# Introducão aos Sistemas Lógicos - Trabalho Prático em Verilog

Lucas Rafael Costa Santos
2021017723

Dezembro 2023 Belo Horizonte-MG

## 1 Introdução

Este trabalho visa explorar a implementação do Vernam Cipher em Verilog, integrando conceitos de lógica combinatorial, lógica sequencial e noções básicas de criptografia. O Vernam Cipher, também conhecido como one-time pad, foi proposto por Claude Shannon da Bell Labs na década de 1940, demonstrando teoricamente que, quando implementado corretamente, o one-time pad é inquebrável.

O projeto foi dividido em duas partes distintas. A primeira parte consiste na implementação de um flip-flop do tipo D em Verilog, enquanto a segunda parte abrange a implementação de registradores e do stream cipher, incluindo montagem de registradores com One-Time Pad (OTP) e mensagem, operação XOR para cifragem, decifragem da mensagem e manipulação de streams. Os desafios aqui presentes incluem lidar com mensagens de tamanho superior ao das chaves, além da necessidade de trabalhar com deslocadores para processamento eficiente de streams.

## 2 Metodologia

Para desenvolvermos o one-time pad adotando a linguagem Verilog, seguimos uma metodologia que envolveu a criação de diversos módulos essenciais, cada um desempenhando um papel específico no processo de cifragem. Estes módulos são:

1. Flip-Flop do tipo D:

   Módulo que armazena um único bit de informação, sendo sensível à borda de subida do sinal de clock e passível de ser resetado por um sinal de reset. O flip-flop é atualizado com os dados de entrada na ausência de um sinal de reset, garantindo que sua saída reflita o bit de dados mais recente.

2. Registrador de Deslocamento:

   É capaz de armazenar e deslocar bits de informação, sendo sensível à borda de subida do sinal de clock e podendo ser resetado por um sinal de reset. Quando ativado, os dados são deslocados para a direita, sendo que, o registrador de deslocamento realiza o deslocamento dos dados apenas quando o sinal de habilitação de deslocamento está ativo.

3. Cifrador de Fluxo (Stream Cipher):

   Tendo a função de cifrar uma mensagem utilizando um one-time pad, este módulo utiliza o registrador de deslocamento para tratar a mensagem como um fluxo de bits. A operação XOR entre a mensagem e o one-time pad é executada para cifrar a mensagem, assim, a cifragem ocorre na borda de subida do sinal de clock e o resultado da operação XOR é utilizado para atualizar a mensagem cifrada.

4. Porta XOR:

   Foi implementada uma porta XOR no estilo estrutural, utilizando portas NOT, AND e OR para realizar a operação XOR entre dois bits.A saída da porta XOR é calculada considerando as entradas e o resultado das portas AND e OR.

Para avaliar a implementação, elaboramos módulos de teste específicos. Um desses módulos cifra a palavra "estrelas" utilizando um one-time pad predefinido, enquanto o outro decifra a palavra resultante. Ambos os módulos de teste geram um arquivo VCD, permitindo a simulação e visualização dos sinais envolvidos no processo de criptografia. Essa abordagem proporciona uma verificação eficiente e eficaz da correta operação do sistema de criptografia implementado.

