FPGA-BASED SYSTEM DESIGN LAB — MINI PROJECT REPORT

Lakshit Verma (230959210) **Priyansh Sarkar** (230959200) **Ayush Gupta** (230959206)

April 2025

INTRODUCTION

1. Background Information

The Data Encryption Standard (DES) is a symmetric-key algorithm widely used for data security. It operates on 64-bit data blocks using a 56-bit key and follows 16 rounds of Feistel network-based encryption. With the rise of FPGA technology, hardware implementations of cryptographic algorithms like DES offer speed and efficiency advantages over software-based encryption.

2. Objectives and Scope

Objectives:

- Implement DES encryption using Verilog on an FPGA.
- Optimize the design for speed and resource efficiency.
- Ensure correctness by comparing results with known DES outputs.

Scope:

- Only encryption is considered; decryption is not implemented.
- The design includes key scheduling and Feistel function implementation.
- The project is tested on software only; hardware testing on an actual FPGA board is not included.

3. Overview of the Verilog Design

The design follows the standard DES encryption process:

- **Initial Permutation (IP)**: Rearranges the 64-bit plaintext input.
- **16 Feistel Rounds**:
  - Expands the right half from 32 to 48 bits (Expansion Permutation E).
  - XORs the expanded right half with a round key.
  - Uses S-Boxes to substitute 6-bit values into 4-bit outputs.
  - Applies a final permutation (P-Box) on the substituted bits.
  - Swaps halves and repeats for 16 rounds.
- **Final Permutation (IP^{-1})**: Restores the order to produce the ciphertext.

Why Verilog?

- **Parallel processing capability** of FPGA, improving encryption speed.
- **Hardware-level security** since keys are not exposed in software memory.
- **Customizability**, allowing for optimized DES implementations.

4. Problem Definition & Motivation

- **Software-based DES implementations** are slower compared to hardware due to sequential processing.
- **FPGAs offer parallel execution**, making them ideal for cryptographic tasks.
- **Applications include** secure communication, embedded security, and fast encryption in resource-constrained environments.

2. SYSTEM DESIGN AND ARCHITECTURE

High-level design overview (block diagram of the system)

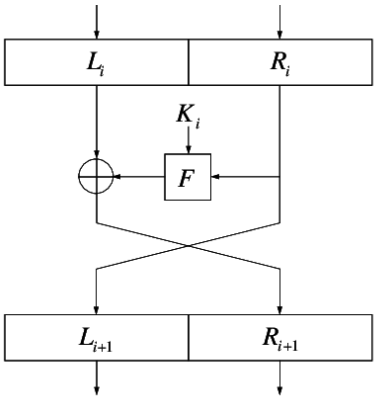

Figure 1: A single module of the Feistel Network

Description of the Overall System Architecture

The system is a hardware implementation of a DES-like encryption algorithm on an FPGA. It consists of key scheduling, encryption, and control modules, ensuring efficient data flow and processing. The architecture is optimized for parallelism and pipelining to achieve high throughput.

The system receives plaintext and a secret key as inputs. The key scheduling module derives subkeys, which the encryption module uses to perform multiple rounds of transformation. The control unit orchestrates data movement, synchronization, and encryption operations.

Key Modules and Their Interconnections

ProcessKey Module

1. **Function:**

The `ProcessKey` module is responsible for key scheduling in the encryption process. It takes a 64-bit input key and generates 16 subkeys, each 48 bits long, which are used in different rounds of encryption.

2. **Inputs and Outputs:**

- **Input:**
  - `key` (64-bit): The original encryption key.
- **Outputs:**
  - `key1` to `key16` (each 48-bit): The 16 round keys used in encryption.

