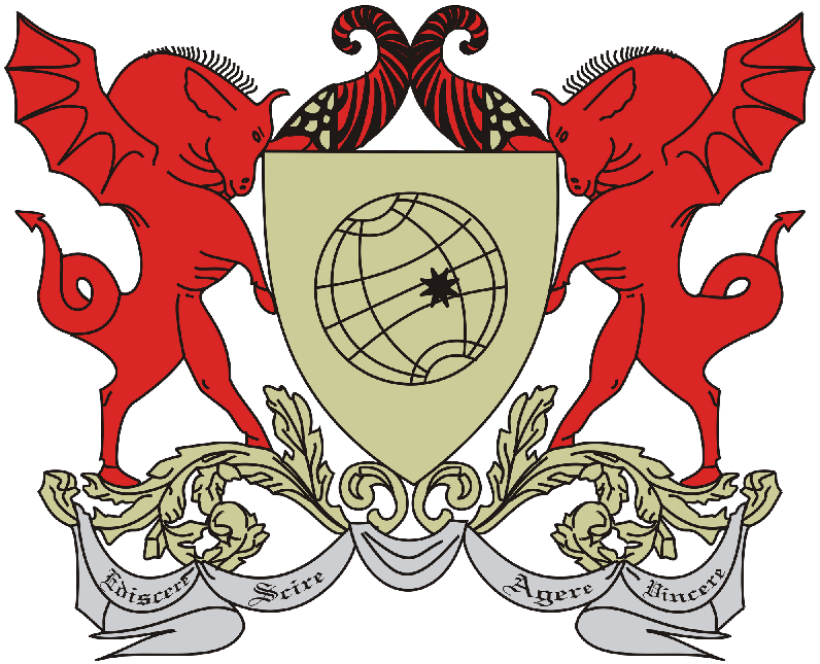


CCF 251 – Introdução aos Sistemas Lógicos

Aula Verilog – Parte 2

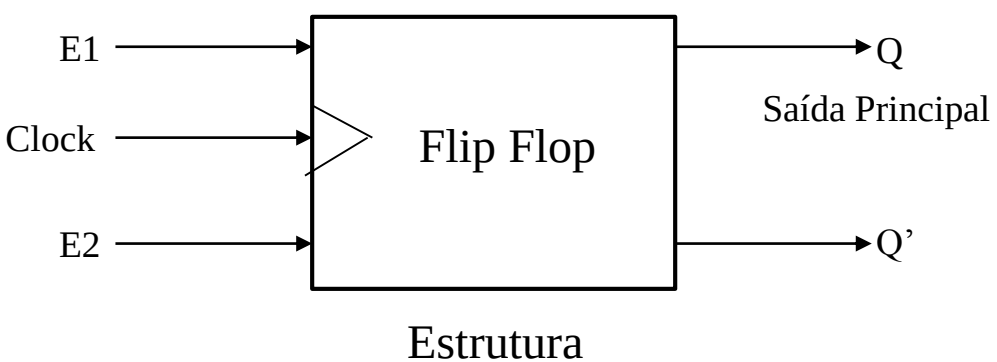
Prof. José Augusto Nacif – jnacif@ufv.br

Revisão

- Lógica Combinacional
  - Os componentes não armazenam valor;
  - As saídas não dependem das entradas;
  - Ex: Mux e ALU;
- Lógica Sequencial
  - Os componentes armazenam valor;
  - As saídas dependem tanto das entradas e dos estados internos;
  - Ex: Memória, banco de Registradores.

Flip Flops

- O que é um Flip Flop ?
  - É um circuito digital capaz de servir como uma memória de 1 bit;
  - É composto por duas entradas (E1 e E2), um sinal de controle (Clock) e duas saídas (Q e Q');
  - Possui basicamente dois estados:
    - Q = 0 e Q' = 1;
    - Q = 1 e Q' = 0;
  - Principais FFs:
    - RS (Reset - Set);
    - J-K;
    - D (Delay);
  - Podem ser utilizados em contadores, registradores, FSM e etc...

