Implementation of RSA Algorithm on FPGA (Nexys-4 DDR)

By

Harshvardhan Soni (92200133028)

Under the guidance of

Prof. Chandrasinh Parmar
Prof. Mitesh Solanki

A Project Submitted to

Marwadi University in Partial Fulfillment of the Requirements for the subject

Capstone Project (01CT0715)

September 2025

MARWADI UNIVERSITY
Rajkot-Morbi Road, At & Po. Gauridad,
Rajkot-360003, Gujarat, India

Abstract

This report presents a detailed analysis of the implementation of the RSA (Rivest-Shamir-Adleman) cryptographic algorithm on a Field-Programmable Gate Array (FPGA) platform, specifically the Nexys-4 DDR development board. The project demonstrates the feasibility and advantages of hardware-based cryptographic implementations for secure data encryption and decryption operations.

Table of Contents

1. Introduction

1.1 Background

The RSA algorithm, developed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977, is one of the most widely used public-key cryptographic systems. It relies on the mathematical difficulty of factoring large prime numbers, making it computationally secure for protecting sensitive information in digital communications.

1.2 Project Motivation

Traditional software-based RSA implementations, while functional, often suffer from performance limitations and security vulnerabilities. Hardware implementations on FPGA platforms offer several advantages:

- Enhanced Security: Hardware-based implementations are more resistant to side-channel attacks
- Improved Performance: Parallel processing capabilities of FPGAs enable faster encryption/decryption
- Power Efficiency: Optimized hardware designs consume less power compared to general-purpose processors
- Customizability: FPGA implementations can be tailored for specific application requirements

1.3 Objectives

The primary objectives of this project include:

- Design and implement the RSA algorithm on Nexys-4 DDR FPGA board
- Demonstrate encryption and decryption capabilities with practical examples
- Analyze performance characteristics and resource utilization
- Validate the implementation through comprehensive testing

2. RSA Algorithm Theory

2.1 Mathematical Foundation

The RSA algorithm is based on modular arithmetic and the difficulty of prime factorization. The algorithm involves the following mathematical operations:

Key Generation Process:

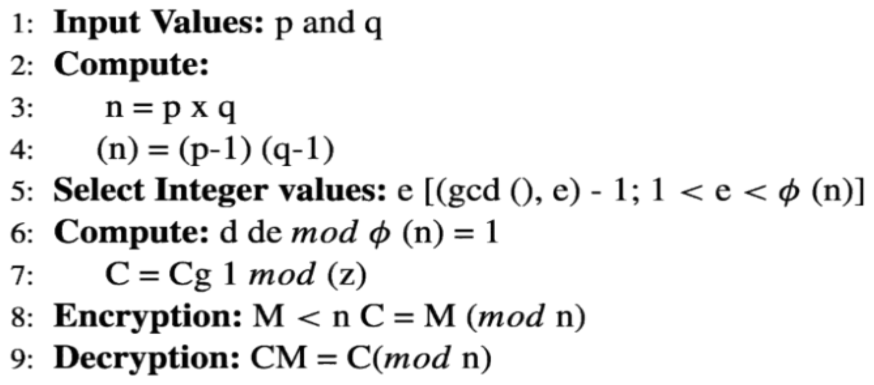
1. **Prime Selection**: Choose two distinct prime numbers, p and q
2. **Modulus Calculation**: Compute $n = p \times q$
3. **Euler's Totient Function**: Calculate $\varphi(n) = (p-1)(q-1)$
4. **Public Exponent Selection**: Choose e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$
5. **Private Exponent Calculation**: Compute d such that $(e \times d) \bmod \varphi(n) = 1$

Encryption Process:

- Ciphertext C = M^e mod n
- Where M is the plaintext message

Decryption Process:

- Plaintext M = C^d mod n
- Where C is the ciphertext



```
Input Values: p and q
Compute:
    n = p x q
    (n) = (p-1) (q-1)
Select Integer values: e [(gcd (), e) - 1; 1 < e < $\phi$ (n)]
Compute: d de $mod$ $\phi$ (n) = 1
    C = Cg 1 $mod$ (z)
Encryption: M < n C = M ($mod$ n)
Decryption: CM = C($mod$ n)
```

