data block for the next round or final permutation.

Code: combine(l_arr, r_arr, fk1, 8);

Encryption Subsystem

The encryption subsystem processes the plaintext through two rounds of the round function, including an initial permutation and a final inverse permutation.

1. **Initial Permutation (IP):**

The 8-bit plaintext undergoes the initial permutation.

Code:

int IP_rule[8] = {2, 6, 3, 1, 4, 8, 5, 7};

Permutation(arr, IP_rule, 8);

2. **First Round Function (fk1):**

The permuted plaintext is processed through the first round function with subkey K1.

Code:

Roundfunction::roundfun(key1, l_arr, r_arr, fk1);

3. **Switching Function (SW):**

The left and right halves of the result are swapped.

Code:

Swap(l_arr, r_arr, 4);

4. **Second Round Function (fk2):**

The swapped result is processed through the second round function with subkey K2.

Code:

Roundfunction::roundfun(key2, l_arr, r_arr, fk1);

5. **Inverse Initial Permutation (IP^{-1}):**

The result undergoes the inverse initial permutation to produce the ciphertext.

Code:

int IP_inv_rule[8] = {4, 1, 3, 5, 7, 2, 8, 6};

Permutation(fk1, IP_inv_rule, 8);

Decryption Subsystem

The decryption subsystem processes the ciphertext through the same components as encryption but with the subkeys applied in reverse order.

1. **Initial Permutation (IP):**

The 8-bit ciphertext undergoes the initial permutation.

Code:

int IP_rule[8] = {2, 6, 3, 1, 4, 8, 5, 7};

Permutation(arr, IP_rule, 8);

2. **First Round Function (fk2):**

The permuted ciphertext is processed through the first round function with subkey K2.

Code:

Roundfunction::roundfun(key2, l_arr, r_arr, fk1);

3. **Switching Function (SW):**

The left and right halves of the result are swapped.

Code:

Swap(l_arr, r_arr, 4);

4. **Second Round Function (fk1):**

The swapped result is processed through the second round function with subkey K1.

Code:

Roundfunction::roundfun(key1, l_arr, r_arr, fk1);

5. **Inverse Initial Permutation (IP^{-1}):**

The final result undergoes the inverse initial permutation to produce the plaintext.

Code:

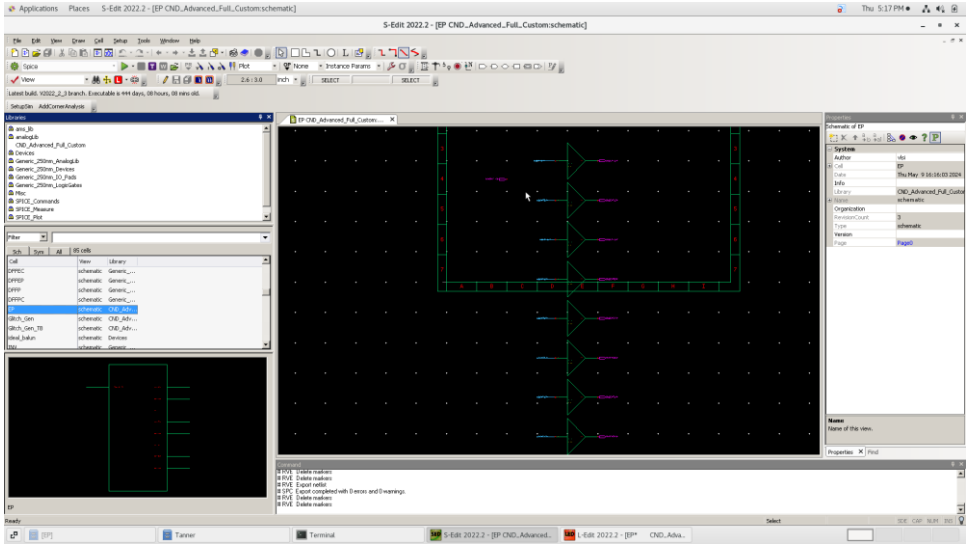
int IP_inv_rule[8] = {4, 1, 3, 5, 7, 2, 8, 6};

Permutation(fk1, IP_inv_rule, 8);

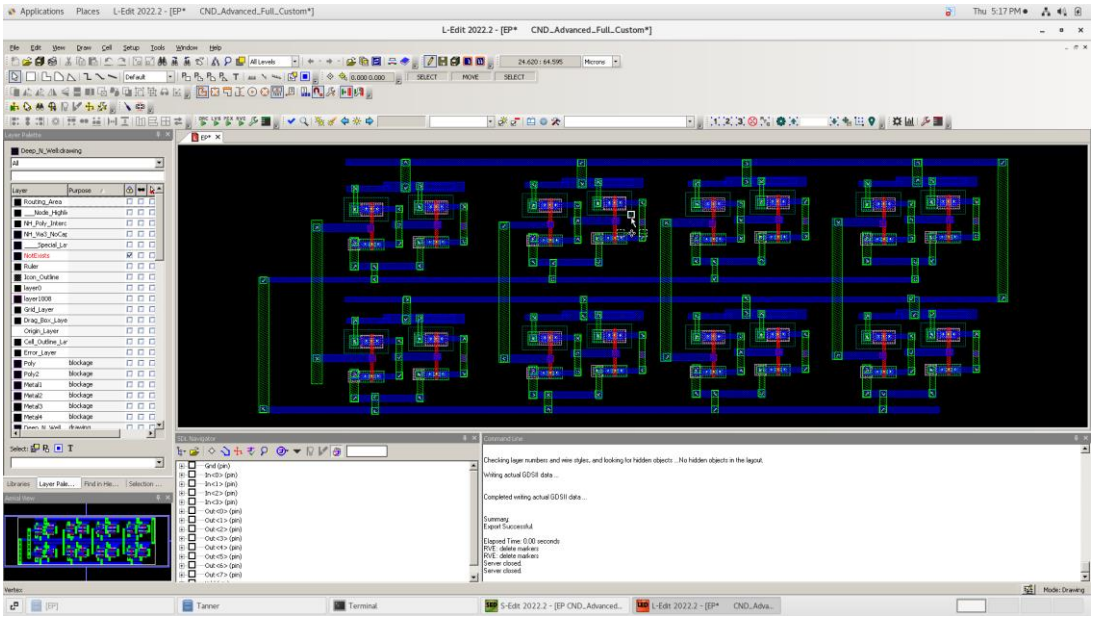
Chapter 6: Schematic and cell designs

In this chapter, we discuss the schematic and cell designs of the Simplified Data Encryption Standard (S-DES) implementation using Tanner EDA tools. Tanner EDA is a popular suite for designing and simulating integrated circuits, and it offers robust tools for creating digital logic designs. This chapter will cover the design flow, key schematic diagrams, and the cell designs that were used to implement the S-DES algorithm.