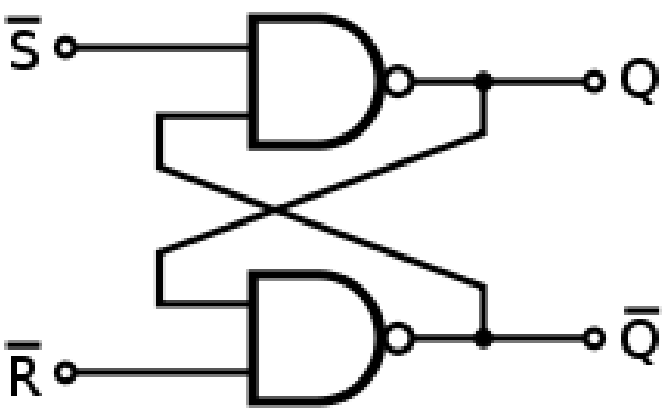


Estrutura

Flip Flop RS (Reset – Set)

- Este circuito irá mudar de estado apenas no instante em que mudam as variáveis de entrada (R e S).
  - A entrada S é denominada Set, pois quando acionada (nível 1), passa a saída para 1;
  - A entrada R é denominada Reset, pois quando acionada (nível 1), passa a saída para 0 (Zerando o flip-flop);

S	R	QAtual	Qfuturo
0	0	0	0
0	0	1	1
1	0	0	1
1	0	1	1
0	1	1	0
0	1	0	0
1	1	x	x



Circuito

Flip Flop RS (Reset – Set)

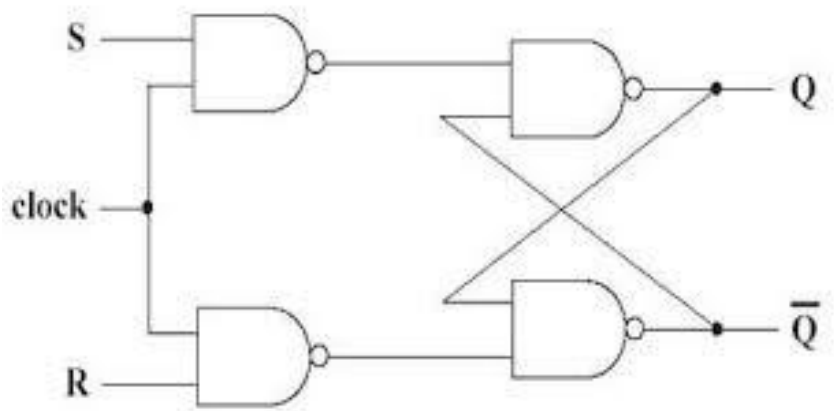
- **Verilog**

```
module ff_rs (s, r, q);
  input s, r;
  output q;
  reg q;
  always @(s or r)  begin
      if (s & r)  q = 0;
      else if (~s & r)  q = 0;
      else if (s & ~r) q = 1;
  end
endmodule
```

Flip Flop RS com entrada clock

- Quando a entrada de clock for igual a zero, o flip flop irá permanecer no seu estado mesmo que variem as entradas R e S.
- Quando a entrada de clock assumir o valor 1, o flip flop irá comportar-se como um flip flop RS básico;