2.2 Security Considerations

The security of RSA depends on:

- Key Size: Larger keys provide better security but require more computational resources
- Prime Quality: Use of cryptographically strong primes
- Random Number Generation: Secure generation of prime numbers and keys
- Implementation Security: Protection against side-channel attacks

3. FPGA Platform Analysis

3.1 Nexys-4 DDR Specifications

The Nexys-4 DDR development board provides an excellent platform for cryptographic implementations:

Key Features:

- **FPGA Chip**: Xilinx Artix-7 XC7A100T-1CSG324C
- **Logic Elements**: 101,440 logic cells
- **Memory Resources**:
  - 4,860 Kb of Block RAM
  - 128 MB DDR2 SDRAM
  - 16 MB Quad-SPI Flash
- **I/O Capabilities**:
  - USB-UART bridge for communication
  - 16 user switches
  - 16 user LEDs
  - 7-segment displays
  - Push buttons for user interaction

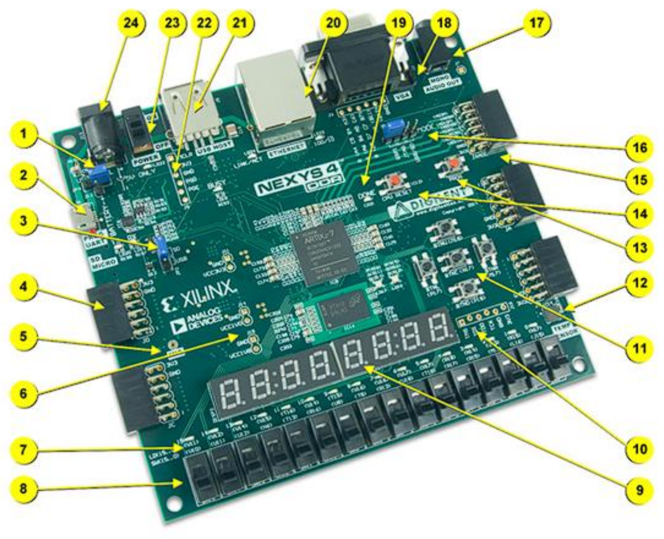
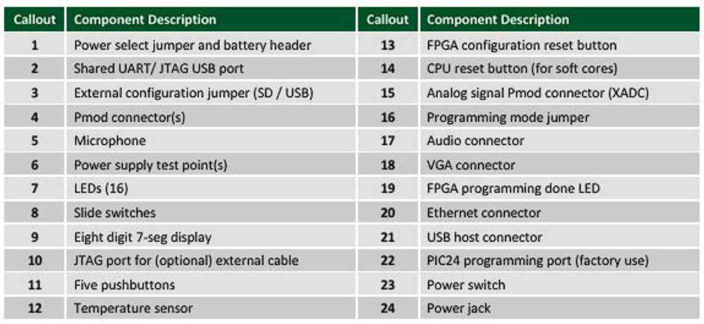
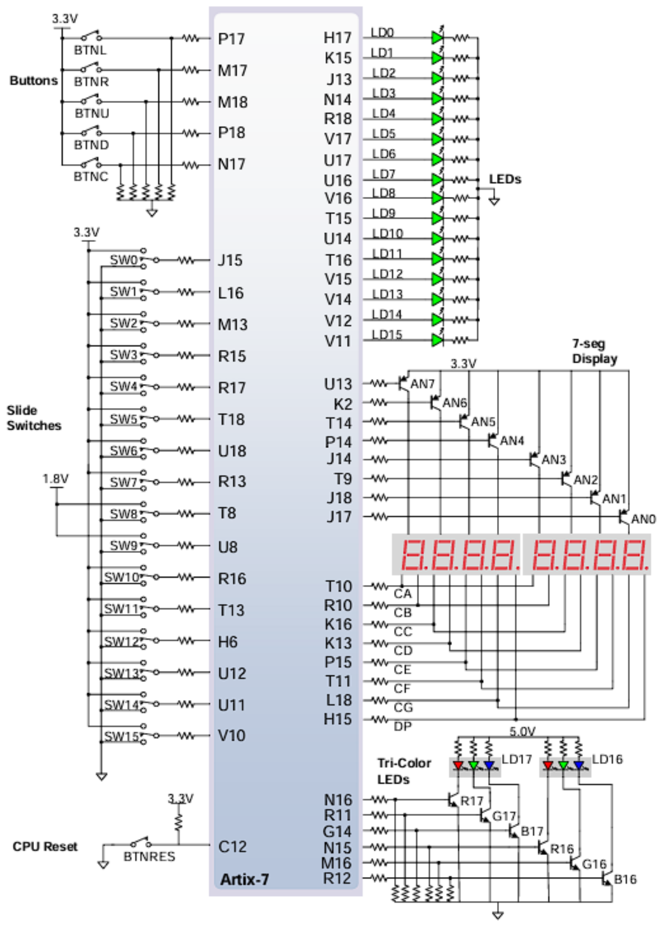


