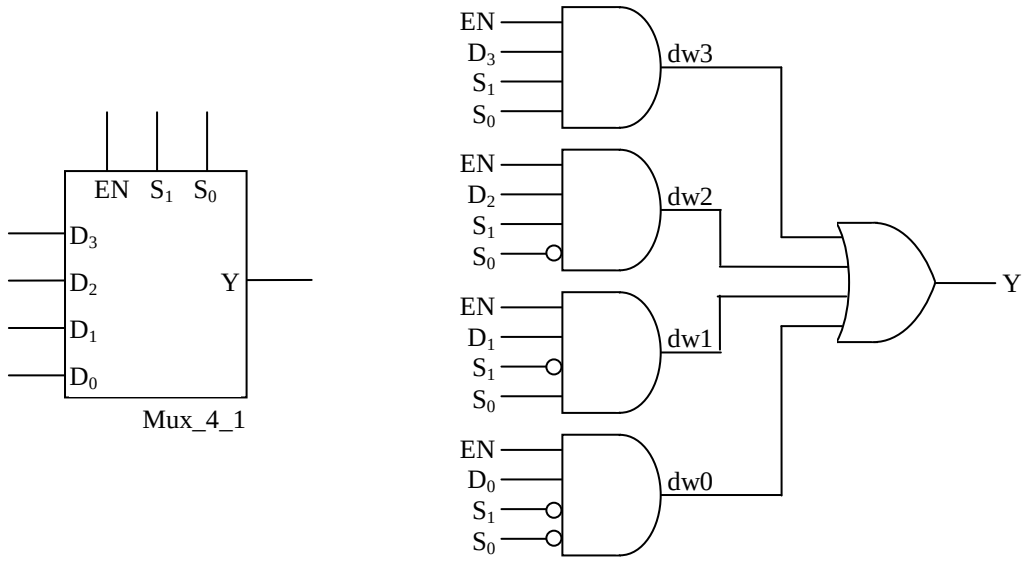
ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos

Verilog – Exemplos Básicos (versão 2)

Prof. Dr. Celso Massatoshi Furukawa

1 Multiplexador 4:1 com enable

Figura 1 Multiplexador Mux_4_1



1.1 Descrição não procedural (atribuições *assign*)

```
module mux_4_1_assign (
    input [3:0] D,   // Por default, inputs e outputs são wires
    input [1:0] S,
    input EN,
    output Y         // não tem virgula após o último parâmetro!
    );               // mas tem ';' após fechar o parêntesis!

    wire [3:0] dw;   // nets internas (auxiliares)

    assign dw[0] = EN & D[0] & ~S[1] & ~S[0];
    assign dw[1] = EN & D[1] & ~S[1] &  S[0];
    assign dw[2] = EN & D[2] &  S[1] & ~S[0];
    assign dw[3] = EN & D[3] &  S[1] &  S[0];

    assign Y = dw[0] | dw[1] | dw[2] | dw[3]; // Y é uma net do tipo wire

endmodule           // não tem ';' após endmodule!
```

1.2 Descrição procedural usando *for*

```
module mux_4_1_for (
    input [3:0] D,
    input [1:0] S,
    input EN,
    output Y
    );

    reg di, yReg;
    integer i;

    always @ (*) begin
        yReg = 0;
        for (i = 0 ; i <= 3; i = i + 1) begin
            // 'i == S' resulta em 1 quando verdadeiro
            di = EN & D[i] & ( i[1:0] == S[1:0] );
            yReg = yReg | di;
        end // for i
    end // always

    assign Y = yReg;

endmodule
```

1.3 Descrição por bloco procedural usando índice

```
module mux_4_1_index (
    input [3:0] D,
    input [1:0] S,
    input EN,
    output Y    // Por default, inputs e outputs são wires
    );

    reg yReg;     // variável auxiliar para o bloco always

    always @ (*) begin    // '*' representa “todas as entradas”
        yReg = EN & D[ S ]; // 'regs' podem ser modificadas no bloco always
    end // always

    assign Y = yReg;  // ’wires’ podem ser modificadas fora do bloco always

endmodule
```

