**4-bit RSA Encoder Decoder**

**Objectives:**

* To design and implement a 4-bit RSA encryption and decryption system using Verilog.
* To demonstrate the functionality of RSA key generation, encryption, and decryption at the hardware level.
* To perform modular arithmetic operations efficiently within a hardware-based environment.
* To validate the correctness of the RSA algorithm through Verilog simulation and testing.
* To provide a foundational understanding of public-key cryptography in hardware design.

**Introduction:**

The RSA algorithm is a widely used public-key cryptographic method that ensures secure communication by leveraging the computational difficulty of factoring large prime numbers. This project focuses on designing and implementing a 4-bit RSA encoder-decoder system using Verilog at the hardware level. The system encompasses key generation, encryption, and decryption modules, demonstrating the fundamental principles of RSA in a simplified context. By utilizing modular arithmetic operations, the project illustrates how secure data encoding and decoding can be achieved in constrained environments. The 4-bit design serves as a practical example for understanding cryptographic operations and their implementation in digital systems.

**Design:**

The block diagram, state diagram and flow chart for the project is drawn. Based on the state diagram the control logic is designed using one state per flip flop method in the following tables and equations:

Table 1: Input signals generation based on the state diagram

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Current State | Next State | init | e | d | En | De | H1 | H4 | H6 | H8 | H9 | H12 | H13 | H14 | H15 |
| S0 | S0 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S0 | S1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S1 | S1 |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| S1 | S2 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| S2 | S3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S3 | S4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S4 | S4 |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |
| S4 | S5 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |
| S5 | S6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S6 | S6 |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |
| S6 | S7 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| S7 | S6 |  | 0 |  |  |  |  |  |  |  |  |  |  |  |  |
| S7 | S8 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| S8 | S8 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |
| S8 | S9 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| S9 | S9 |  |  |  |  |  |  |  |  |  | 0 |  |  |  |  |
| S9 | S10 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| S10 | S8 |  |  | 0 |  |  |  |  |  |  |  |  |  |  |  |
| S10 | S11 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
| S11 | S14 |  |  |  | 0 |  |  |  |  |  |  |  |  |  |  |
| S11 | S12 |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |
| S12 | S12 |  |  |  |  |  |  |  |  |  |  | 0 |  |  |  |
| S12 | S13 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |
| S13 | S13 |  |  |  |  |  |  |  |  |  |  |  | 0 |  |  |
| S13 | S0 |  |  |  |  | 0 |  |  |  |  |  |  | 1 |  |  |
| S13 | S14 |  |  |  |  | 1 |  |  |  |  |  |  | 1 |  |  |
| S14 | S14 |  |  |  |  |  |  |  |  |  |  |  |  | 0 |  |
| S14 | S15 |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |
| S15 | S15 |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| S15 | S0 |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |

Based on the table the following state equations can be generated:

* S0 = (S0 & ~init) | (S13 & H13 & ~De) | (S15 & H15)
* S1 = (S0 & init) | (S1 & ~H1)
* S2 = S1 & H1
* S3 = S2
* S4 = S3 | (S4 & ~H4)
* S5 = S4 & H4
* S6 = S5 | (S6 & ~H6) | (S7 & ~e)
* S7 = S6 & H6
* S8 = (S7 & e) | (S8 & ~H8) | (S10 & ~d)
* S9 = (S8 & H8) | (S9 & ~H9)
* S10 = S9 & H9
* S11 = S10 & d
* S12 = (S11 & En) | (S12 & ~H12)
* S13 = (S12 & H12) | (S13 & ~H13)
* S14 = (S11 & ~En) | (S13 & H13 & De) | (S14 & ~H14)
* S15 = (S14 & H14) | (S15 & ~H15)

Table 2: Output signals based on the control signals

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | load | mul | dec | gcd | cmp | mod | pow | out | sel | Inc |
| S0 |  |  |  |  |  |  |  |  |  |  |
| S1 |  | 1 |  |  |  |  |  |  | 0 |  |
| S2 | 1 |  |  |  |  |  |  |  | 0 |  |
| S3 |  |  | 1 |  |  |  |  |  |  |  |
| S4 |  | 1 |  |  |  |  |  |  | 0 |  |
| S5 | 1 |  |  |  |  |  |  |  | 1 |  |
| S6 |  |  |  | 1 |  |  |  |  |  |  |
| S7 |  |  |  |  | 1 |  |  |  | 0 | 1 |
| S8 |  | 1 |  |  |  |  |  |  | 1 |  |
| S9 |  |  |  |  |  | 1 |  |  | 0 |  |
| S10 |  |  |  |  | 1 |  |  |  | 1 | 1 |
| S11 |  |  |  |  |  |  |  |  |  |  |
| S12 |  |  |  |  |  |  | 1 |  | 0 |  |
| S13 |  |  |  |  |  | 1 |  | 1 | 1 |  |
| S14 |  |  |  |  |  |  | 1 |  | 1 |  |
| S15 |  |  |  |  |  | 1 |  | 1 | 1 |  |

Based on the table the following output equations can be generated:

* load = S2 | S5
* mul = S1 | S4 | S8
* dec = S3
* gcd = S6
* cmp = S7 | S10
* mod = S9 | S13 | S15
* pow = S12 | S14
* out = S13 | S15
* sel = S5 | S8 | S10 | S13 | S14 | S15
* inc = S7 | S10

Based on these control signals the ALU and other modules are controlled and all the states can be generated and simulated.

**Discussion:**

The implementation of a 4-bit RSA encoder-decoder system allowed us to successfully perform basic key generation, including the computation of the modulus (n), Euler’s totient (phi), and partial GCD calculations to identify valid public and private keys. However, due to the inherent complexity of modular exponentiation required for encryption and decryption, these operations could not be fully implemented at the hardware level within the constraints of this project. The high computational overhead and intricate resource management made it challenging to realize encryption and decryption circuits effectively in Verilog. While the project demonstrates the feasibility of key generation, it highlights the significant hardware challenges involved in executing the full RSA algorithm.

**Conclusion:**

In this project, a 4-bit RSA encoder-decoder system was designed using Verilog, focusing on hardware-level implementation of basic RSA principles. While we successfully implemented key generation and partial GCD computations, the complexity of modular exponentiation prevented the realization of encryption and decryption processes at the hardware level. This limitation underscores the challenges of implementing computationally intensive cryptographic algorithms in constrained hardware environments.

**References:**

* Class slides
* Class lectures
* Lab lectures