Introdução aos Sistemas Lógicos - TW
Trabalho Prático em Verilog

Flávio Marcilio

1. Em Verilog, implementar um flip-flop do tipo D. Você deve apresentar a especificação descritiva e comportamental, e testbench. Deve entregar o código, print screen do diagrama de tempo.

O Flip-Flop tipo D implementado neste trabalho segue o esquema apresentado na Figura 1.

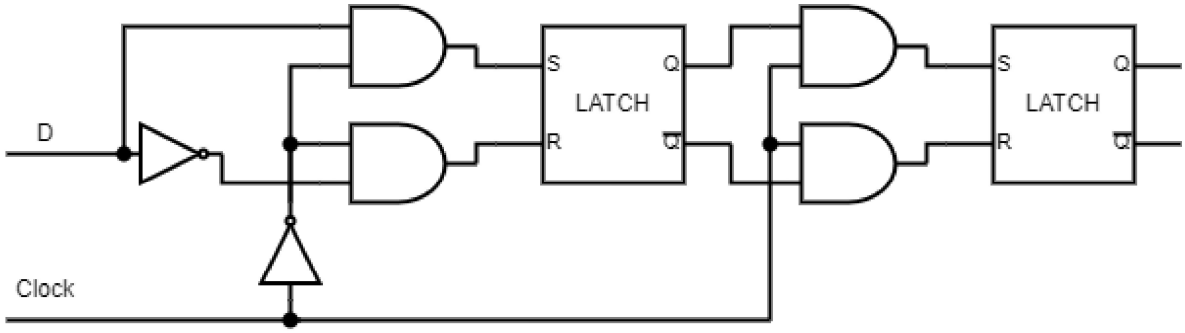

Figura 1 - Esquema do Flip-Flop tipo D disparado por borda ascendente

Para implementar este Flip-Flop foi utilizado um Latch desenvolvido com portas NOR, como apresentado na Figura 2.

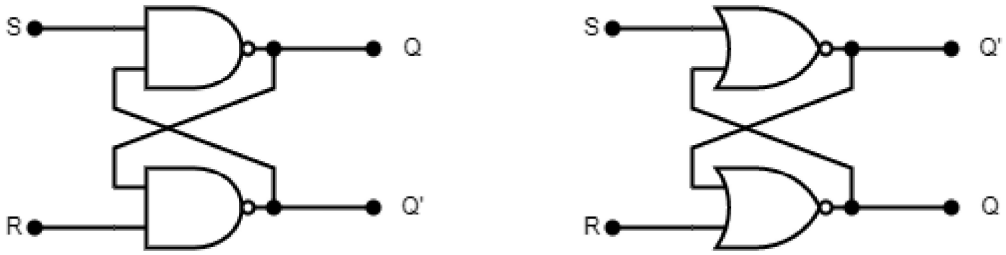

Figura 2 - Latch S-R implementado com portas NAND e NOR

O código Verilog do Latch S-R com portas NOR é apresentado abaixo:

```
module Latch_SR(Q, Qbar, S, R);
  input S, R;
  output Q, Qbar;

  nor nor1(Q, R, Qbar);
  nor nor2(Qbar, S, Q);
endmodule
```

O testbench utilizado para validar a implementação é apresentado abaixo:

```
module testbench;
  reg s, r;
  wire q, qbar;
  Latch_SR latch(q, qbar, s, r);
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1,testbench);
     $monitor($time," s = %b, r = %b, q = %b, qbar = %b",s,r,q,qbar);

      s=1'b0;
      r=1'b0;
      #1
      s=1'b1;
      r=1'b1;
      #1
      s=1'b1;
      r=1'b0;
      #1
      s=1'b0;
      r=1'b1;
      #1
      s=1'b1;
      r=1'b1;
      #1
      s=1'b0;
      r=1'b0;
      #1 $finish;
    end
endmodule
```

O diagrama de tempo para o Latch_SR implementado é mostrado na Figura 3.