CLK	QFuturo
0	QAtual
1	RS básico



Circuito

Flip Flop RS com entrada clock

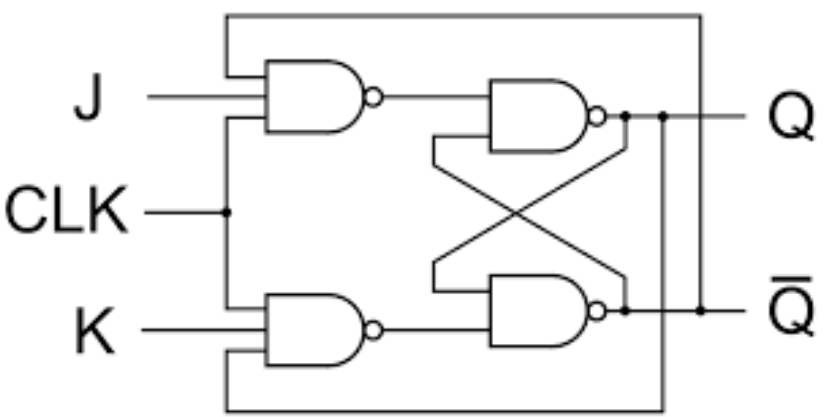
- **Verilog**

```
module ff_rs (clk, s, r, q);
  input clk, s, r;
  output q;
  reg q;
  always @(posedge clk)  begin
      if (s & r)  q = 0;
      else if (~s & r)  q = 0;
      else if (s & ~r) q = 1;
  end
endmodule
```

Flip Flop JK

- Aprimora o funcionamento do flip flop RS interpretando a condição R e S igual a 1;
  - J = 1 e K = 0 ativa o set;
  - J = 0 e K = 1 ativa o reset;
  - J = 1 e K = 1 Qfuturo = Q'Atual

J	K	Qfuturo
0	0	Qatual
0	1	0
1	0	1
1	1	Q'Atual



Circuito

Flip Flop JK

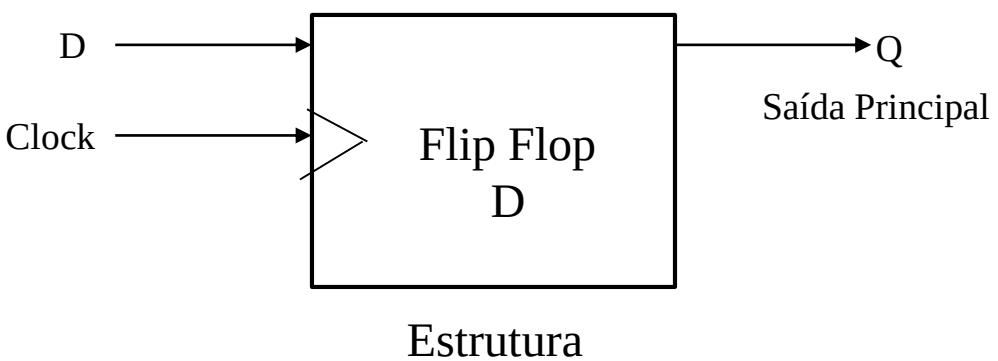
- **Verilog**

```
module jkff(J, K, clk, Q);
  input J, K, clk;
  output Q;
  reg Q;
  reg Qm;

  always @(posedge clk) begin
    if(J == 1 && K == 0)  Qm <= 1;
    else if(J == 0 && K == 1) Qm <= 0;
    else if(J == 1 && K == 1) Qm <= ~Qm;

    Q <= Qm;
   end
endmodule
```

Flip Flop D

- Possui apenas uma entrada que é ligado diretamente na saída;
- Quando ocorre a borda de subida de clock a saída recebe o valor da entrada;

Clk	D	Q
0	0	0
1	1	1
0	0	1
1	0	0



Estrutura

Flip Flop D

- **Verilog**

```
module ff_D (D, Clk, Q);
    input D, Clk;
    output Q;
    reg Q;

    always @Clk begin
       if (Clk)
       Q = D;
     end
endmodule
```

Registrador

- Conjunto de FFs D que são controlados pelo mesmo sinal de clock e reset. Por exemplo, Registrador de 8 bits.

```
module reg_8b
(
   input wire clk, reset,
   input wire [7:0] d ,
   output reg [7:0] q
);
// body
always@(posedge clk, posedge reset) begin
  If (reset)
       q <= 0 ;
   else
        q <= d;
  end
endmodule
```