- Implementação porta XOR, Flip-Flop D, Registrador e Cifrador:

```
// Modulo para a porta XOR implementada no estilo estrutural
module xor_gate(a, b, out);
 input a, b;
 output out;
 wire a_bar, b_bar, x, y;

 // Complemento (inverso) das entradas
 not a_inv (a_bar, a);
 not b_inv (b_bar, b);

 // AND gates para a implementacao da XOR
 and a1 (x, a_bar, b);
 and a2 (y, a, b_bar);

 // OR gate para a saida da XOR
 or or1 (out, x, y);
endmodule

// Modulo do Flip-Flop D
module d_flip_flop (
 input wire clk,
 input wire rst,
 input wire d,
 output reg q
);

 always @(posedge clk or posedge rst) begin
   if (rst) begin
     q <= 1'b0; // Reseta o flip-flop
   end else begin
     q <= d;    // Atualiza o flip-flop com os dados de entrada
   end
 end

endmodule

// Modulo do Registrador de Deslocamento
module shift_register #(parameter WIDTH = 32)(
 input wire clk,
 input wire rst,
 input wire [WIDTH-1:0] data_in,
 input wire shift_enable,
 output reg [WIDTH-1:0] data_out
);

 always @(posedge clk or posedge rst) begin
   if (rst) begin
     data_out <= {WIDTH{1'b0}}; // Reseta o registrador de deslocamento
   end else begin
     if (shift_enable) begin
       data_out <= {data_in[WIDTH-1], data_out[WIDTH-1:1]}; // Desloca os dados
     end
   end
 end

endmodule

// Modulo do Cifrador de Fluxo
module stream_cipher #(parameter BITS = 64)(
 input wire clk,
 input wire rst,
 input wire [BITS-1:0] message,
 input wire [BITS/2-1:0] otp,
 output reg [BITS-1:0] ciphered_message
);

 wire [BITS-1:0] shift_register_out;

 // Instanciacao do registrador de deslocamento
 shift_register #(BITS) sr (
   .clk(clk),
   .rst(rst),
   .data_in(message),
```

```
    .shift_enable(1'b1),
    .data_out(shift_register_out)
  );

  // Operacao XOR para criptografia
  always @(posedge clk or posedge rst) begin
    if (rst) begin
      ciphered_message <= 0; // Reseta a mensagem cifrada
    end else begin
      // Atualiza a mensagem cifrada usando a saida do registrador de deslocamento e o OTP
      ciphered_message <= shift_register_out ^ {otp, {BITS/2{1'b0}}};
    end
  end

endmodule
```

**Listing 1:** *design.sv*

- Testbench - Cifrador:

```
// Modulo de teste para o cifrador
module cifra #(parameter BITS = 64);
  // Registrador de tamanho 64, contendo a palavra "estrelas" em ASCII
  reg [BITS-1:0] in = 64'b01100101_01110011_01110100_01110010_01100101_01101100_01100001_01110011;
  // Registrador de tamanho 32, contendo o OTP
   reg [BITS/2-1:0] otp = 32'b01100110_01101001_01110110_01100101;
  // Registradores e fios auxiliares, para cifrar a mensagem bit-a-bit.
  // A saida precisa ser um fio, pois a porta XOR precisa produzir seu sinal
  // em um meio que permita atribuicao continua.
  reg a, b;
  wire out;
  // Instanciacao da porta XOR
  xor_gate xorg(.a(a), .b(b), .out(out));

  // Inicializacao do arquivo VCD para simulacao
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1, a, b, out);
  end

  // Contador do loop for
  integer i;
  integer bits_per_group = 8; // Numero de bits por grupo
  integer bits_counter = 0;  // Contador de bits no grupo

  // Inicializacao do loop for para cifrar a mensagem bit-a-bit
  initial begin
    for(i = BITS-1; i >= 0; i = i - 1) begin
      a = in[i];
      b = (i >= BITS/2) ? otp[i-BITS/2] : otp[i]; // Correcao na logica para selecionar os bits
     corretos do OTP
      #1
      // Exibe o bit atual
      $write(out);

      // Contagem de bits no grupo
      bits_counter = bits_counter + 1;

      // Adiciona espaco entre grupos
      if (bits_counter == bits_per_group) begin
        $write(" ");
        bits_counter = 0;
      end
    end

    // Adiciona uma quebra de linha no final
    $write("\n");
  end
endmodule
```