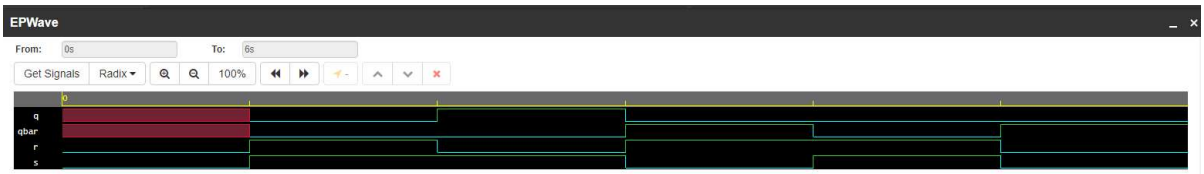

Figura 3 - Diagrama de tempo para o Latch_SR NOR

A implementação em Verilog do Flip-Flop tipo D seguindo a descrição estrutural é apresentada no código abaixo, conforme modelo de Katz em Contemporary Logic Design[1]:

```
// Implementação Estrutural do Flip-Flop tipo D
// utilizando o modelo de Katz (p. 271)
module FlipFlopD(clk, D, Q);
  input clk, D;
  output Q;
  wire nD, nC, R_M, S_M, R_S, S_S, Q_M, Qb_M, Q_S, Qb_S;

  not notD(nD, D);
  not notC(nC, clk);

  and and_M1(S_M, D, nC);
  and and_M2(R_M, nD, nC);
  Latch_SR Master(Q_M, Qb_M, S_M, R_M);

  and and_S1(S_S, Q_M, clk);
  and and_S2(R_S, Qb_M, clk);
  Latch_SR Slave(Q_S, Qb_S, S_S, R_S);

  assign Q = Q_S;

endmodule
```

Utilizando a descrição comportamental, o Flip-Flop tipo D pode ser implementado em Verilog como:

```
module DFlipFlop(clk, D, Q);
  input clk, D;
  output reg Q;
  always @(posedge clk)
    begin
      Q = D;
    end
endmodule
```

O testbench utilizado para validar a implementação foi definido como:

```
module testbench;
  reg clk, D;
  wire Q;
  FlipFlopD DFF(clk,D,Q);
  initial
    begin
      clk = 0;
      forever #1 clk=~clk;
    end
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1,testbench);
      $monitor($time," clk = %b, D = %b, Q = %b",clk, D, Q);
      D=0;
      #2 D=1;
```

```
        #2 D=0;
        #2 D=1;
        #2 D=0;
        #2 D=0;
        #2 D=1;
        #2 D=0;
        #4 $finish;
    end
endmodule
```

O diagrama de tempo de validação do Flip-Flop tipo D é mostrado na Figura 4.

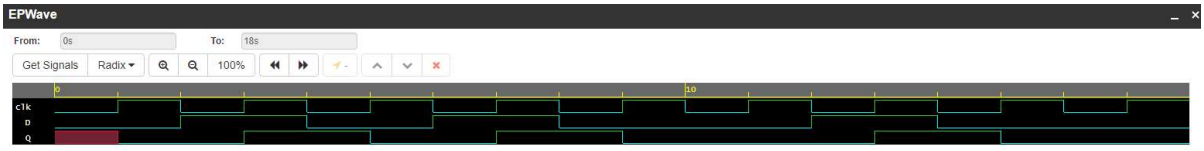

Figura 4 - Diagrama de tempo para o Flip-Flop tipo D

2. Em Verilog, implementar registradores e stream cypher. Assim necessário:
    a. Montagem dos registradores contendo One-Time Pad (OTP) e mensagem a ser cifrada.
    b. Operação XOR para cifragem de mensagens.
    c. Decifragem da mensagem.
    d. Entregar código, test bench e resultado.

Para desenvolver o Stream Cypher será implementado um registrador de deslocamento de 8 bits, conforme apresentado na Figura 5, para armazenar o OTP. Este registrador segue o modelo do registrador universal apresentado por Roth e Kinney em Fundamentals of Logic Design[2].