Registrador

- **Registrador de deslocamento**

```
module reg_shift  #(parameter N=4) (a, clk, r_l);
  input clk;
  input r_l;
  output [N-1:0] a;
  reg [N-1:0] a;

  always@(negedge clk) begin
      a = (r_l) ? (a>>1'b1) : (a<<1'b1);
  end

endmodule
```

Exemplos

- **Contador Up / Down**
  - **Conta de 0 a N. N é um número de 4 bits;**
  - **Incrementado/Decrementado na borda de descida do clock;**
  - **Inputs**
    - **Clear**
      - **Reseta contador;**
    - **UP/Down**
      - **1: Contagem crescente;**
      - **0: Contagem decrescente;**

Exemplos

- **Contador Up**

```
module counter_up  #(parameter T=4) (a, clk, N);
  input clk;
  input [T-1:0] N;
  output [T-1:0] a;
  reg [T-1:0] a;

  always@(negedge clk) begin
      a = (a==N) ? 4'b0000 :  a + 1'b1;
  end

endmodule
```

Exemplos

- **Contador Down**

```
module counter_down  #(parameter T=4) (a, clk, N);
  input clk;
  input [T-1:0] N;
  output [T-1:0] a;
  reg [T-1:0] a;

  always@(negedge clk) begin
      a = (a==4'b0000) ? N :  a - 1'b1;
  end

endmodule
```

Exemplos

- **Contador UP/Down**

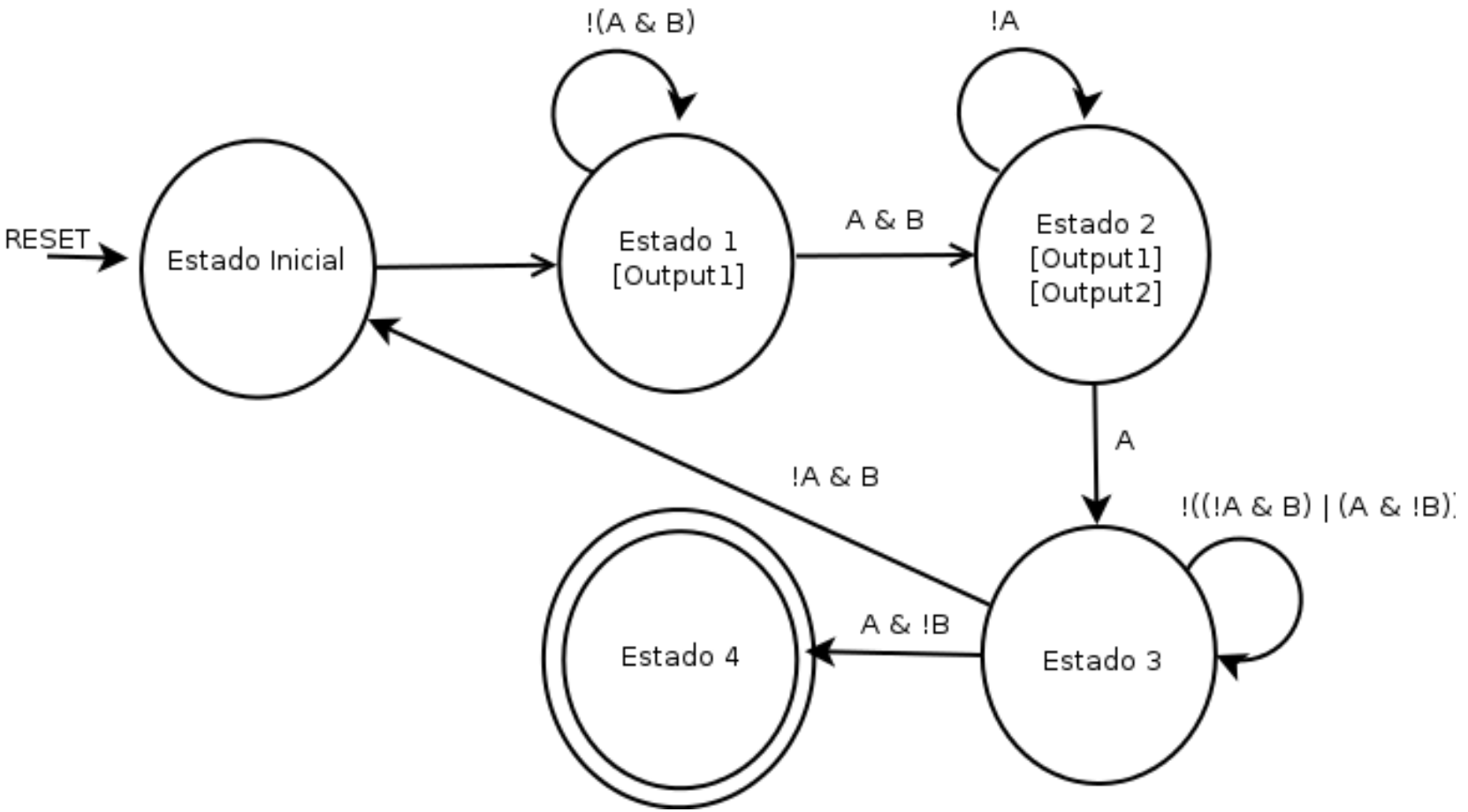
```
module counter_up_down  #(parameter T=4) (a, clk, N, up_down);
  input clk;
  input [T-1:0] N;
  output [T-1:0] a;
  reg [T-1:0] a;

  always@(negedge clk) begin
      a = (u_d) ? (= (a==N) ? 4'b0000 :  a + 1'b1) : ((a==4'b0000) ? N :  a - 1'b1);
  end

endmodule
```

