TRABALHO PRÁTICO: CIFRA XOR

Introdução aos Sistemas Lógicos

Aluno: Marney Santos de Melo

Matrícula: 2024106034

Professora: Michele Nogueira

Data: 07 de Julho de 2025

Observação: O projeto foi implementado utilizando SystemVerilog (.sv), conforme autorizado pela professora em conversa no dia 07/07/2025.

1. IMPLEMENTAÇÃO PRINCIPAL (cipher.sv)

1.1. Descrição do Funcionamento

O projeto implementa um módulo de cifra XOR sequencial, conforme especificado no enunciado do trabalho. O projeto utiliza uma Máquina de Estados Finita (FSM) para controlar o processo de cifragem, que ocorre bit a bit ao longo de múltiplos ciclos de clock.

A FSM gerencia o fluxo de operação através de quatro estados principais:

IDLE: Estado de espera inicial. O circuito aguarda o sinal `start` ser ativado para iniciar a operação. A saída `done` é mantida em 0.

LOAD: Ao receber o sinal `start`, o estado transita para `LOAD`. Neste estado, os valores de `plaintext` e `key` das entradas são carregados nos registradores.

PROCESS: Neste estado, a cifragem ocorre. A cada ciclo de clock, um bit do `plaintext` é combinado com o bit correspondente da `key` através da operação XOR. Um contador de bits (`bit_counter`) controla o processo por 8 ciclos.

DONE: Após os 8 ciclos, o contador indica o fim do processamento e a FSM transita para o estado `DONE`. Aqui, o sinal de saída `done` é ativado para indicar a conclusão, e o resultado final é mantido na saída `ciphertext. O sistema retorna ao estado `IDLE` quando o sinal `start` for desativado.

1.2. Justificativa das Decisões de Projeto

Linguagem (SystemVerilog): Apesar de o enunciado pedir em Verilog, foi usado o SystemVerilog, com autorização da professora. Essa linguagem foi escolhida porque ela tem recursos mais modernos que ajudam a deixar o código mais claro, organizado e fácil de entender. Por exemplo, foram usadas formas mais simples de declarar sinais e estados, além de blocos que separam bem o que acontece a cada

ciclo de clock e o que depende só das condições do sistema. Isso tudo ajudou a evitar erros e deixou o projeto mais tranquilo de desenvolver.

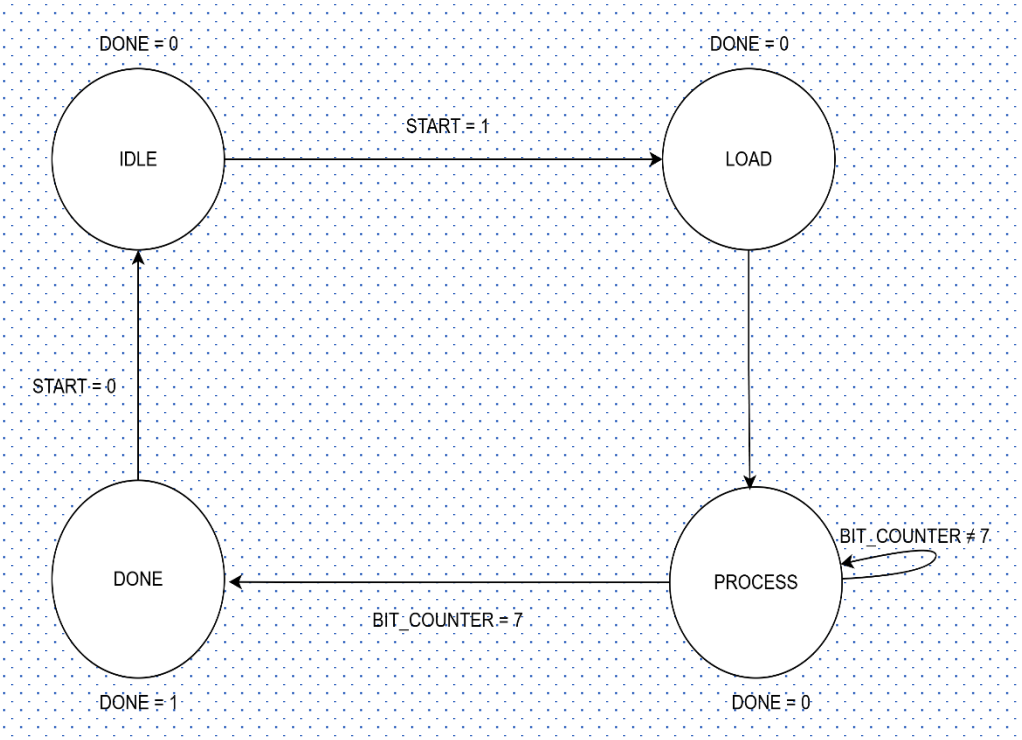
Máquina de Estados (FSM): A FSM foi implementada seguindo o modelo de Moore, onde as saídas dependem apenas do estado atual. A FSM foi dividida em dois blocos `always`: um `always_ff` para o registro síncrono dos estados e outro `always_comb` para a lógica combinacional que define as transições e as saídas. Esta é uma prática padrão de design que resulta em um circuito robusto e de fácil depuração.