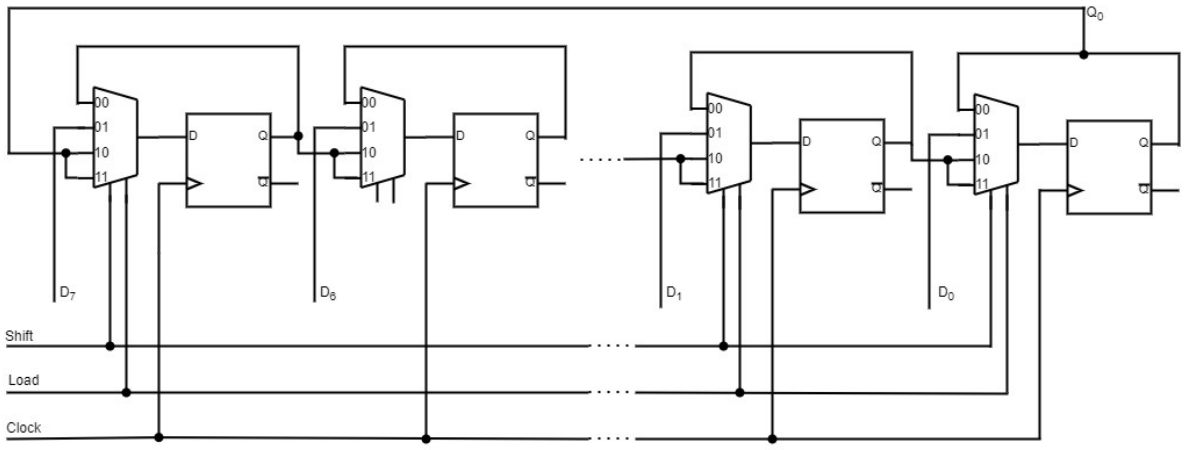

Figura 5 - Registrador de Deslocamento de 8 bits para OTP

O OTP será composto de 8 bits, portanto, para cifrar uma mensagem maior que 8 bits será necessário passar pelo OTP mais de uma vez. A cifragem e a decifragem da mensagem serão realizadas bit-a-bit.

Para implementar o registrador apresentado na Figura 5, foi utilizado multiplexadores 4 para 1, com o código apresentado abaixo:

```
module mux_4to1(in00, in01, in10, in11, Sh, L, out);
  input in00, in01, in10, in11, Sh, L;
  output out;

  assign out = ((~Sh & ~L) & in00) | ((~Sh & L) & in01) | ((Sh & ~L)
& in10) | ((Sh & L) & in11);

endmodule
```

O código de implementação do registrador:

```
// Registrador de Deslocamento conforme modelo mostrado
// no livro Fundamentals of Logic Design, 7ª Edição, pg.382
module   ShiftRegister_8bit(data,   sh,   load,   clk,   serial_out,
parallel_out);
  input [7:0]data;
  input sh;
  input load;
  input clk;
  output serial_out;
  output [7:0]parallel_out;

  // Resultado dos multiplexadores
  wire wire_m7;
  wire wire_m6;
  wire wire_m5;
  wire wire_m4;
  wire wire_m3;
  wire wire_m2;
  wire wire_m1;
  wire wire_m0;

  // Resultado dos Flip-Flop
  wire wire_dff7;
  wire wire_dff6;
  wire wire_dff5;
  wire wire_dff4;
  wire wire_dff3;
  wire wire_dff2;
  wire wire_dff1;
  wire wire_dff0;

  // Instanciando os multiplexadores
  mux_4to1 m7(.in00(wire_dff7), .in01(data[7]), .in10(wire_dff0),
```

```
                .in11(wire_dff0), .Sh(sh), .L(load), .out(wire_m7));

  mux_4to1 m6(.in00(wire_dff6), .in01(data[6]), .in10(wire_dff7),
                .in11(wire_dff7), .Sh(sh), .L(load), .out(wire_m6));

  mux_4to1 m5(.in00(wire_dff5), .in01(data[5]), .in10(wire_dff6),
                .in11(wire_dff6), .Sh(sh), .L(load), .out(wire_m5));

  mux_4to1 m4(.in00(wire_dff4), .in01(data[4]), .in10(wire_dff5),
                .in11(wire_dff5), .Sh(sh), .L(load), .out(wire_m4));

  mux_4to1 m3(.in00(wire_dff3), .in01(data[3]), .in10(wire_dff4),
                .in11(wire_dff4), .Sh(sh), .L(load), .out(wire_m3));

  mux_4to1 m2(.in00(wire_dff2), .in01(data[2]), .in10(wire_dff3),
                .in11(wire_dff3), .Sh(sh), .L(load), .out(wire_m2));

  mux_4to1 m1(.in00(wire_dff1), .in01(data[1]), .in10(wire_dff2),
                .in11(wire_dff2), .Sh(sh), .L(load), .out(wire_m1));

  mux_4to1 m0(.in00(wire_dff0), .in01(data[0]), .in10(wire_dff1),
                .in11(wire_dff1), .Sh(sh), .L(load), .out(wire_m0));

  // Instanciando os 8 Flip-Flops
  FlipFlopD dff7(.clk(clk), .D(wire_m7), .Q(wire_dff7));
  FlipFlopD dff6(.clk(clk), .D(wire_m6), .Q(wire_dff6));
  FlipFlopD dff5(.clk(clk), .D(wire_m5), .Q(wire_dff5));
  FlipFlopD dff4(.clk(clk), .D(wire_m4), .Q(wire_dff4));
  FlipFlopD dff3(.clk(clk), .D(wire_m3), .Q(wire_dff3));
  FlipFlopD dff2(.clk(clk), .D(wire_m2), .Q(wire_dff2));
  FlipFlopD dff1(.clk(clk), .D(wire_m1), .Q(wire_dff1));
  FlipFlopD dff0(.clk(clk), .D(wire_m0), .Q(wire_dff0));

  // a saída vai ser o conteúdo do último flip-flop
  assign serial_out = wire_dff0;

  // parallel out
  assign parallel_out[7] = wire_dff7;
  assign parallel_out[6] = wire_dff6;
  assign parallel_out[5] = wire_dff5;
  assign parallel_out[4] = wire_dff4;
  assign parallel_out[3] = wire_dff3;
  assign parallel_out[2] = wire_dff2;
  assign parallel_out[1] = wire_dff1;
  assign parallel_out[0] = wire_dff0;

endmodule
```