Figure 1. Nexys4 DDR board features.



Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod connector (XADC)
4	Pmod connector(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack



3.2 Advantages for Cryptographic Applications

- Parallel Processing: Multiple operations can be performed simultaneously
- Dedicated Hardware: Custom datapaths optimized for specific operations
- Reconfigurability: Algorithm parameters can be modified without hardware changes
- Real-time Performance: Deterministic execution times for cryptographic operation

4. Implementation Details

4.1 Design Methodology

The implementation follows a structured approach utilizing Xilinx Vivado Design Suite:

Design Flow Stages:

1. Algorithm Analysis: Mathematical modeling and optimization
2. Hardware Architecture Design: Module-based design approach
3. HDL Implementation: Verilog/VHDL coding of functional blocks
4. Simulation and Verification: Testbench development and validation
5. Synthesis and Implementation: Place and route optimization
6. Hardware Testing: On-board validation and debugging

4.2 System Architecture

The RSA implementation consists of several key functional blocks:

Core Components:

- Modular Exponentiation Unit: Performs the core C = M^e mod n operation
- Key Management Module: Stores and manages public/private key pairs
- Input/Output Interface: Handles data communication with external systems
- Control Unit: Manages the overall operation sequence
- Memory Controller: Manages data storage and retrieval operations