Controle de Processamento: Um contador de 3 bits (`bit_counter`) foi usado no estado `PROCESS` para controlar os 8 ciclos necessários para a cifragem bit a bit, garantindo que a operação seja concluída no tempo correto, conforme especificado.

1.3 Diagrama da FSM

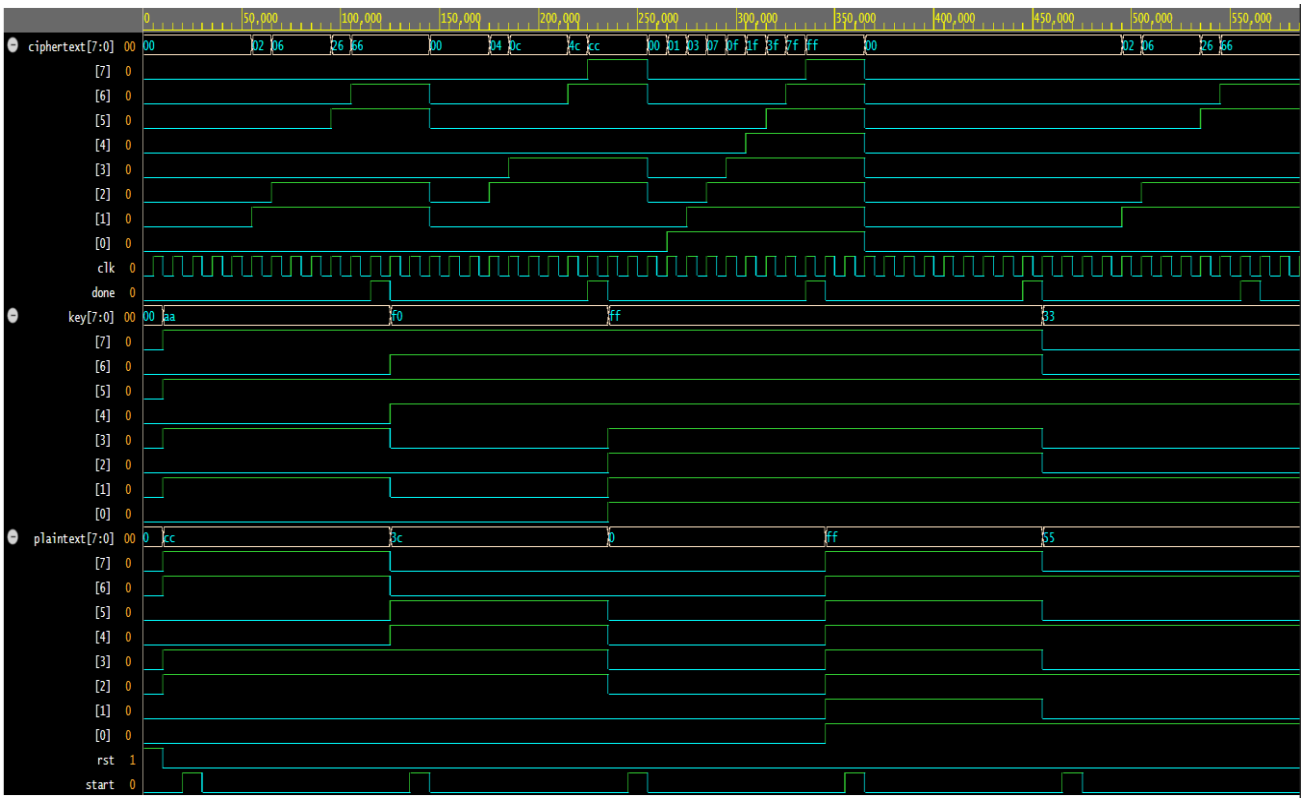
O diagrama abaixo representa as transições de estado da FSM implementada.