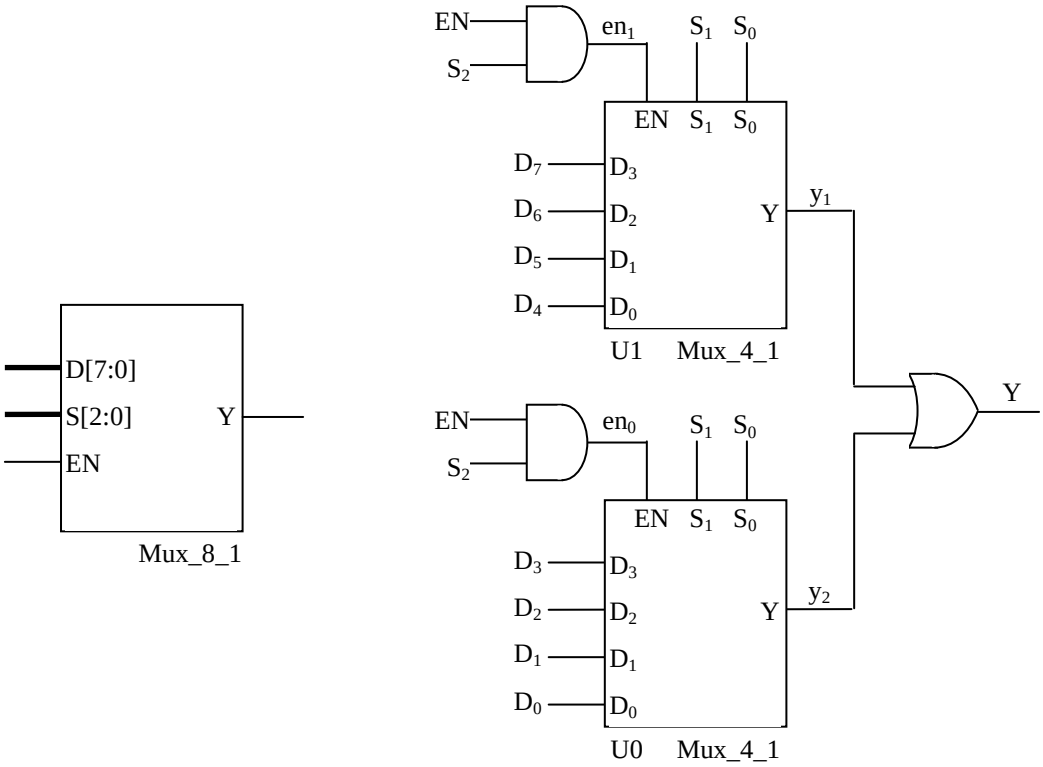
1.4 Descrição procedural usando *if-then-else* e *case*

```
module mux_4_1_case (
    input [3:0] D,
    input [1:0] S,
    input EN,
    output Y
    );

    reg yReg;

    always @ (*) begin
        if (EN == 0) begin
            yReg = 0;
        end // if
        else begin
            case (S)
                2'b00 : yReg = D[0]; // "2'b00" 2 bits, 0 em binário
                2'b01 : yReg = D[1]; // "2'b01" 2 bits, 1 em binário
                2'b10 : yReg = D[2]; // "2'b10" 2 bits, 2 em binário
                2'b11 : yReg = D[3]; // "2'b11" 2 bits, 3 em binário
                default: yReg = 0;   // recomendado
            endcase;
        end // else
    end // always

    assign Y = yReg;

endmodule
```

2 Instanciação de um módulo em outro

Exemplo: multiplexador 8:1 com *enable* usando o módulo mux_4_1_assign

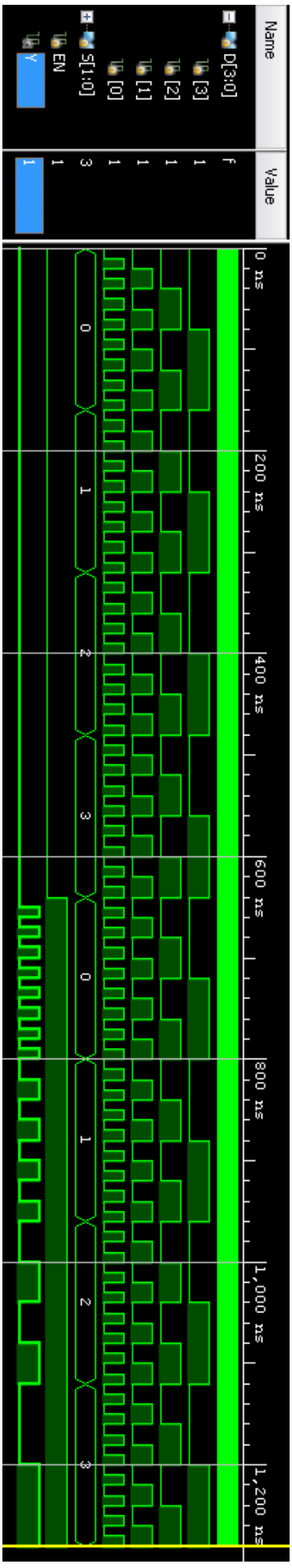
Figura 2 Multiplexador 8:1 cascateando multiplexadores 4:1



```
module mux_8_1_assign (
    input [7:0] D,
    input [2:0] S,
    input EN,
    output Y
    );

    wire y0, y1, en0, en1;  // nets internas (auxiliares)

    // instanciando os multiplexadores 4:1
    mux_4_1_assign U0 (D[3:0], S[1:0], en0, y0);
    mux_4_1_assign U1 (D[7:4], S[1:0], en1, y1);

    // conexões entre os módulos
    assign en0 = EN & ~S[2];
    assign en1 = EN &  S[2];
    assign Y = y0 | y1;

endmodule
```