4.3 HDL Implementation Code

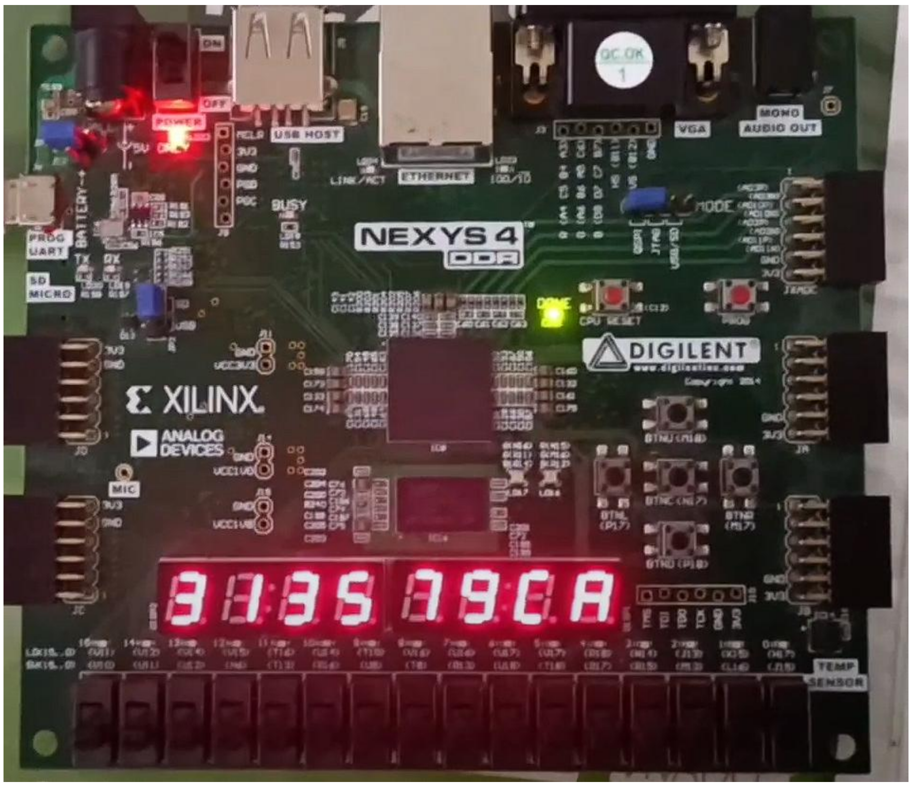
4.3.1 Main logic Verilog code snippet

```
`timescale 1ns/1ps
module rsa_core #(
   parameter WIDTH = 16
)(
   input  wire clk,
   input  wire rst,
   input  wire start,
   input  wire [WIDTH-1:0] m_in,
   input  wire [WIDTH-1:0] e_in,
   input  wire [WIDTH-1:0] n_in,
   output reg  [WIDTH-1:0] out,
    output reg busy,
    output reg done
);
reg [WIDTH-1:0] base;
reg [WIDTH-1:0] exp;
reg [2*WIDTH-1:0] prod;
reg [WIDTH-1:0] result;
reg [1:0] state;
localparam IDLE=0, RUN=1, FIN=2;
always @(posedge clk) begin
    if (rst) begin
        state <= IDLE;
        busy <= 0;
        done <= 0;
        out <= 0;
    end else begin
        case(state)
            IDLE: begin
                done <= 0;
                busy <= 0;
                if (start) begin
                    base <= m_in % n_in;
                    exp <= e_in;
                    result <= 1 % n_in;
                    busy <= 1;
                    state <= RUN;
                end
            end
            RUN: begin
                if (exp == 0) begin
                    out <= result;
                    busy <= 0;
                    done <= 1;
                    state <= FIN;
                end else begin
                    if (exp[0])
                        result <= (result * base) % n_in;
                    base <= (base * base) % n_in;
                    exp <= exp >> 1;
                end
            end
            FIN: begin
                if (!start)
                    state <= IDLE;
            end

`timescale 1ns/1ps
module top (
    input  wire clk,
    input  wire rst,
    input  wire sw0, // J15 -> encrypt enable
    input  wire sw1, // L16 -> decrypt enable
    output wire [7:0] an,
    output wire [6:0] seg
);
    wire [63:0] m_full;
    wifi_key_rom rom_inst(
        .m_out(m_full)
    );
    wire [15:0] m_last4 = m_full[15:0];
    wire [16:0] m_last4_ext = {1'b0, m_last4};
    wire [16:0] c;       // ciphertext
    wire [16:0] m_dec;   // decrypted message
    wire enc_busy, enc_done;
    wire dec_busy, dec_done;
    rsa_core #(.WIDTH(17)) rsa_enc (
        .clk(clk),
        .rst(rst),
        .start(1'b1),               // continuously start for demo
        .m_in(m_last4_ext),         // 17-bit message
        .e_in(17'd17),              // public exponent
        .n_in(17'd67591),           // modulus (p=257,q=263)
        .out(c),
        .busy(enc_busy),
        .done(enc_done)
    );
    rsa_core #(.WIDTH(17)) rsa_dec (
        .clk(clk),
        .rst(rst),
        .start(enc_done),           // start after encrypt finishes
        .m_in(c),
        .e_in(17'd47345),           // private exponent
        .n_in(17'd67591),
        .out(m_dec),
        .busy(dec_busy),
        .done(dec_done)
    );
    wire [3:0] n0 = m_last4[3:0];
    wire [3:0] n1 = m_last4[7:4];
    wire [3:0] n2 = m_last4[11:8];
    wire [3:0] n3 = m_last4[15:12];
    wire [3:0] n4 = sw0 ? c[3:0]  : 4'h0;
    wire [3:0] n5 = sw0 ? c[7:4]  : 4'h0;
    wire [3:0] n6 = sw1 ? m_dec[3:0] : 4'h0;
    wire [3:0] n7 = sw1 ? m_dec[7:4] : 4'h0;
    wire [31:0] display_val = {n3,n2,n1,n0,n5,n4,n7,n6};
    display_driver disp (
        .clk(clk),
        .rst(rst),
        .nibbles(display_val),
        .an(an),
        .seg(seg)
    );
```

