Universidade Federal de Minas Gerais

Documentação do Trabalho Prático - Introdução aos Sistemas Lógicos

Wesley Marques Daniel Chaves

Introdução

O trabalho prático foi dividido em duas partes, uma com a implementação de um flip-flop tipo D e outra com o problema de cifrar e decifrar uma mensagem utilizando one-time pad. Ambas as partes foram feitas exclusivamente na linguagem de descrição de hardware, Verilog.

Em todas as imagens apresentadas no relatório, os comentários descritivos foram retirados para diminuir a quantidade de informações, porém, os códigos enviados estão devidamente comentados.

Flip-Flop

A implementação foi feita com base em um resettable e settable flip-flop, pois além de possuir os inputs e outputs comuns de um flip-flop tipo D, possuem inputs para tornar o output do flip-flop 1 (set) ou 0 (reset).

```
module d_flip_flop (
  input d,
  input clk,
  input reset,
  input set,
  output q1,
  output q2
);

  reg reg_q1;

  always @(posedge clk or posedge reset or posedge set) begin
    if (reset) begin
      reg_q1 <= 1'b0;
    end else if (set) begin
      reg_q1 <= 1'b1;
    end else begin
      reg_q1 <= d;
    end
  end

  assign q1 = reg_q1;
  assign q2 = ~reg_q1;

endmodule
```

Como pode-se ver na imagem acima, a modelagem foi feita tendo como base a borda ascendente do clock (clk), ocorrendo a transferência do bit armazenado no input data (d) apenas quando o clock alterar o seu valor de 0 para 1. Além disso, cabe observar que o flip-flop construído é assíncrono com relação aos inputs set e reset, pois a alteração de seu output ocorre no mesmo instante que um desses inputs se alteram de 0 para 1, independente do clock.

O testbench do módulo foi feito para se testar as variações dos inputs. Temos no total três casos de testes, um com tanto reset quanto set iguais a zero, outro com set igual a 1 e reset igual a zero, e outro com set igual a zero e reset igual a 1. Desse modo, é possível observar com o diagrama de tempo disponibilizado o comportamento do módulo, e verificar que o módulo se comporta como um flip flop tipo D.

One-Time Pad (OTP)

O problema de cifragem e decifragem de uma mensagem foi resolvido criando um módulo na qual encapsula todas as operações necessárias para sua realização, isto é, a geração de uma chave pseudo-aleatória, a cifragem e a decifragem da mensagem.

O módulo possui, além dos inputs para receber a mensagem a ser cifrada e o clock, um input de sinal, que diz se o módulo irá realizar a cifragem (sinal = 1) ou se irá realizar a decifragem (sinal = 0) na borda ascendente do clock. Um ponto importante: a decifragem é sempre com base na mensagem imediatamente anterior, ou seja, se o módulo realizou a cifragem da mensagem **A** e logo depois realizou a cifragem da mensagem **B**, o módulo irá realizar a decifragem para obter novamente a mensagem **B,** caso o input de sinal seja configurado como zero neste momento.

A lógica utilizada para realizar a cifragem e a decifragem foi feita considerando a mensagem como streams de bits, ou seja, a operação XOR entre a mensagem e a chave é realizada bit a bit. Para sua implementação, foi utilizado um for loop no qual a cada laço, o bit mais significativo da sequência é computado com o bit correspondente da chave, e em seguida, o resultado é concatenado à direita com uma variável que armazena os resultados já computados dos bits mais significativos anteriores, armazenando toda a mensagem cifrada após todos os bits forem computados. Veja o seguinte exemplo:

mensagem: **10101000**
chave: **1100**

1) realização bit a bit da operação XOR entre o primeiro bit mais significativo da mensagem e da chave, repetindo a operação até ao quarto bit, tendo o seguinte resultado:

(1010) XOR (1100)

2) o quinto bit da mensagem irá ser operado com o primeiro bit mais significativo da chave, o sexto com o segundo bit, etc; finalizando a cifragem após todos os bits da mensagem serem computados, gerando um outro bloco de bits:

(1000) XOR (1100)

Veja que obteríamos o mesmo resultado se tivéssemos duplicado a chave, obtendo uma chave de 8 bits e realizando a operação:

(10101000 XOR 11001100)