3 Simulação (bloco *initial*): varredura de todas as possibilidades

```
`timescale 1ns / 1ps   // unidade de tempo / precisão da simulação

module mux4_1_sim(
    );

    reg [3:0] D;
    reg [1:0] S;
    reg EN;
    wire Y;
    integer i;

    mux_4_1_assing UUT (D, S, EN, Y);

    initial begin
    $dumpfile("dump.vcd");  // arquivo para salvar sinais da simulação
    $dumpvars(0);           // define quais sinail salvar. 0: todos
        $timeformat(-9, 0, "ns", 3);  // Tempo em ns (10E-9)
        // i varre todas as combinações de 7 variáveis
        for (i = 0; i < 128; i = i + 1) begin
            D[3:0] = i[3:0];
            S[1:0] = i[5:4];
            EN = i[6];
            #10;  // avança uma unidade tempo definida por timescale
            $write( "%t: EN=%b, S=%b, D=%b => Y=%b\n", $time, EN, S, D, Y);
        end // for i
        $finish;
    end // initial

endmodule
```


Name
Value
D[3:0]
[3]
[2]
[1]
[0]
S[1:0]
EN
Y
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns

4 Simulação (bloco *initial*): vetor de teste

```
`timescale 1ns / 1ps   // unidade de tempo / precisão da simulação

module mux4_1_sim(
    );

    reg [3:0] D;
    reg [1:0] S;
    reg EN;
    wire Y;

    mux_4_1_assing UUT (D, S, EN, Y);

    initial begin
        $dumpfile("dump.vcd");  // arquivo para salvar sinais da simulação
        $dumpvars(0);           // define quais sinais salvar. 0: todos
        // Testa apenas algumas combinações de entrada
        // Definição do vetor de teste e impressão da saída
        EN = 1; S = 2'b00;  D = 4'b0000; #10;
        $write( "En=%b, S=%b, D=%b => Y=%b\n", EN, S, D, Y);
        D = 4'b0001; #10;
        $write( "En=%b, S=%b, D=%b => Y=%b\n", EN, S, D, Y);
        EN = 0; #10;
        $write( "En=%b, S=%b, D=%b => Y=%b\n", EN, S, D, Y);
        $finish;
    end // initial

endmodule
```

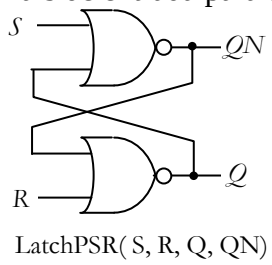
Saídas do simulador:

```
En=1, S=00, D=0000 => Y=0
En=1, S=00, D=0001 => Y=1
En=0, S=00, D=0001 => Y=0
```

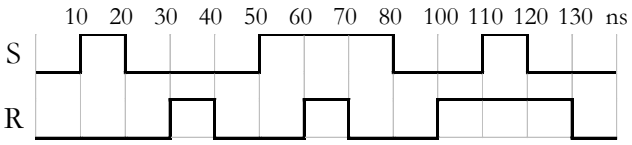
5 Simulação de circuitos sequenciais

5.1 Simulação sem sinal de clock

Exemplo: sinais de entrada para um latch SR positivo



LatchPSR(S, R, Q, QN)