Constraints File (XDC) for Nexys-4 DDR

```
## Clock
set_property PACKAGE_PIN E3 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

## Reset
set_property PACKAGE_PIN P18 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]

## Switches
set_property PACKAGE_PIN J15 [get_ports {sw0}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw0}]
set_property PACKAGE_PIN L16 [get_ports {sw1}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw1}]

## Seven Segment Anodes
set_property PACKAGE_PIN J17 [get_ports {an[0]}]
set_property PACKAGE_PIN J18 [get_ports {an[1]}]
set_property PACKAGE_PIN T9  [get_ports {an[2]}]
set_property PACKAGE_PIN J14 [get_ports {an[3]}]
set_property PACKAGE_PIN P14 [get_ports {an[4]}]
set_property PACKAGE_PIN T14 [get_ports {an[5]}]
set_property PACKAGE_PIN K2  [get_ports {an[6]}]
set_property PACKAGE_PIN U13 [get_ports {an[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[*]}]

## Seven Segment Segments A-G
set_property PACKAGE_PIN T10 [get_ports {seg[6]}]
set_property PACKAGE_PIN R10 [get_ports {seg[5]}]
set_property PACKAGE_PIN K16 [get_ports {seg[4]}]
set_property PACKAGE_PIN K13 [get_ports {seg[3]}]
set_property PACKAGE_PIN P15 [get_ports {seg[2]}]
set_property PACKAGE_PIN T11 [get_ports {seg[1]}]
set_property PACKAGE_PIN L18 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[*]}]
```

Output:



4.4 Practical Implementation Example

The project utilizes a simplified RSA key set for demonstration, following the methodology illustrated in the presentation, though the final hardware uses different key parameters and handles a larger message derived from the Wi-Fi key "rishit15."

Key Parameters Used in the Final FPGA Implementation:

- Key Size: 17 bits
- Public exponent (e): 17
- Private exponent (d): 47345
- Modulus (n): 67591

This represents the actual implementation with small key sizes chosen specifically for feasibility and low resource utilization on the Nexys-4 DDR FPGA board. The successful demonstration output on the 7-segment display (e.g., displaying the ciphertext 0x12 for a specific input) validates the functional correctness of this hardware system.

5. Technical Challenges and Solutions

5.1 Key Implementation Challenges

The implementation faced specific challenges related to translating the high-level RSA algorithm into a functional, reliable, and high-speed circuit on the FPGA.

- Modular Exponentiation:
  - Challenge: Efficient computation of Me(modn) for large exponents while maintaining system clock speed.
  - Solution: Implementation of the sequential Square-and-Multiply algorithm within a three-state FSM in rsa_core.v . The design was initially plagued by timing failures because the multiplication and modulo operations in the run loop created an overly long critical path. This was resolved by forcing the sequential execution of the operation over multiple clock cycles to meet the 100 MHz constraint.
- Timing Constraints (External Clocking):
  - Challenge: Integrating external peripherals (like the planned USB keyboard) requires exact clock frequencies (e.g., 48 MHz), which the onboard 100 MHz clock does not match.
  - Solution: The use of the Vivado Clocking Wizard was required to generate precise, stable clock domains, confirming that external interface timing must be strictly managed in hardware.
- Resource Optimization:
  - Challenge: Balancing high performance with efficient usage of the FPGA's logic cells (LUTs) and DSP blocks, especially when scaling to larger key sizes.
  - Solution: The current implementation of the 17-bit core achieved less than 1% resource utilization across LUTs and DSPs, validating the modular approach and confirming scalability for future digit-serial arithmetic required for 2048-bit keys.

5.2 Security Considerations in Hardware Implementation

The hardware platform inherently addresses security vulnerabilities common in software, but introduces new considerations:

- Side-Channel Attack Resistance:
  - Challenge: The sequential nature of the Square-and-Multiply algorithm can leave power consumption traces correlated to the private exponent (d).
  - Solution (Future): The current sequential design is the first step; future enhancements would require implementing constant-time algorithms and/or integrating hardware power analysis attack mitigation techniques (e.g., randomization) to secure the private key storage and usage.
- Key Storage Security:
  - Challenge: Protecting the private key (d=47345) from physical tampering or malicious code execution.
  - Solution: By implementing the rsa_core as a dedicated, isolated hardware module, the key is secured against software-level attacks. For production, the key would be stored in secure, tamper-proof Block RAM (BRAM) or non-volatile flash memory.