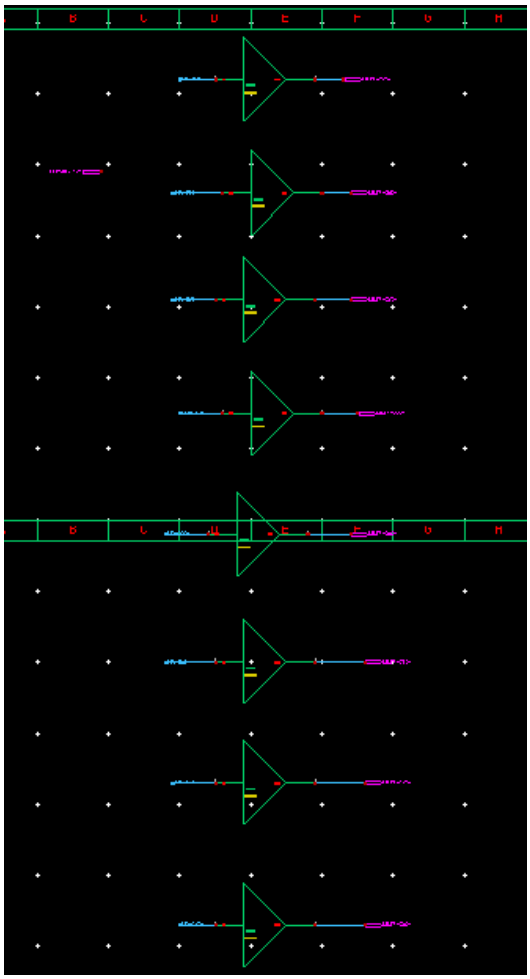
1. EP Schematic



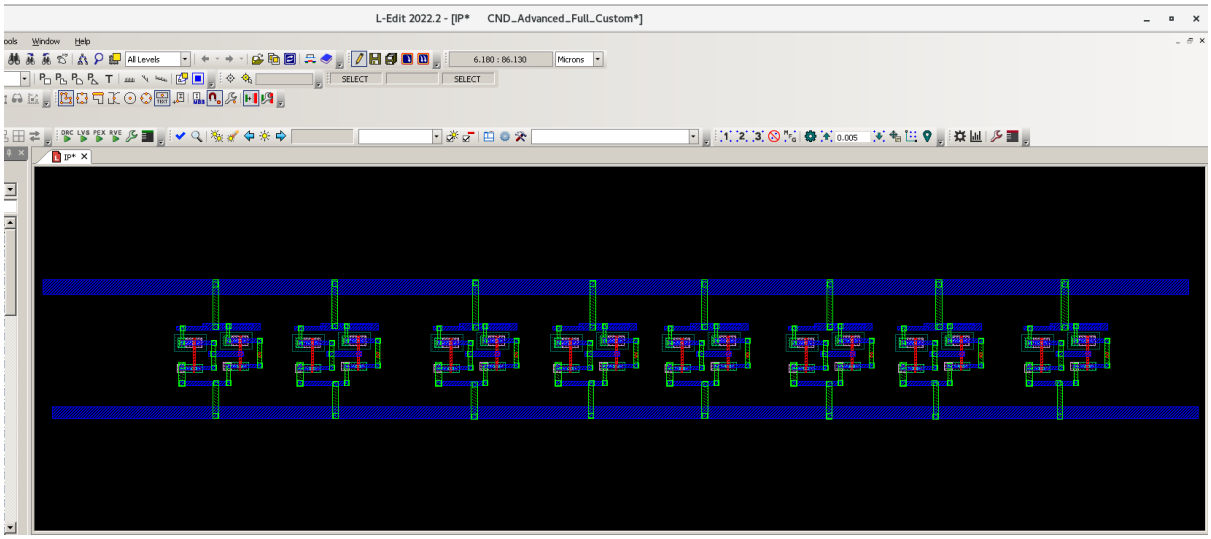
2. EP Layout



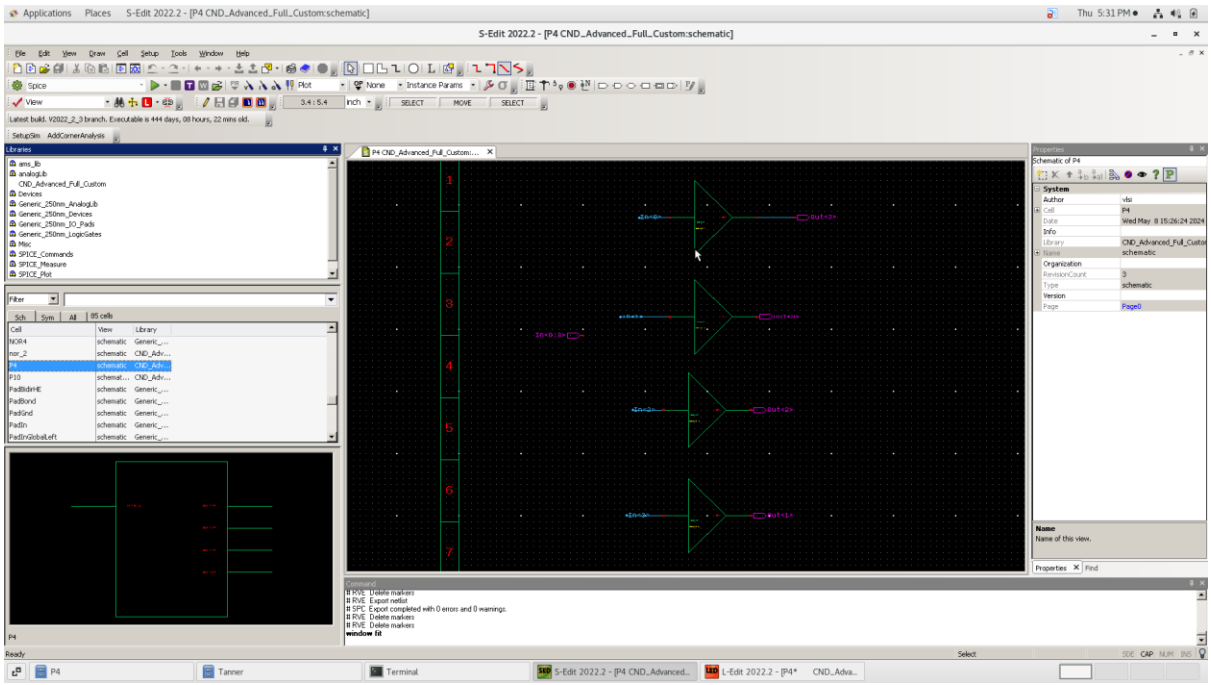
3. IP Schematic



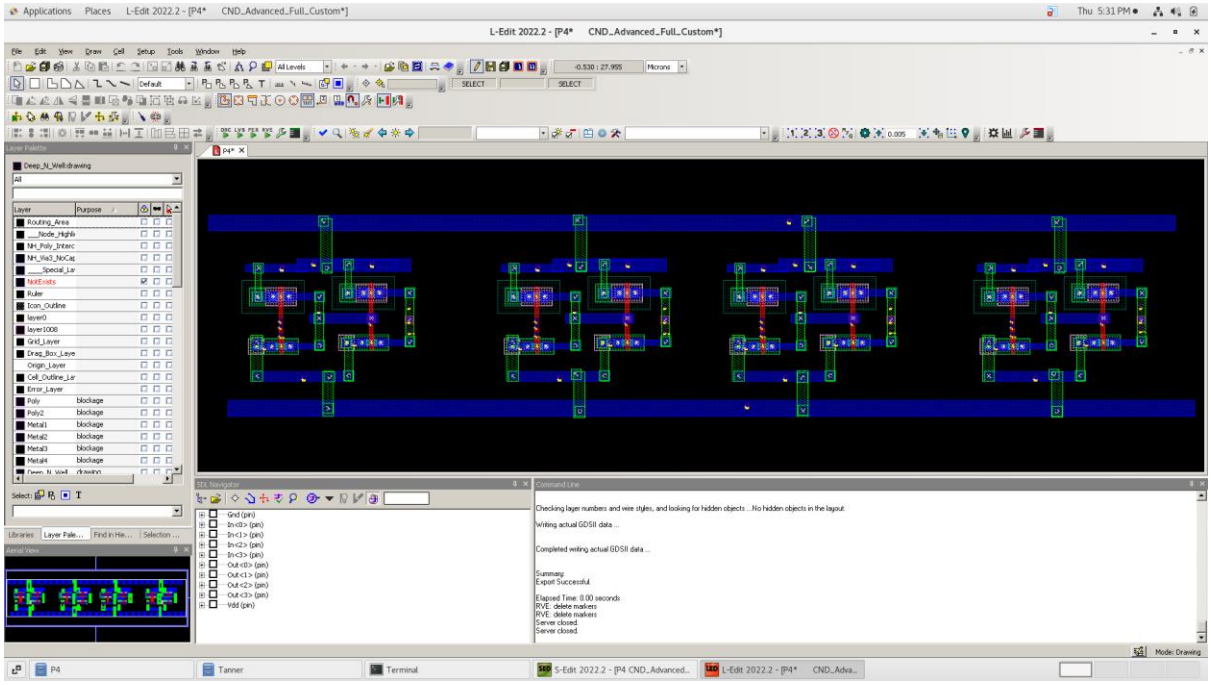
4. IP Layout



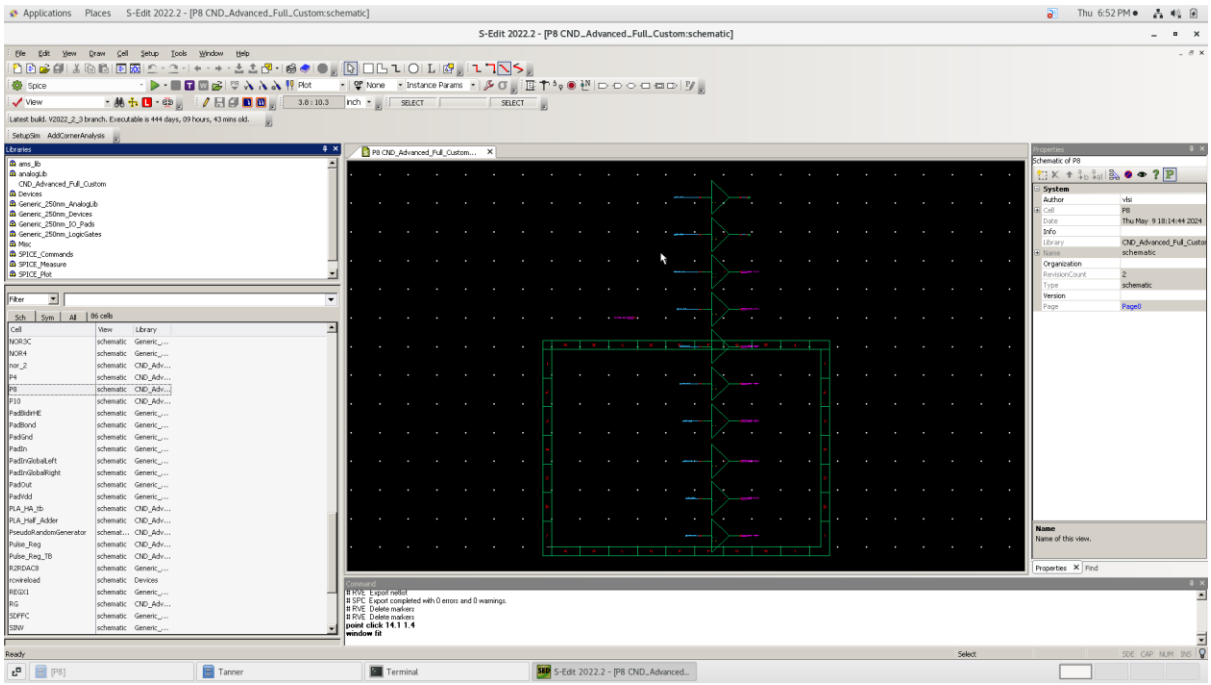
5. P4 Schematic



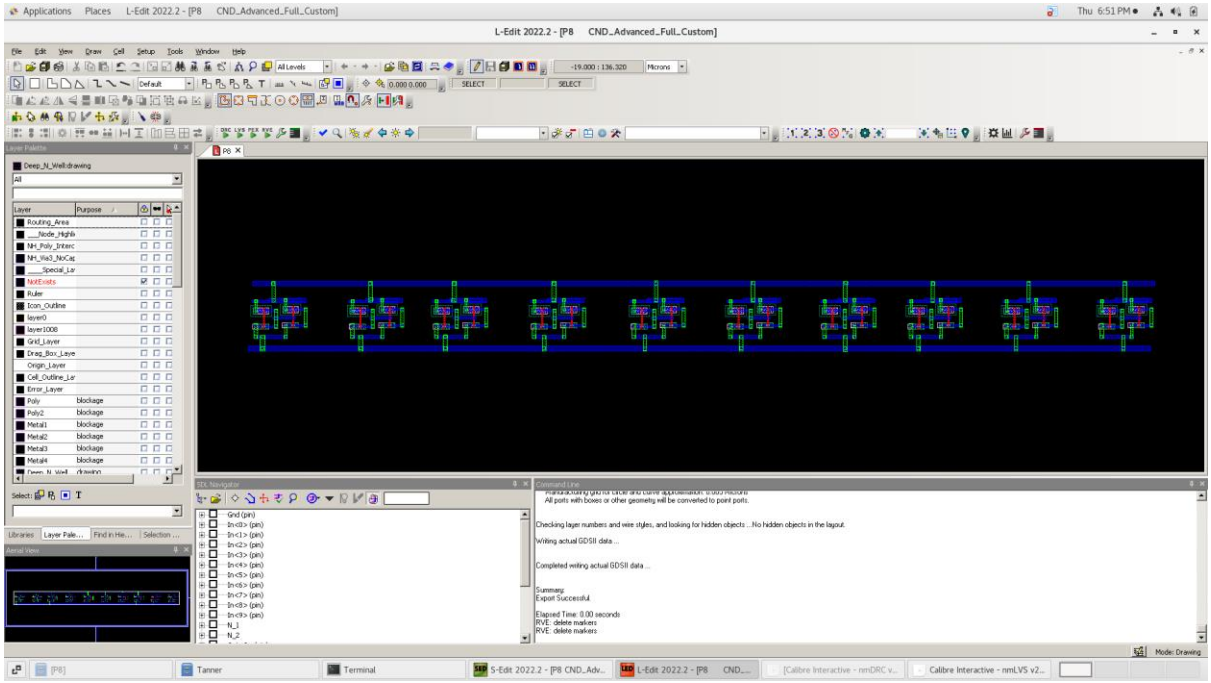
6. P4 Layout



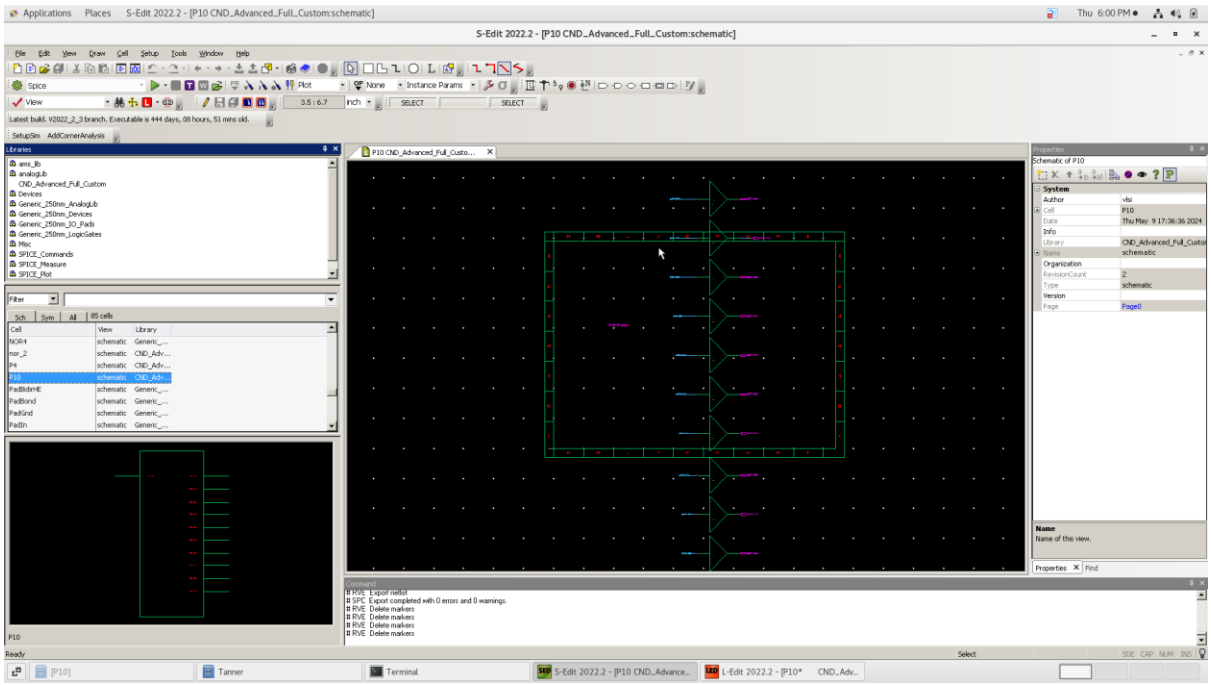
7. P8 Schematic



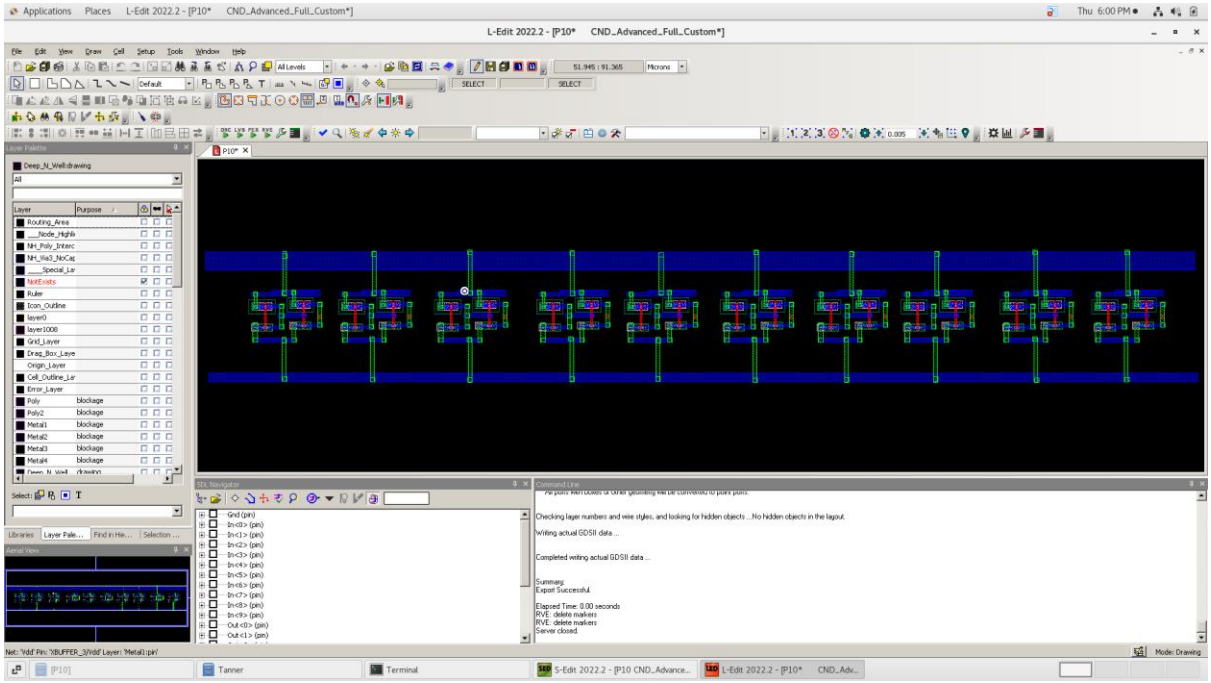
8. P8 Layout



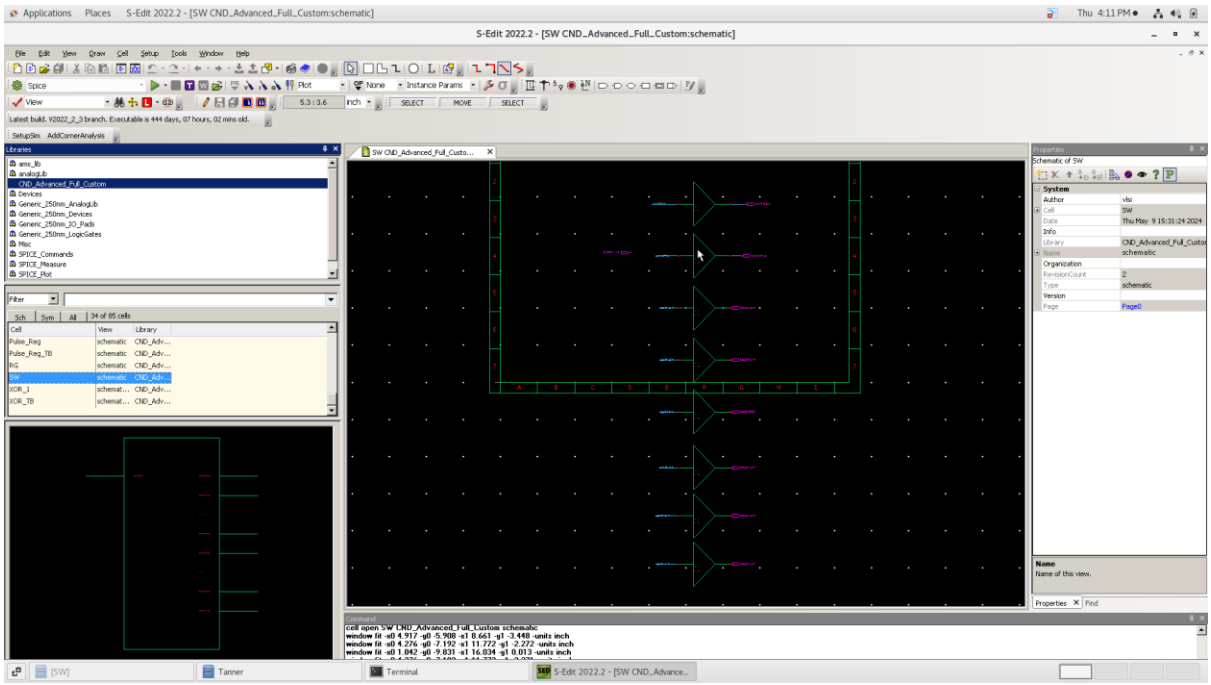
9. P10 Schematic



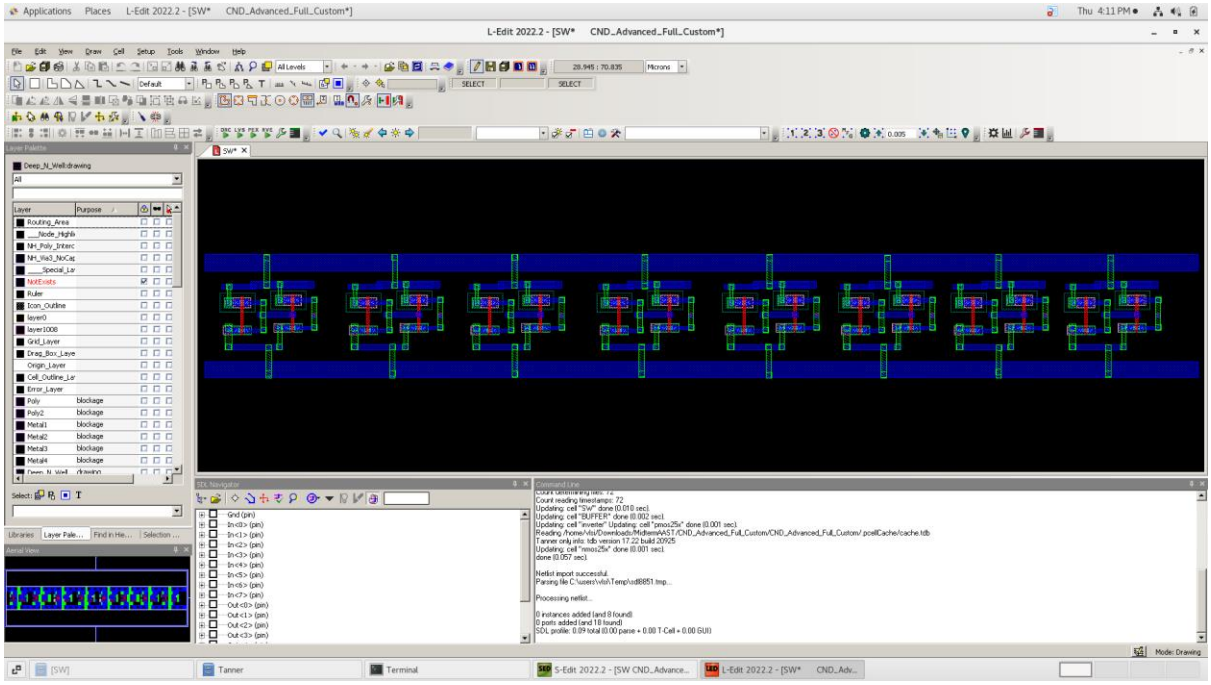
10. P10 Layout



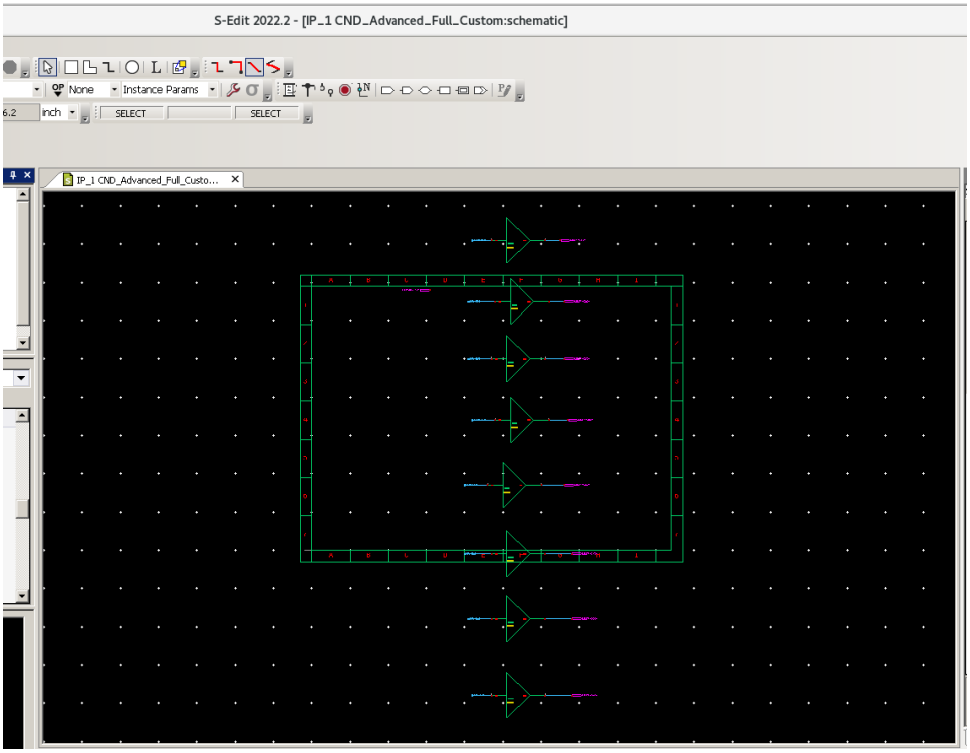
11. SW Schematic



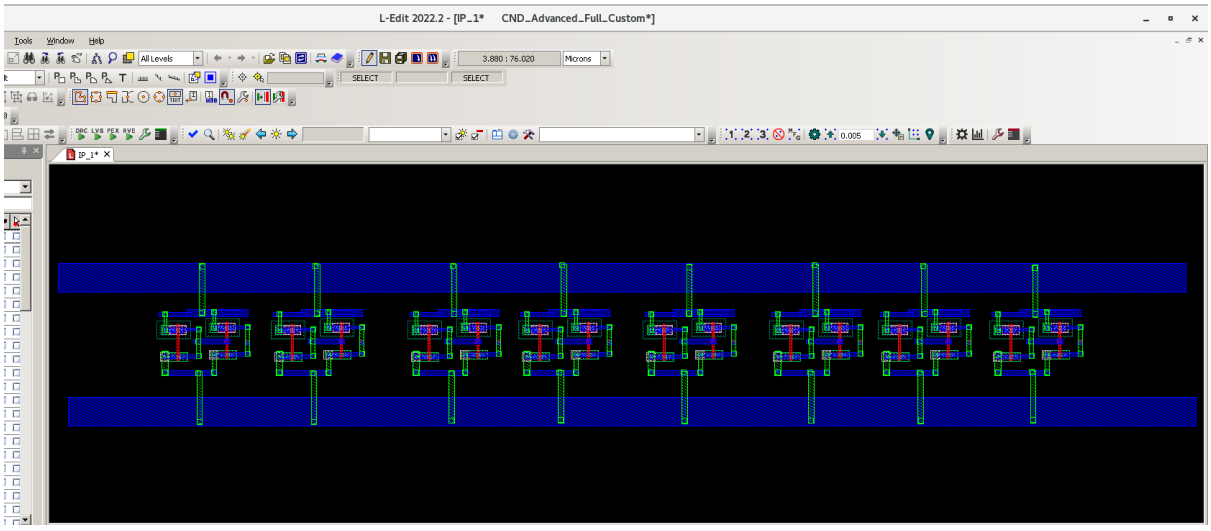
12. SW Layout



13. IP-1 Schematic