3. **Subcomponents and Functions:**

4. `PC1_perm` **Function:**
    - Performs the Permuted Choice 1 (PC1) operation, reducing the 64-bit key to 56 bits.
    - The permutation table `PC1` rearranges and removes 8 parity bits.
5. `PC2_perm` **Function:**
    - Performs the Permuted Choice 2 (PC2) operation to generate 48-bit subkeys from 56-bit values.
    - This function is called for each round key.
6. `C_i_D_i` **Function:**
    - Implements the left shifts applied to the key halves (`C` and `D`) for each round.
    - Uses an array `shift_left` to determine the number of shifts per round.
7. **Key Schedule Execution (`always @(key)`)**
    - Calls `PC1_perm` to generate `C0` and `D0` from the original key.
    - Iterates through 16 rounds, applying `C_i_D_i` and `PC2_perm` to generate `key1` to `key16`.

Encrypt Module

1. **Function:**

The `Encrypt` module is responsible for performing the main encryption process using the Feistel structure. It applies the DES encryption algorithm to a 64-bit message using a 64-bit key.

2. **Inputs and Outputs:**

- **Inputs:**
    - `message` (64-bit): The plaintext input message.
    - `key` (64-bit): The encryption key.
- **Output:**
    - `ciphertext` (64-bit): The encrypted output.

Subcomponents and Functions:

1. `perm_IP` **Function:**
    - Applies the Initial Permutation (IP) on the message.
2. `perm_IP_inverse` **Function:**
    - Applies the Final Permutation (IP_{-1}) to reverse the initial permutation at the end of encryption.
3. `perm_E` **Function:**
    - Expands the 32-bit right half of data to 48 bits for key mixing.
4. `perm_P` **Function:**
    - Applies the P-box permutation on the output of S-boxes.
5. `SBOX` **Function:**
    - Implements the 8 substitution boxes (S1 to S8).
    - Takes a 6-bit input and returns a 4-bit output.
6. `f` **Function:**
    - Implements the Feistel function used in each round.
    - Expands the right half of data using `perm_E`, XORs with the round key, applies `SBOX` substitution, and then `perm_P`.
7. **Encryption Process (`always @(message, key)`)**
    - Calls `ProcessKey` to generate subkeys.
    - Splits the input message into left (`L`) and right (`R`) halves.
    - Performs 16 rounds of the Feistel network, applying `f` function and XOR operations.

- Swaps `L` and `R` before applying `perm_IP_inverse` to get the final ciphertext.

Interconnection Between Modules

1. `ProcessKey` → `Encrypt`
   - The `ProcessKey` module generates 16 subkeys (`key1` to `key16`).
   - These subkeys are used in the `Encrypt` module for each round of encryption.
2. `Encrypt` → **Output (`ciphertext`)**
   - The `Encrypt` module processes the input message and outputs the final ciphertext.

Explanation of Design Choices Made for the Architecture

1. **Pipeline Optimization:** The design leverages pipelining to improve throughput, allowing multiple encryption operations to be processed simultaneously.
2. **Parallel Processing:** Parallel execution of subkey generation and encryption rounds speeds up computation.
3. **Resource Utilization:** The design minimizes logic and memory footprint to fit within FPGA constraints.

3. RESULTS

Final achieved simulation results (e.g., waveforms, output data)

4. CONCLUSION

This project successfully implemented the Data Encryption Standard (DES) algorithm using Verilog, achieving encryption through a structured Feistel network. The design incorporates key expansion, generating 16 subkeys from a 64-bit key using proper permutations (PC1, PC2) and shifting operations. The encryption process follows 16 iterative rounds, where each round applies expansion (E), S-Box substitution, and P-Box permutation to ensure diffusion and confusion. Initial and Final Permutations (IP and IP^{-1}) further enhance security, making the encryption robust.

By structuring the implementation into modular components (`ProcessKey` and `Encrypt`), the design remains efficient, reusable, and adaptable for hardware-based cryptographic applications such as FPGA and ASIC systems.

Beyond encryption, this project gave us an in-depth understanding of both block cipher principles, including permutation, substitution, key-dependent transformations and a greater understanding of Verilog in general. Additionally, this project can serve as a foundation for further cryptographic advancements, such as extending DES to 3DES or transitioning to AES for enhanced security.