UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Arquitetura e Organização de Computadores 1

Prática 1 Implementação de um testbench em Verilog

Professores: Luciano de Oliveira Neris e Mauricio Fernandes Figueiredo

Autores (Grupo R)

Guilherme Campos Marques, 727338 e Engenharia da Computação

Leticia Bossatto Marchezi, 791003 e Ciência da Computação

Marcos Cardoso Vendrame, 790725 e Ciência da Computação

Mateus Grota Nishimura Ferro, 771043 e Ciência da Computação

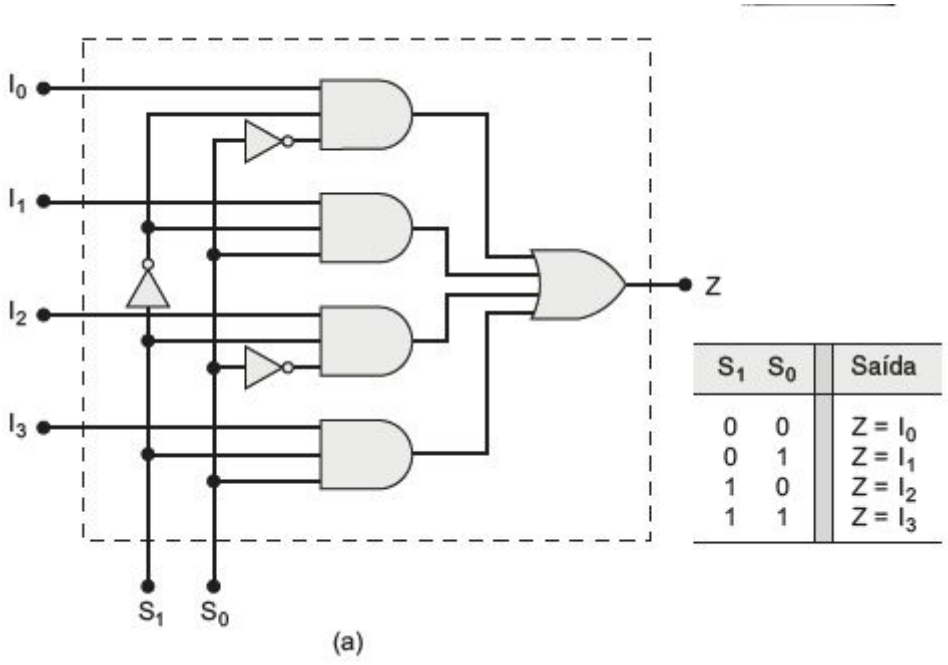
São Carlos, 11 de Março de 2021

1. Introdução

Utilizando a plataforma de aprendizado online EDA playground, foi designado ao grupo realizar a tarefa de gerar projetos Verilog, primeiramente foi gerado um projeto correspondente ao Multiplexador 4x1 (Parte A) e testado todas as possibilidades, adotando a simulação por varredura. Na segunda parte do trabalho, foi realizado um projeto com registradores, comparando atribuições blocantes e não-blocantes (Parte B).

2. Descrição da execução do experimento

Para a Parte A[1] utilizou-se do ambiente de simulação online EDA Playground para a implementação dos circuitos e dos testes. O primeiro circuito implementado foi o seguinte multiplexador 4x1:



S_1	S_0	Saída
0	0	$Z = I_0$
0	1	$Z = I_1$
1	0	$Z = I_2$
1	1	$Z = I_3$

Imagem 1: Multiplexador de 4 entradas usando soma de produtos.

Desta forma, a implementação do circuito na plataforma EDA Playground através da descrição procedural está representada na imagem abaixo.

```
// Módulo Multiplexador 4x1
module multiplex4x1(input x0, x1, x2, x3, [1:0] s,
                    output reg f);

  always @(x0,x1,x2,x3,s[1],s[0]) begin


    if(s == 2'b00) begin
      f = x0;
    end

    else if(s == 2'b01) begin
      f = x1;
    end

    else if(s == 2'b10) begin
      f = x2;
    end

    else if(s == 2'b11) begin
      f = x3;
    end
  end
endmodule
```

Imagem 2: Circuito "multiplex4x1" projetado no EDA Playground

Em seguida, na Parte B², foi implementado o design.sv do circuito com as variáveis a, b, c e d, usando atribuições blocantes e não-blocantes, assim como a lista de sensibilidade(borda de subida nas entradas clock e reset). Nesse projeto, os valores das saídas em T = 0s são a=1 e c=1, e nos intervalos de tempo posteriores ocorrem atribuições descritas pelas sentenças no Caso 1 e Caso 2. O módulo do caso 1 usa atribuição blocante(=), como descrito na imagem:

```
module caso1(b,d, clock, reset, a, c);

  output a,c;
  reg [3:0] a, c;
  input b, d, clock, reset;
  logic [3:0] b, d;


  always @(posedge clock, reset) begin

    if(reset)begin
      a = 1;
      c = 1;
    end

    else begin
      c = b + d;
      a = b + c;
    end

  end
endmodule
```