14. IP-1 Layout



Chapter 7: Logical & Physical Verification

The primary goal is to validate the correctness of the designed circuit through various verifications, including functional test, Design Rule Check (DRC), and Layout Versus Schematic (LVS) verification. These steps ensure that the circuit functions as intended and meets the design specifications and fabrication constraints.

Functional Verification:

Enter 10-bit Master Key (using space)

1 0 0 1 0 0 0 0 0 1 0

KEY GENERATION..

After P10 Permutation: 1 0 0 0 0 0 1 1 0 0

After split,

l = 1 0 0 0 0

r = 0 1 1 0 0

After LeftShift-1,

l = 0 0 0 0 1

r = 1 1 0 0 0

After P8, Key-1: 1 0 1 0 0 1 0 0

After LeftShift-2,

l = 0 0 1 0 0

r = 0 0 0 1 1

After P8, Key-2: 0 1 0 0 0 0 1 1

Enter e for Encryption | Enter d for Decryption | Enter b for Both

b

Enter 8-bit Plain Text (using space)

1 0 0 1 0 1 1 1

ENCRYPTING..

After IP: 0 1 0 1 1 1 0 1

After split,

l = 0 1 0 1

r = 1 1 0 1

Round Function(fk)-1

After EP: 1 1 1 0 1 0 1 1

XOR with key

0 1 0 0 1 1 1 1

After Split

l = 0 1 0 0

r = 1 1 1 1

After S-Box: 1 1 1 1

After P4

1 1 1 1

XOR with leftarray

1 0 1 0

After combine

1 0 1 0 1 1 0 1

After Swap

l = 1 1 0 1

r = 1 0 1 0

Round Function(fk)-2

After EP: 0 1 0 1 0 1 0 1

XOR with key

0 0 0 1 0 1 1 0

After Split

l = 0 0 0 1

r = 0 1 1 0

After S-Box: 1 1 1 1

After P4

1 1 1 1

XOR with leftarray

0 0 1 0

After combine

0 0 1 0 1 0 1 0