**Listing 2:** *testbench.sv - Cifrador*

- Testbench - Decifrador:

```
// Modulo de teste para o decifrador
module decifra #(parameter BITS = 64);
 // Mensagem cifrada de entrada
  reg [BITS-1:0] in = 64'b00000011_00011010_00000010_00010111_00000011_00000101_00010111_00010110;
 // Registrador de tamanho 32, contendo o OTP
 reg [BITS/2-1:0] otp = 32'b01100110_01101001_01110110_01100101;
 // Registradores e fios auxiliares, para decifrar a mensagem bit-a-bit.
 // A saida precisa ser um fio, pois a porta XOR precisa produzir seu sinal
 // em um meio que permita atribuicao continua.
 reg a, b;
 wire out;
 // Instanciacao da porta XOR
 xor_gate xorg(.a(a), .b(b), .out(out));

 // Inicializacao do arquivo VCD para simulacao
 initial begin
   $dumpfile("dump_decifra.vcd");
   $dumpvars(1, a, b, out);
 end

 // Contador do loop for
 integer i;
 integer bits_per_group = 8; // Numero de bits por grupo
 integer bits_counter = 0;  // Contador de bits no grupo

 // Inicializacao do loop for para decifrar a mensagem bit-a-bit
 initial begin
   for(i = BITS-1; i >= 0; i = i - 1) begin
     a = in[i];
     b = (i >= BITS/2) ? otp[i-BITS/2] : otp[i]; // Correcao na logica para selecionar os bits
    corretos do OTP
     #1
     // Exibe o bit atual
     $write(out);

     // Contagem de bits no grupo
     bits_counter = bits_counter + 1;

     // Adiciona espaco entre grupos
     if (bits_counter == bits_per_group) begin
       $write(" ");
       bits_counter = 0;
     end
   end

   // Adiciona uma quebra de linha no final
   $write("\n");
 end
endmodule
```

**Listing 3:** *testbench.sv - Decifrador*

# 3 Resultados

## 3.1 Cifrador

Para realizar o teste de cifragem, empregamos a palavra 'estrelas' em sua representação binária de 64 bits (01100101 01110011 01110100 01110010 01100101 01101100 01100001 01110011) e a chave de 32 bits (01100110 01101001 01110110 01100101). Ao efetuar a operação XOR de forma manual, podemos ver a resposta por meio da tabela abaixo:

| Palvra | 01100101 | 01110011 | 01110100 | 01110010 | 01100101 | 01101100 | 01100001 | 01110011 |
|---|---|---|---|---|---|---|---|---|
| Chave | 01100110 | 01101001 | 01110110 | 01100101 | 01100110 | 01101001 | 01110110 | 01100101 |
| Resultado | 00000011 | 00011010 | 00000010 | 00010111 | 00000011 | 00000101 | 00010111 | 00010110 |

**Table 1:** *Tabela de Cifragem*

Realizando o nosso teste obtivemos os seguintes resultados:

```
VCD info: dumpfile dump.vcd opened for output.
00000011 00011010 00000010 00010111 00000011 00000101 00010111 00010110
```

**Figure 1:** *Resultado - Teste1*

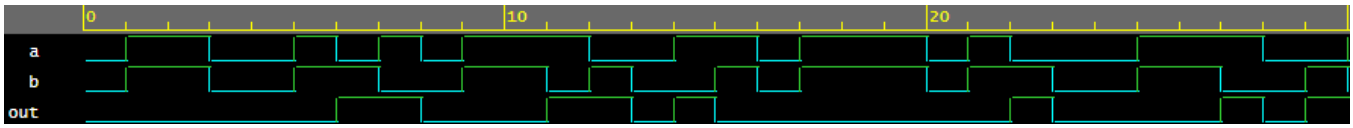


**Figure 2:** *EP Wave 1 - Parte 1*

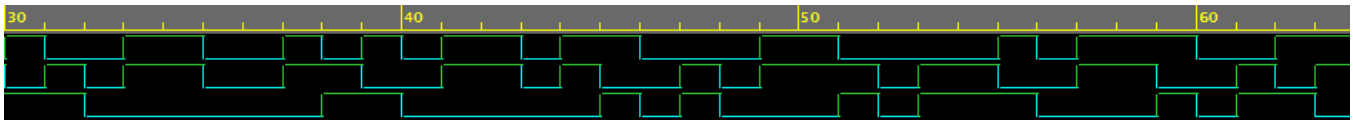


**Figure 3:** *EP Wave 1 - Parte 2*

Como podemos ver, os resultados obtidos estão de acordo com a operação feita manualmente.