IDLE: Saída `done` = 0.
LOAD: Saída `done` = 0.
PROCESS: Saída `done` = 0.
DONE: Saída `done` = 1



1.4. Formas de Onda

As formas de onda a seguir foram geradas a partir do `testbench` (`cipher_tb.sv`) e demonstram o funcionamento do circuito obtido a partir dos casos de test.



2. IMPLEMENTAÇÃO BÔNUS (cipher_bonus.sv)

2.1. Descrição do Funcionamento

Para obter a bonificação, o módulo `cipher` foi estendido para uma versão que possui um parâmetro, um módulo `cipher_bonus`, que agora pode lidar com um texto claro (`plaintext`) de tamanho `N`, que pode ser maior que o tamanho da chave de 8 bits. A lógica principal da FSM permanece a mesma, mas o processamento é adaptado para N ciclos, e a chave é reutilizada conforme necessário.

2.2. Justificativa das Decisões de Projeto

Parametrização do Módulo: O tamanho do `plaintext` foi definido como um parâmetro `N`, permitindo que o mesmo módulo seja utilizado para qualquer tamanho de dado (ex: 16, 24, 32 bits), tornando o código mais flexível e reutilizável.

Contador Dinâmico: O contador de bits (`bit_counter`) foi projetado para se adaptar automaticamente ao tamanho `N` do `plaintext`. Sua largura é definida por ***`[$clog2(N):0]`***,

esse comando calcula o menor inteiro que é maior ou igual ao logaritmo de N na base 2, garantindo que ele sempre terá bits suficientes para contar até `N-1`, para qualquer valor de `N`.

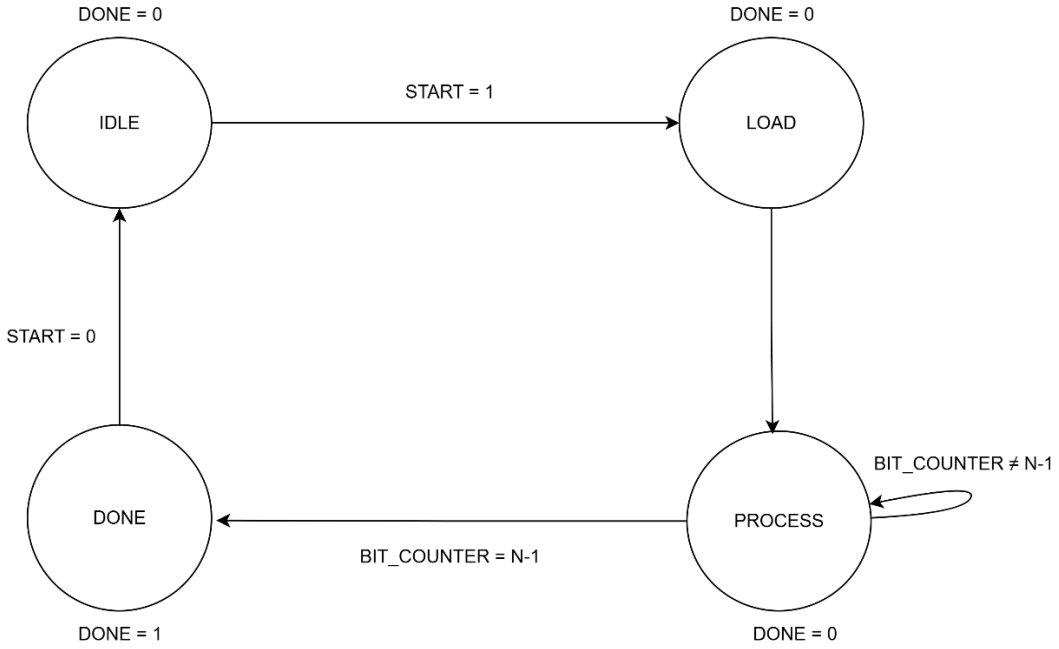
Lógica de Reutilização da Chave: Esta é a principal modificação para o bônus. No estado `PROCESS`, a seleção do bit da chave a ser usado na operação XOR é feita pela expressão:

$$`key\_reg[bit\_counter \% 8]`$$

O uso do operador módulo (`% 8`) faz com que o índice do vetor da chave "reinicie" para 0 a cada 8 ciclos. Isso implementa de forma eficiente e elegante a repetição da chave de 8 bits para cifrar todo o texto de `N` bits.

2.3 Diagrama da FSM

O diagrama abaixo representa as transições de estado da FSM implementada para o bônus, a única modificação é o critério de transição de process para done.



2.4. Formas de Onda

As formas de onda para a versão bônus foram geradas usando o `cipher_bonus_tb.sv`, com o parâmetro `N` ajustado para 16.