After IP-Inverse, 8-bit Cipher Text is:

0 0 1 1 1 0 0 0

IP

0 0 1 0 1 0 1 0

Split

0 0 1 0

1 0 1 0

Round Function(fk)-1

After EP: 0 1 0 1 0 1 0 1

XOR with key

0 0 0 1 0 1 1 0

After Split

l = 0 0 0 1

r = 0 1 1 0

After S-Box: 1 1 1 1

After P4

1 1 1 1

XOR with leftarray

1 1 0 1

After combine

1 1 0 1 1 0 1 0

swap

1 0 1 0

1 1 0 1

Round Function(fk)-2

After EP: 1 1 1 0 1 0 1 1

XOR with key

0 1 0 0 1 1 1 1

After Split

l = 0 1 0 0

r = 1 1 1 1

After S-Box: 1 1 1 1

After P4

1 1 1 1

XOR with leftarray

0 1 0 1

After combine

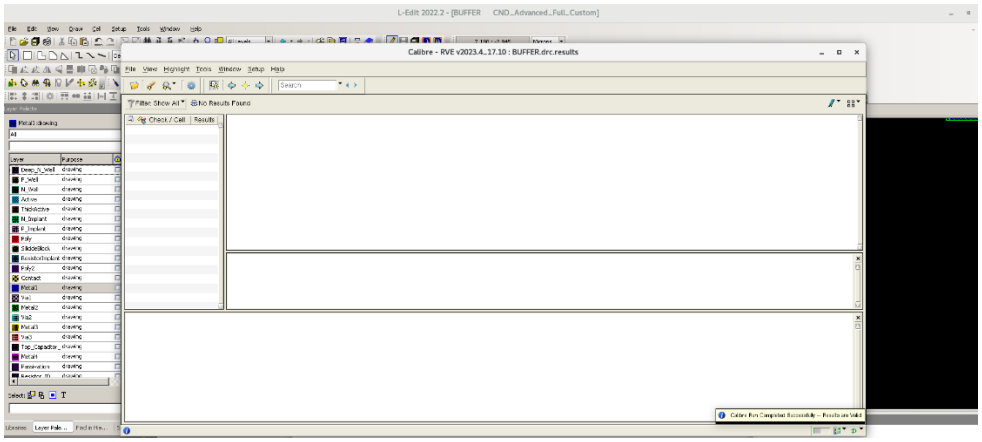
0 1 0 1 1 1 0 1

After IP-Inverse, 8-bit Plain Text is:

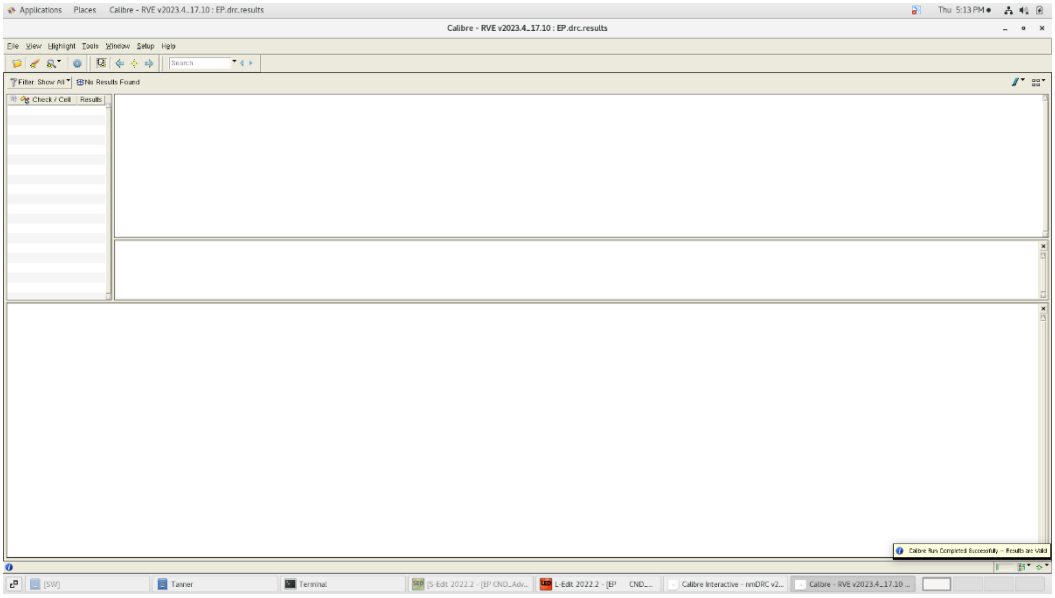
1 0 0 1 0 1 1 1

DRC:

1. Buffer



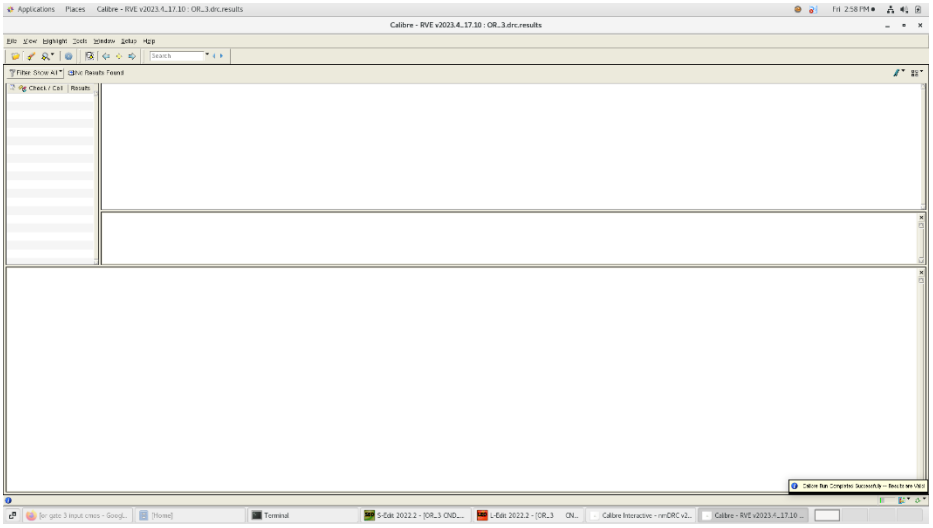
2. EP



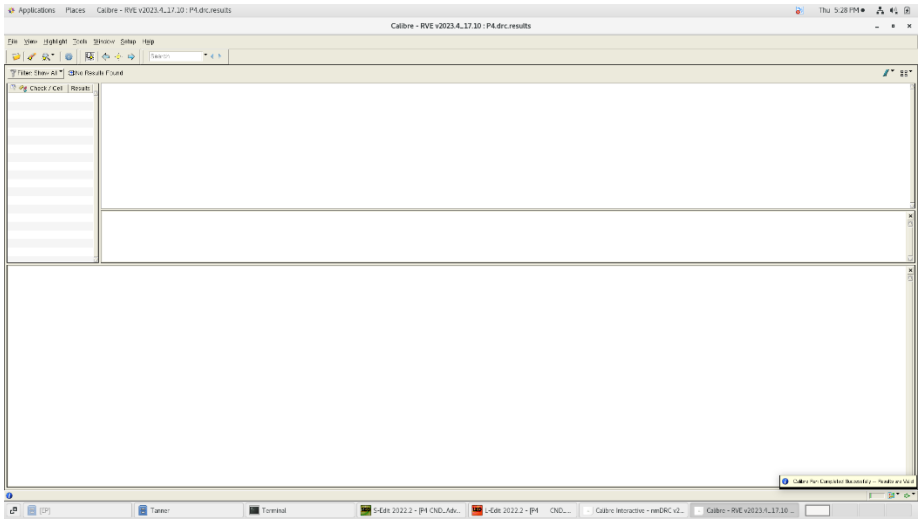
3. IP

4. **IP-1**

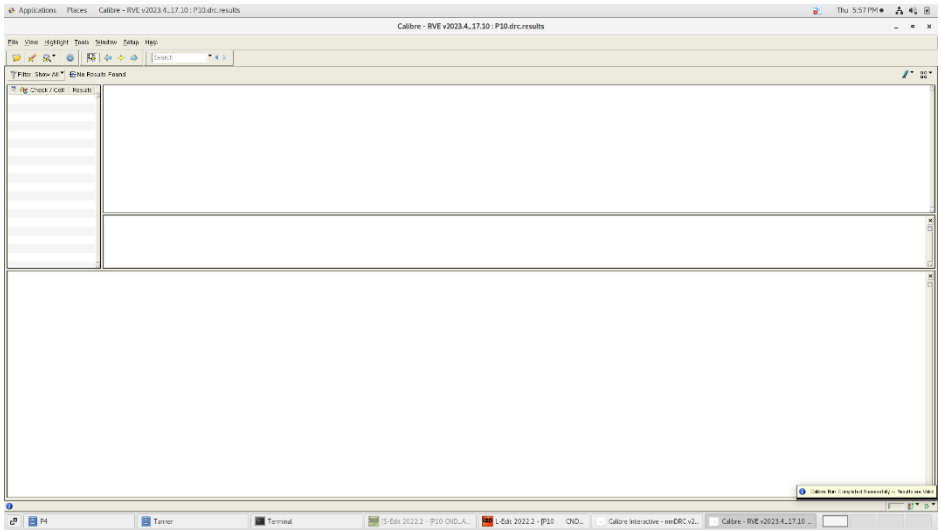
5. **OR**



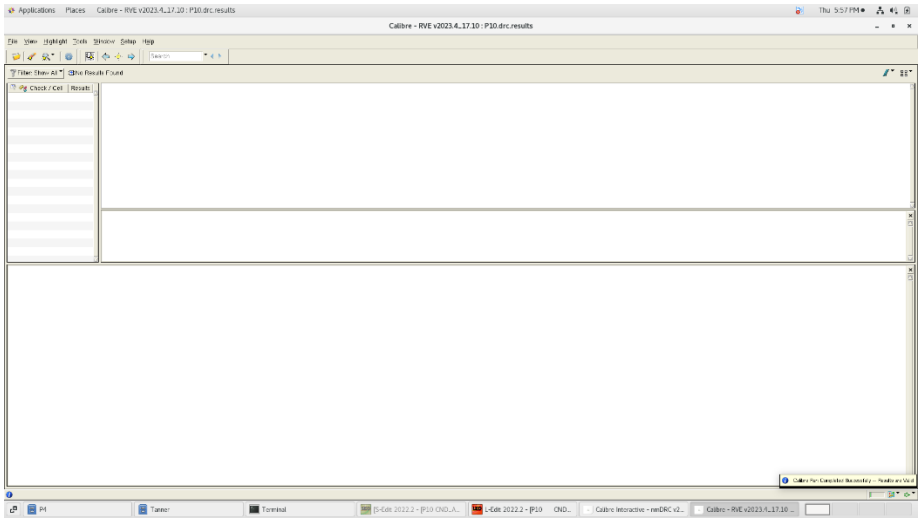
6. **P4**



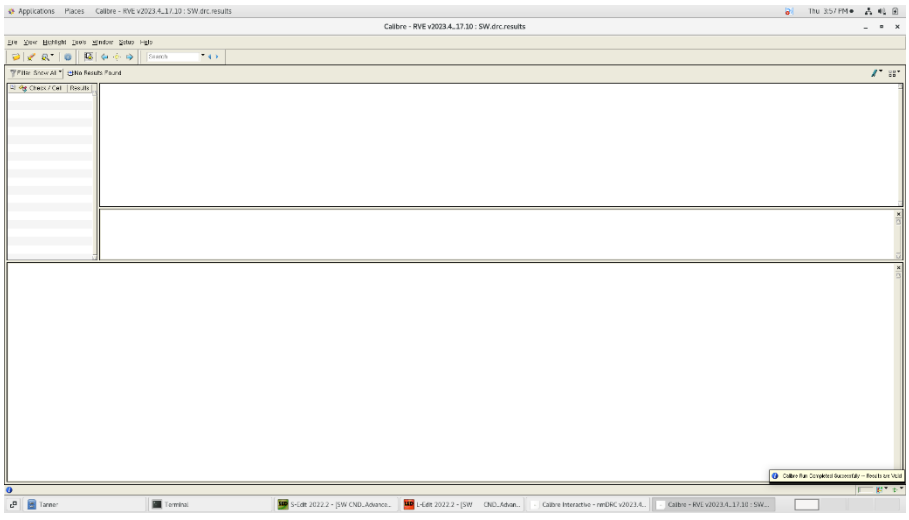
7. P8



8. P10

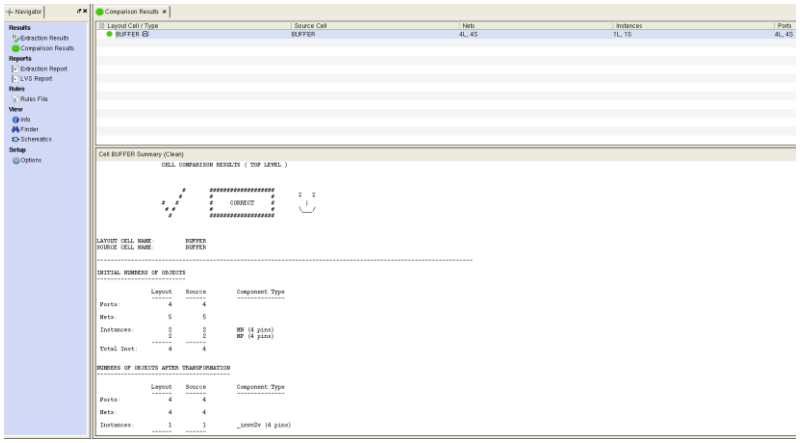


9. SW

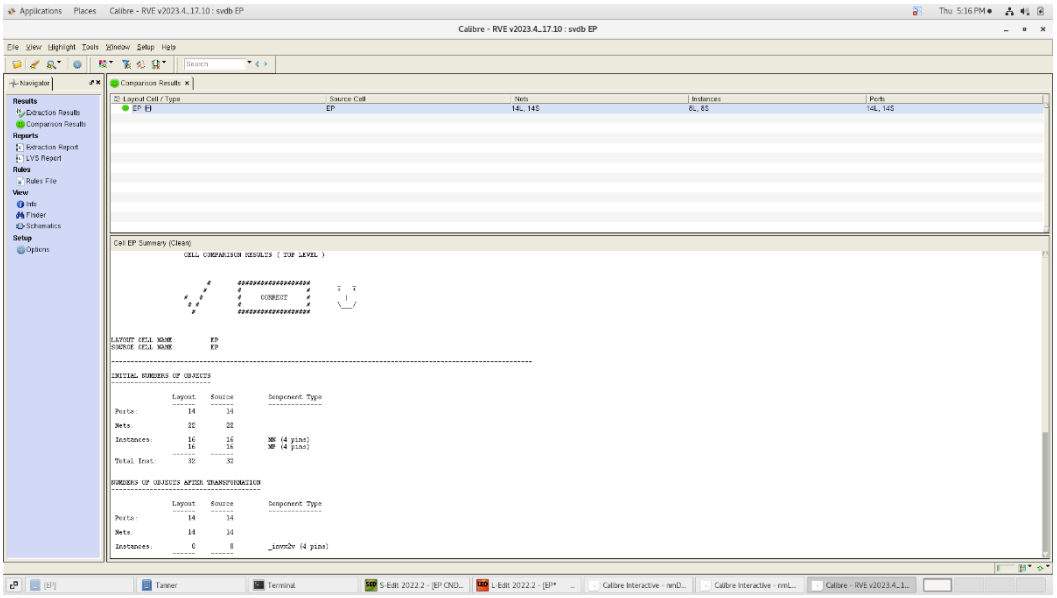


LVS:

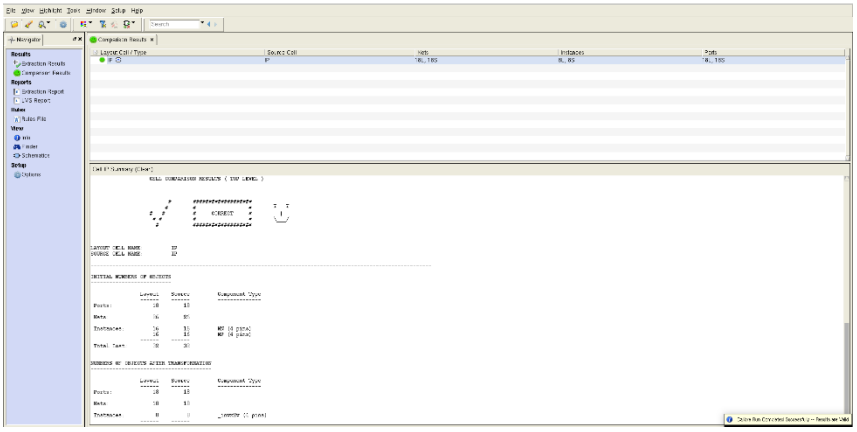
1. Buffer



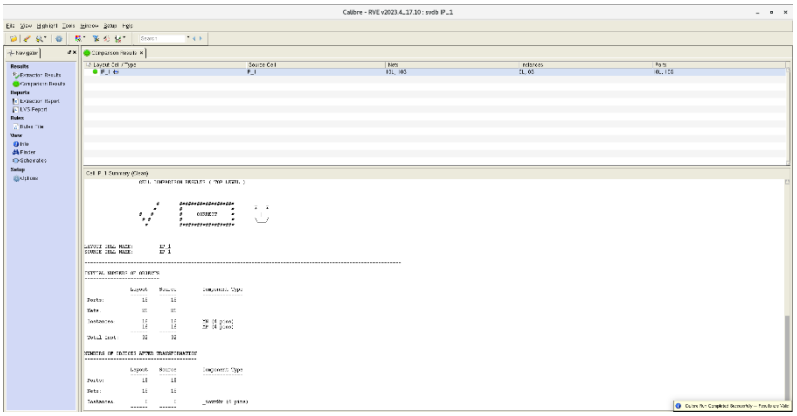
2. EP



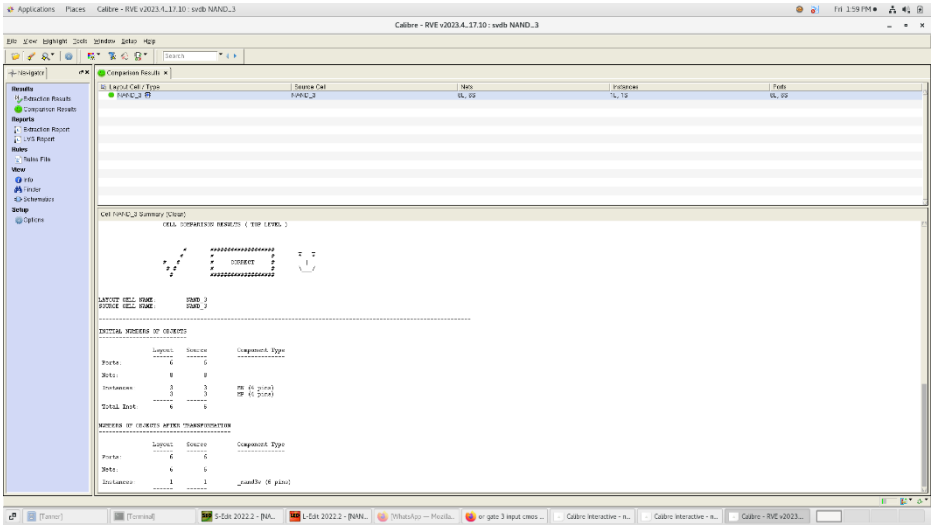
3. IP



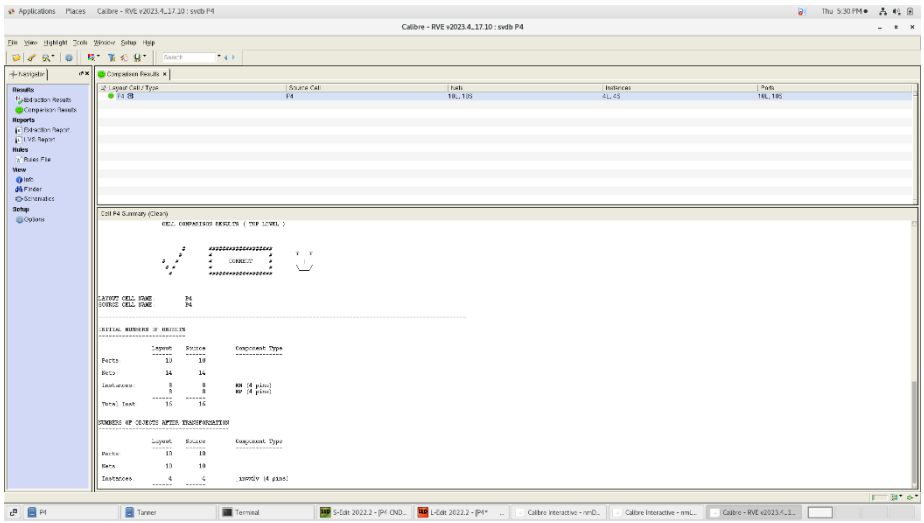
4. **IP-1**



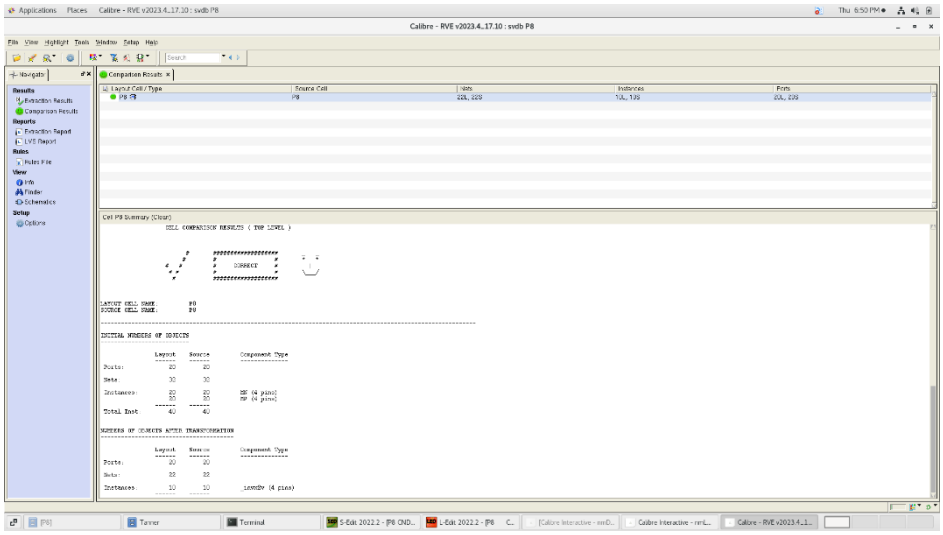
5. **OR**



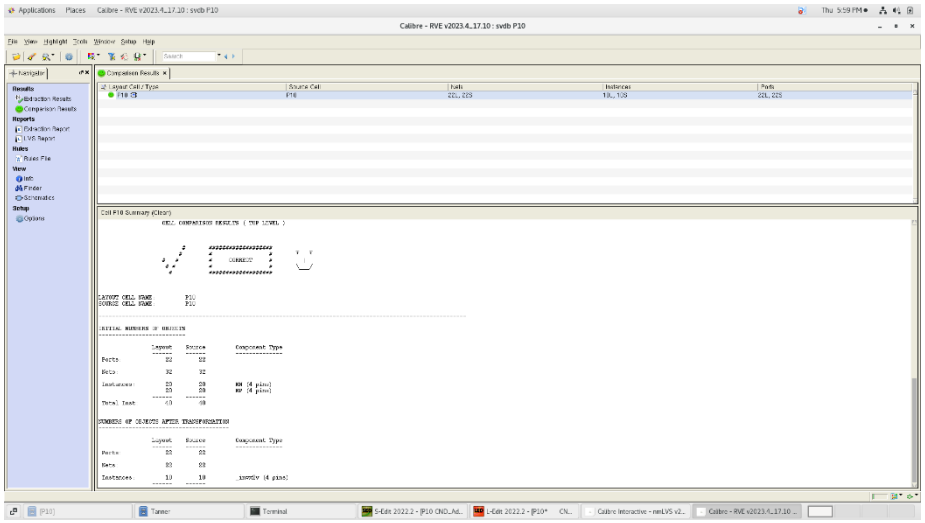
6. **P4**



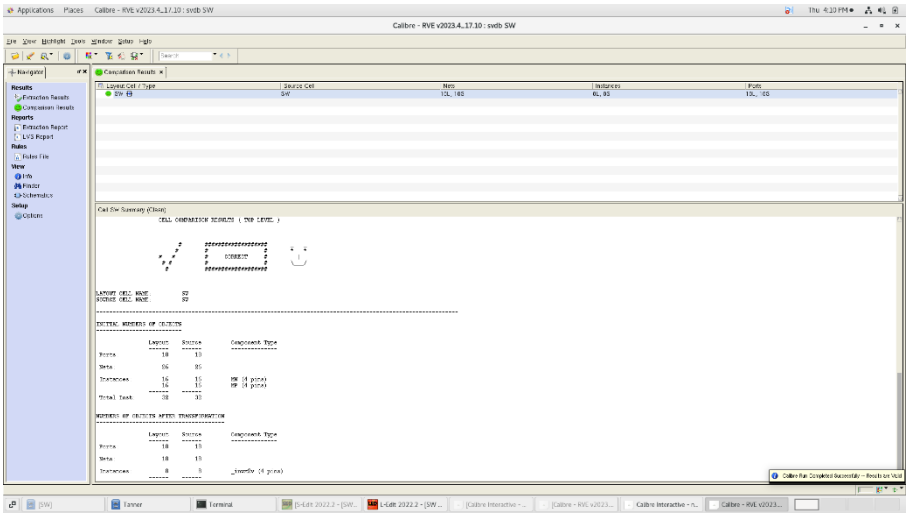
7. P8



8. P10



9. SW



Chapter 8: Lessons Learned, Problems Faced, and Unexpected Findings

Lessons Learned

Understanding DES and S-DES

One of the most significant lessons learned from this project is the deep understanding of the Data Encryption Standard (DES) and its simplified version (S-DES). Through the implementation process, it became clear how the various components of the algorithm work together to achieve encryption and decryption. This project provided practical insights into:

1. Feistel Structure: The iterative nature of the Feistel network and how it allows for both encryption and decryption using similar structures.
2. Key Scheduling: The importance of subkey generation and its impact on the security of the encrypted data.
3. Permutation and Substitution: How these operations increase the complexity and security of the encryption process by diffusing and confusing the plaintext data.

Practical Skills in EDA Tools

Working with Tanner EDA tools for the schematic design, simulation, and layout development has been invaluable. Key skills developed include:

1. Schematic Design in S-Edit: Learning to create and connect various components to form the complete S-DES encryption and decryption circuits.
2. Simulation in T-Spice: Setting up and running simulations to verify the functional correctness of the design.
3. Layout Design in L-Edit: Translating the schematic into a physical layout and ensuring it meets design rules and matches the schematic.

Problems Faced

Initial Design Challenges

During the initial design phase, understanding the intricacies of the S-DES algorithm posed some challenges. Mapping theoretical concepts into practical design required iterative learning and adjustments. Specifically, the permutation and substitution steps needed careful attention to ensure they were implemented correctly.

Key Generation Complexity

Generating the subkeys (K1 and K2) from the 10-bit master key involved multiple steps of permutation and left shifts. Ensuring the accuracy of these operations was crucial, and minor errors in the permutation indices or shift operations led to incorrect subkey generation, affecting the entire encryption/decryption process.

Synchronization Issues in Simulation

During the simulation phase, ensuring that all components were correctly synchronized was a challenge. Misalignments in timing or logic led to incorrect results, requiring thorough debugging. For instance, ensuring that the split, permutation, and XOR operations happened in the correct sequence was critical.