O testbench implementado é apresentado abaixo:

```
module ShiftRegister_8bit_test;
  reg [287:0]msg; // Mensagem a ser cifrada
  reg [287:0]result; // Resultado da cifragem e decifragem
```

```
  reg [7:0]otp; // One-Time-Pad
  reg sh_enc;
  reg sh_dec;
  reg load;
  reg clk;
  wire serial_out_enc;
  wire serial_out_dec;

  integer i;

  // Instanciando o Registrador para cifrar a mensagem
     ShiftRegister_8bit  enc(.data(otp),  .sh(sh_enc),  .load(load),
.clk(clk),
                         .serial_out(serial_out_enc));

  // Instanciando o Registrador para decifrar a mensagem
     ShiftRegister_8bit  dec(.data(otp),  .sh(sh_enc),  .load(load),
.clk(clk),
                         .serial_out(serial_out_dec));

  initial
    begin
        clk = 1'b0;
      forever #10 clk = ~clk;
     end

  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1, ShiftRegister_8bit_test);

      // msg <= Universidade Federal de Minas Gerais

                                                          msg         <=
288'b010101010110111001101001011101100110010101110010011100110110100
101100100011000010110010001100101001000000100011001100101011001000110
001010111001001100001011011000010000001100100011001010010000001001100
101101001011011100110000101110011001000000100011101100101011100100110
000010110100101110011;
      otp <= 8'b00101010;
      sh_enc <= 0;
      sh_dec <= 0;
      load <= 1;

      #10
      $display("Mensagem Original: %36s", msg);
      $display("One-Time-Pad     : %b", otp);
      $display("\nCifrando...");
      sh_enc <= 1;
      load <= 0;

      for (i = 0; i < 288; i = i + 1) begin
        #20
        result[i] <= msg[i] ^ serial_out_enc;
      end
```

```
        $display("Mensagem Cifrada: %36s", result);


        $display("\nDecifrando...");
        sh_dec <= 1;

        for (i = 0; i < 288; i = i + 1) begin
          #20
          result[i] <= result[i] ^ serial_out_dec;
        end

        $display("Mensagem Decifrada: %36s", result);

        #10
        $finish;
    end
endmodule
```

O resultado dos testes são apresentados abaixo:

```
Mensagem Original: Universidade Federal de Minas Gerais
One-Time-Pad     : 00101010

Cifrando...
Mensagem Cifrada:  DC\OXYCNKNO
lONOXKF
NO
gCDKY
mOXKCY

Decifrando...
Mensagem Decifrada: Universidade Federal de Minas Gerais
```

Referências

[1] Katz, Randy H., Borriello, Gaetano. Contemporary Logic design (2nd ed.). Upper Saddle River, NJ: Prentice Hall, 2004

[2] Roth Jr., Charles H., Kinney, Larry L. Fundamentals of Logic Design (7th ed.). Stanford: Cengage Learning, 2014