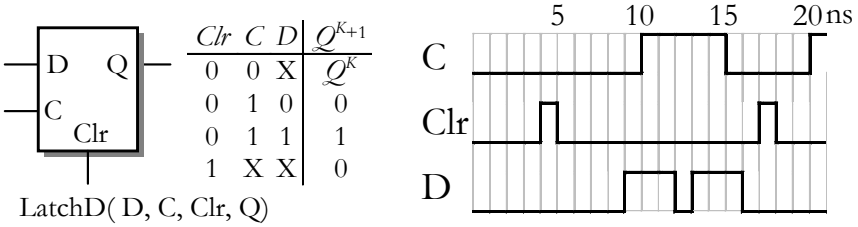

```
`timescale 1ns / 1ps   // unidade de tempo / precisão da simulação

module LatchPSR_sim( );

    reg S, R;
    wire Q, QN;
    integer t;  // t é o tempo da simulação

    LatchPSR UUT (S, R, Q, QN);  // Instanciação do latch SR

    initial begin
        $dumpfile("dump.vcd");  // Nome do arquivo para salvar os sinais
        $dumpvars(1);           // 1: salva apenas os sinais deste módulo
        $timeformat(-9, 0, "ns", 3);  // Tempo em ns (10E-9)

        // Monitora os sinais (com exceção de $time) e imprime quando mudam
        $monitor( "Time %t: S=%b, R=%b => Q=%b  QN=%b", $time, S, R, Q, QN );

        for (t=0; t<=140; t=$time) begin
            #10;  // avança 10 unidade tempo definida por `timescale
        end; // for t

        $finish;
    end // initial

    always @ (*) begin
        S = 0; R = 0;   // Valores default
        // Edição do sinal S
        if ( (t>=10 && t<20) || (t>=50 && t<80) || (t>=110 && t<120) ) S = 1;
        // Edição do sinal R
        if ( (t>=30 && t<40) || (t>=60 && t<70) || (t>=100 && t<130) ) R = 1;
    end // always

endmodule
```

- Os blocos “initial” e “always” são executados **em paralelo**
- Bloco initial: faz o controle do tempo e copia no reg ***t*** o tempo de simulação
- Bloco always: altera os sinais S e R em função do tempo de simulação *t*.

Nota: $dumpvars(1) faz com que o simulador salve para plotagem apenas os sinais do primeiro nível de hierarquia – ou seja, apenas os sinais definidos no corrente módulo.

5.2 Simulação com sinal de clock

Exemplo: sinais para um latch D com entrada variando antes ou depois das bordas de clock




```
`timescale 1ns / 1ps   // unidade de tempo / precisão da simulação

module LatchD_SR_sim( );

    reg D, C, Clr;  // Entradas do Latch
    wire Q, QN;     // Saídas do Latch
    integer h, t;   // h: contador de meio-periodo;  t: tempo

    LatchD UUT (D, C, Clr, Q, QN);  // Instanciação do Latch D

    initial begin
        $dumpfile("dump.vcd");  // Nome do arquivo para salvar os sinais
        $dumpvars(0);           // 0: salva os sinais de TODOS os módulos
        $timeformat(-9, 0, "ns", 3);  // Tempo em ns (10E-9)

        // Monitora os sinais (com exceção de $time) e imprime quando mudam
        $monitor("Time %t: Clr=%b, D=%b => Q=%b QN=%b", $time, Clr, D, Q, QN);

        h = 0; C = 0; D = 0;        // Inicialização
        for (t=0; t<=21; t=$time) begin
            if ( t>=10 && h>=5 ) begin
                h = 0;
                C = ~C;       // após t=10, inverte C a cada meio-periodo
            end;  //if
            #1;               // Avança a simulação 1 unidade de `timescale
            h = h+1;
        end; //for t

        $finish;
    end // initial

    always @ (*) begin
        Clr <= 0; D <= 0;           // Valores default
        if ( (t>=4 && t<5) || (t>=17 && t<18) ) Clr <= 1; // Edição do sinal Clr
        if (t>=9 && t<16 && t!=12) D <= 1;                // Edita o sinal D
    end

endmodule
```

- Os blocos “initial” e “always” são executados **em paralelo**
- Bloco initial: faz o controle do tempo, gera o sinal de clock C e copia no reg ***t*** o tempo de simulação
- Bloco always: altera os sinais Clr e D em função do tempo de simulação.

Nota: $dumpvars, sem parâmetros, salva todos os sinais de todos os módulos instanciados na simulação. Posteriormente é possível escolher quais sinais devem plotados.