## 3.2 Decifrador

Agora, para decifrar, pegamos o resultado anterior (00000011 00011010 00000010 00010111 00000011 00000101 00010111 00010110) e fazemos a operação utilizando a mesma chave (01100110 01101001 01110110 01100101). A resposta esperada é nossa palavra 'estrelas' (01100101 01110011 01110100 01110010 01100101 01101100 01100001 01110011).

| Palvra | 00000011 | 00011010 | 00000010 | 00010111 | 00000011 | 00000101 | 00010111 | 00010110 |
|---|---|---|---|---|---|---|---|---|
| Chave | 01100110 | 01101001 | 01110110 | 01100101 | 01100110 | 01101001 | 01110110 | 01100101 |
| Resultado | 01100101 | 01110011 | 01110100 | 01110010 | 01100101 | 01101100 | 01100001 | 01110011 |

**Table 2:** *Tabela de Decifragem*

Executando este teste temos os seguintes resultados:

```
VCD info: dumpfile dump_decifra.vcd opened for output.
01100101 01110011 01110100 01110010 01100101 01101100 01100001 01110011
```

**Figure 4:** *Resultado - Teste2*

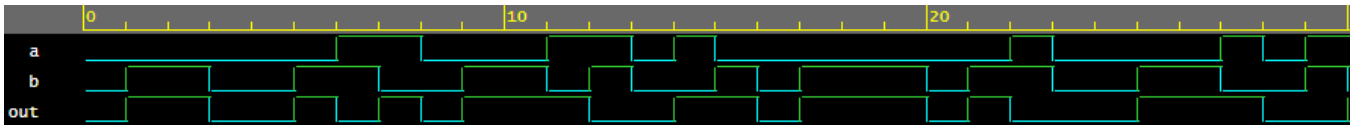


**Figure 5:** *EP Wave 2 - Parte 1*

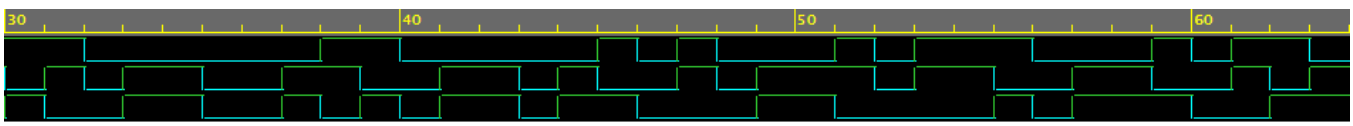


**Figure 6:** *EP Wave 2 - Parte 2*

Novamente, os resultados atenderam às espectativas sobre o que era esperado.

# 4 Conclusão

Neste trabalho, foi apresentada a implementação do Vernam Cipher em Verilog, explorando conceitos de lógica combinatória, lógica sequencial e criptografia básica. A implementação foi dividida em duas partes principais: a implementação de um flip-flop do tipo D e a implementação de registradores e do stream cipher.

A implementação do flip-flop do tipo D e do registrador de deslocamento foi bem-sucedida, com os módulos funcionando conforme o esperado. Além disso, o módulo do cifrador de fluxo, que utiliza o registrador de deslocamento para tratar a mensagem como um fluxo de bits e realiza a operação XOR entre a mensagem e o one-time pad para cifrar a mensagem, também foi implementado com sucesso.

Os testes realizados confirmaram a correta operação dos módulos implementados, de forma que, a palavra "estrelas" foi cifrada com sucesso usando um one-time pad predefinido, e a palavra cifrada foi decifrada corretamente. Assim, os resultados dos testes foram consistentes com as operações manuais de cifragem e decifragem.

Em conclusão, este trabalho demonstrou que é possível implementar o Vernam Cipher em Verilog de forma eficaz e segura. Além disso, os conceitos e técnicas aqui utilizados podem ser aplicados em outras áreas da engenharia de hardware, como o design de sistemas digitais e na implementação de outros algoritmos de criptografia.