Além disso, note que o algoritmo se comporta de maneira correta quando a quantidade de bits da mensagem é maior que a quantidade de bits da chave, porém, não é um múltiplo da mesma. Por exemplo, se a mensagem possui 9 bits, teremos que duplicar a chave duas vezes e realizar a última operação entre o último bit da mensagem e o bit mais significativo da chave. Como as operações são realizadas bit a bit, a interrupção do algoritmo em tal ponto não tem nenhum efeito

negativo, diferentemente de outras implementações que utilizam blocos fixos, tendo que realizar operações adicionais para lidar com tais casos.

```
module reg_otp #(parameter DATA_WIDTH = 8)(
  input clk,
  input sinal,
  input [DATA_WIDTH-1:0] mensagem,
  output [DATA_WIDTH-1:0] cifragem,
  output [DATA_WIDTH-1:0] decifragem,
  output [7:0] chave
  );

  reg [DATA_WIDTH-1:0] reg_cifragem;
  reg [DATA_WIDTH-1:0] reg_decifragem;
  reg [DATA_WIDTH-1:0] temp;
  reg [7:0] reg_chave;
  reg reg_bit;
  integer i;
  integer j;

  always @(posedge clk) begin
    j = 7;
    if(sinal) begin
        reg_chave = $random;
      for (i = DATA_WIDTH; i > 0; i = i - 1) begin
         reg_bit = mensagem[i-1 -: 1];
         temp[i-1 -: 1] = reg_bit ^ reg_chave[j];
         reg_cifragem = {reg_cifragem, temp};

         j = j - 1;
         if (j < 0) begin
            j = 7;
         end
       end
    end else begin
      for (i = DATA_WIDTH; i > 0; i = i - 1) begin
         reg_bit = reg_cifragem[i-1 -: 1];
         temp[i-1 -: 1] = reg_bit ^ reg_chave[j];
         reg_decifragem = {reg_decifragem, temp};

         j = j - 1;
         if (j < 0) begin
            j = 7;
         end
      end
    end
```

```
44    end
45
46    assign cifragem = reg_cifragem;
47    assign decifragem = reg_decifragem;
48    assign chave = reg_chave;
49  endmodule
```

Como já discutido, quando o sinal recebe o valor 1, ao ocorrer a borda ascendente do clock é gerado uma chave pseudo-aleatória e as operações bit a bit são realizadas. Para o controle do bit correspondente da chave, foi utilizado uma variável adicional **j** que diminui seu valor de 7 a 0. Ou seja, os bits da chave são selecionados periodicamente a cada intervalo de 8 bits da mensagem original.

Caso o sinal receba o valor 0, é executado o bloco dentro da instrução *else*, que possui a mesma lógica anterior, porém, ao invés de operar sobre a mensagem original, opera sobre reg_cifragem, que é um registrador contendo a mensagem cifrada.

Por fim, note que a cada operação de cifragem, o módulo gera outra chave, e a mensagem armazenada em reg_cifragem passa a ser a cifragem da nova mensagem, desse modo, a decifragem é sempre com base na mensagem imediatamente anterior, pois os valores das mensagens anteriores já foram perdidos, como também a chave necessária para sua decifragem.

O testbench do módulo possui quatro mensagens de 64 bits que devem ser cifradas e decifradas, porém, caso seja necessário alterar a quantidade de bits, basta alterar o parâmetro **DATA_WIDTH** no início do testbench para a quantidade desejada e alterar a quantidade de bits das mensagens em cada teste. Cada caso de teste é apresentado da seguinte forma no terminal da plataforma:

```
NOVO TESTE
Mensagem: 1100101010111101101011111001010011011101001110010010111011010011
Chave OTP: 01100011
RESULTADOS
Cifragem: 1010100111011110110011001111011110111110010110100100110110110000
Decifragem: 1100101010111101101011111001010011011101001110010010111011010011
```

Desse modo, é possível ver a mensagem original, a chave OTP utilizada e os resultados.

OBS: A chave OTP está sendo retornada como output apenas para fins de correção, porém, numa implementação sem esse fim, a chave iria se perder logo após sua utilização.

Execução

Para a execução do código junto com o testbench, utilize a plataforma online EDA Playground e faça o upload em seus respectivos locais dos arquivos design.sv e testbench.sv disponibilizados no arquivo .zip entregue. Cabe as seguintes configurações na plataforma:

Testbench + Design: SystemVerilog/Verilog
Tools & Simulators: Free Simulators - Icarus Verilog 0.9.7 ou 0.9.6