Etapas Programação FSM

- **1°)** Definir entradas e saídas;
- **2°)** Definir e codificar os estados;
- **3°)** Definir transações entre estados;

Exemplo programação FSM

- Objetivo: Receber valor 1 nas saídas.



Exemplo programação FSM

- Definir entradas e saídas

```
 module FSM (
//Inputs
input clk, input reset,
input  A,  input B,

 // Outputs
output Output1,
output Output2,
output reg [2:0] Status
 );
```

```
assign Output1 = ( estado_atual = estado_1 ) | ( estado_atual == estado_2) ;
assign Output2 = ( estado_atual  == estado_2 ) ;
```

Exemplo programação FSM

- Declarar e codificar estados

```
localparam estado_inicial = 3 ’ d000,
                estado_1       = 3 ’ d001,
                estado_2       = 3 ’ d010,
                estado_3       = 3 ’ d011,
                estado_4       = 3 ’ d100;
```

Exemplo programação FSM

- Transações entre estados
  - É necessário criar dois regs para se armazenar o estado atual e o próximo estado.

```
reg [2:0] estado_atual;
reg [2:0] prox_estado;
```

Exemplo programação FSM

- Transações entre estados – Parte Sequencial

```
always@ ( posedge clk ) begin
        if ( reset )
           estado_atual  <= estado_inicial;
        else
           estado_atual <= prox_estado;
  end
```

Exemplo programação FSM

- Transações entre estados – Parte Combinacional

```
always@ ( * ) begin
    prox_estado  = estado_atual;
    case ( estado_atual )
      estado_incial : begin
                prox_estado = estado_1;
      end
      estado_1: begin
            if ( A & B) prox_estado = estado_2;
      end
      estado_2: begin
            if ( A ) prox_estado = estado_3;
      end
      estado_3 : begin
            if (! A & B )
                prox_estado = estado_inicial;
            else if ( A & ! B )
                prox_estado = estado_4;
      end
      estado_4: begin
            prox_estado = estado_inicial;
      end
    endcase
end
```

Exercícios

- Criar um contador em verilog para calcular a sequência de Fibonacci. Realizar a sequência até que o termo seja menor 9999, reiniciar quando o termo for maior. Se o sinal de reset for ativo a sequência deve ser reiniciada.

- Implmentar em verilog uma máquina de estados para verificar se o número é par ou ímpar.