Unexpected Findings

1. Impact of Permutation Order

An unexpected finding was the significant impact of the order of permutations on the encryption strength. Even slight deviations in the permutation indices affected the diffusion properties of the encryption process, leading to weaker encryption. This highlighted the precision needed in cryptographic algorithm design.

2. Sensitivity to Input Variations

The S-DES algorithm's sensitivity to variations in input plaintext and keys was another unexpected observation. Small changes in input values resulted in substantial differences in the output ciphertext, demonstrating the algorithm's effectiveness in ensuring data security through confusion and diffusion principles.

3. Importance of DRC and LVS

The importance of Design Rule Check (DRC) and Layout Versus Schematic (LVS) verification was underscored during this project. These steps not only ensured that the design met fabrication requirements but also helped identify and rectify logical and connectivity errors early in the design process. Issues that were not apparent in the schematic were caught during these checks, preventing potential fabrication problems.

Conclusion

This project provided a comprehensive learning experience, from understanding the theoretical foundations of S-DES to practically implementing and verifying the design using EDA tools. The challenges faced and the unexpected findings highlighted the complexities and intricacies involved in cryptographic design and reinforced the importance of thorough verification at each stage.

Conclusion

In conclusion, the implementation of the Simplified Data Encryption Standard (S-DES) project has provided a comprehensive learning journey, encompassing theoretical understanding, practical design, and verification using Electronic Design Automation (EDA) tools. Through this project, a deep comprehension of cryptographic principles, particularly the Feistel structure, key scheduling, and permutation-substitution operations, was attained.

Challenges encountered during the design process, such as key generation complexity and synchronization issues in simulation, underscored the importance of precision and thoroughness in cryptographic algorithm implementation. Unexpected findings, including the sensitivity to input variations and the impact of permutation order, provided valuable insights into algorithm robustness and security.

Moreover, the significance of Design Rule Check (DRC) and Layout Versus Schematic (LVS) verification in ensuring design correctness and adherence to fabrication constraints was highlighted. Overall, this project not only enhanced practical skills in EDA tools but also deepened understanding of cryptographic concepts, paving the way for future endeavors in hardware-based security systems.