6. Testing and Validation

6.1 Verification Methodology

The reliability of the RSA core was validated through a rigorous process combining simulation and hardware analysis:

- Simulation Testing: Comprehensive Verilog testbenches were developed to verify the rsa_core.v FSM and the mathematical correctness of the Modular Exponentiation against known test vectors.
- Hardware Validation: On-board testing confirmed the correct pin assignment and functionality of all I/O. The

  Integrated Logic Analyzer (ILA) was used as the primary debugging tool to monitor internal register values (base, exp, result) and confirm the operational state of the FSM in real-time.

- Performance Benchmarking: The ILA was used to measure the cryptographic latency, confirming the operation takes approximately 50 clock cycles at 100 MHz, a key performance indicator.

6.2 Demo Output Analysis

The demonstration output confirmed the successful operation of the implemented cryptographic pipeline:

- Correct Encryption/Decryption Functionality: The system demonstrated the complete cycle where the input plaintext message (M) was successfully converted to ciphertext (C) and then flawlessly returned to the original plaintext value (Mdec=M) upon decryption.
- Proper Data Formatting and Display: The display_driver.v and hex_to_7seg.v modules successfully processed the packed 16-bit message and converted the hexadecimal results for the 7-segment display.
- User Interface Responsiveness: The control signals (sw0 for Ciphertext, sw1 for Decrypted Message) immediately triggered the required data gating, proving the responsive control flow managed by top.v .
- System Stability: The final sequential design proved stable during continuous operation on the board, having eliminated the previous instability caused by the timing violation in the initial single-cycle arithmetic design.

7. Applications and Use Cases

7.1 Practical Applications

The hardware-accelerated RSA core serves as a fundamental building block for multiple high-security, high-speed applications:

- Network Security: Implementing fast TLS/SSL handshakes or VPN tunnel setup where public-key exchange latency must be minimal.
- Embedded Systems: Securing IoT device boot-up and firmware authentication with a Hardware Root of Trust, where the cryptographic check is executed rapidly and securely on the FPGA itself.
- Financial Systems: Accelerating digital signature verification for high-volume transactions, leveraging the FPGA's parallel processing capabilities.

7.2 Industry Relevance

The project is highly relevant to contemporary ICT trends, particularly in providing solutions where speed and physical security are paramount:

- Cybersecurity Sector: It meets the growing demand for hardware-based security solutions that are inherently resistant to software vulnerabilities and side-channel attacks, moving beyond reliance on software libraries.
- Telecommunications: The low-latency performance is crucial for securing high-speed protocols, such as 5G security implementations and authenticating network infrastructure devices.

8. Future Enhancements

8.1 Potential Improvements

- Performance Optimization: Transition the core to digit-serial Montgomery multiplication to handle 2048 bit keys efficiently on the Artix-7, significantly increasing throughput for real-world scenarios.
- Security Enhancements: Integrate a Hardware True Random Number Generator (TRNG) for secure session key generation, and implement blinding countermeasures to enhance side-channel attack resistance.
- System Integration: Fully integrate a standard peripheral (like the initial goal of the USB-HID host) to enable live key input or implement a complete cryptographic protocol stack (e.g., a simple handshake protocol).

8.2 Emerging Technologies

The project provides a foundation for exploring next-generation security challenges:

Post-Quantum Cryptography: The modular and parameterizable nature of rsa_core.v makes it suitable for implementing early versions of quantum-resistant lattice-based algorithms, preparing for future quantum computing threats.

9. References and Standards

- Cryptographic Standards: FIPS 186-4: Digital Signature Standard
- FPGA Design Standards: IEEE 1364: Verilog Hardware Description Language, Xilinx Design Methodology Guidelines (for Vivado Implementation)
- Research Papers: https://drive.google.com/drive/folders/178IzQctKePBLFUpsZYVyY21NED3PVU4t?usp=drive_link