Imagem 3: Circuito "caso1" projetado no EDA Playground

Já no caso 2, a atribuição ocorre de forma não-blocante(<=), onde ambos processos ocorrem de forma simultânea. Ou seja, a ordem dos comandos não interfere no resultado.

```
module caso2(b,d, clock, reset, a, c);

  output a,c;
  reg [3:0] a, c;
  input b, d, clock, reset;
  logic [3:0] b, d;


  always @(posedge clock, reset) begin

    if(reset)begin
      a = 1;
      c = 1;
    end

    else begin
      c <= b + d;
      a <= b + c;
    end

  end
endmodule
```

Imagem 4: Circuito "caso2" projetado no EDA Playground

3. Apresentação dos resultados do experimento

A saída fornecida pelo multiplexador 4x1 da parte A descreve as entradas(x0, x1, x2, x3) e os dois bits de sinal de controle s0 e s1, que tem como saída “f”:

Parte A:

```
# KERNEL: 10ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 20ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 30ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 40ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 50ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 60ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 70ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 80ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 0
# KERNEL: 90ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 100ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 110ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 120ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 130ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 140ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 150ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 160ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 0 =>   f = 1
# KERNEL: 170ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 180ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 190ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 200ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 210ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 220ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 230ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 240ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 250ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 260ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 270ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 280ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 0
# KERNEL: 290ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 300ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 310ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 320ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 0 =>   f = 1
# KERNEL: 330ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 340ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 350ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 360ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 370ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 380ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 390ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 400ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 410ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 420ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 430ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 440ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 450ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 460ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 0
# KERNEL: 470ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 480ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 0,    s1 = 1 =>   f = 1
# KERNEL: 490ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 1 =>   f = 0
# KERNEL: 500ns: x0 = 0,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 1 =>   f = 1
# KERNEL: 510ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 1 =>   f = 0
# KERNEL: 520ns: x0 = 0,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 1 =>   f = 1
# KERNEL: 530ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 1 =>   f = 0
# KERNEL: 540ns: x0 = 0,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 1 =>   f = 1
```

```
# KERNEL: 550ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 1 =>    f = 0
# KERNEL: 560ns: x0 = 0,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 1 =>    f = 1
# KERNEL: 570ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 1 =>    f = 0
# KERNEL: 580ns: x0 = 1,    x1 = 0,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 1 =>    f = 1
# KERNEL: 590ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 1 =>    f = 0
# KERNEL: 600ns: x0 = 1,    x1 = 0,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 1 =>    f = 1
# KERNEL: 610ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 0,    s0 = 1,    s1 = 1 =>    f = 0
# KERNEL: 620ns: x0 = 1,    x1 = 1,    x2 = 0,    x3 = 1,    s0 = 1,    s1 = 1 =>    f = 1
# KERNEL: 630ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 0,    s0 = 1,    s1 = 1 =>    f = 0
# KERNEL: 640ns: x0 = 1,    x1 = 1,    x2 = 1,    x3 = 1,    s0 = 1,    s1 = 1 =>    f = 1
```

Imagem 5: Resultado da simulação do circuito "multiplex4x1" no site EDA Playground

A saída fornecida pelos circuitos de soma blocante e não-blocante em forma de tabela foi a seguinte:

Parte B:

```
# KERNEL:              CASO 1                |              CASO 2               |      Entradas
# KERNEL: t     a      b      c      d       |      a      b      c      d       |      a      c
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 10ns  1      2      1      2       |      1      2      1      2       |      3      4
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 20ns  6      2      4      2       |      3      2      4      2       |      6      4
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 30ns  6      2      4      2       |      6      2      4      2       |      6      4
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 40ns  6      2      4      2       |      6      2      4      2       |      6      4
```

Imagem 6: Resultado do testbench "case_tb" no site EDA Playground

Simulando a alteração das sentenças nos módulos[3], pode-se observar que em comparação ao resultado anterior, a saída 'a' no caso 1(blocante) apresenta resultado diferente:

```
# KERNEL:              CASO 1                |              CASO 2               |      Entradas
# KERNEL: t     a      b      c      d       |      a      b      c      d       |      a      c
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 10ns  1      2      1      2       |      1      2      1      2       |      3      4
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 20ns  3      2      4      2       |      3      2      4      2       |      6      4
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 30ns  6      2      4      2       |      6      2      4      2       |      6      4
# KERNEL: ------------------------------------------------------------------------------------------
# KERNEL: 40ns  6      2      4      2       |      6      2      4      2       |      6      4
```

Imagem 7: Alternando as sentenças de atribuição da parte B no site EDA Playground

4. Conclusão

A parte A do experimento consistiu na implementação de um multiplexador 4x1, que, como observado nos resultados, faz a seleção de qual entrada terá seu valor

expresso na saída, dependendo de um seletor (entrada s). Com o seletor recebendo o valor 00, é selecionada a entrada x0; com o valor 01, a entrada x1; com o valor 10, a entrada x2 e com valor 11, a entrada x3 é a selecionada.

A parte B da atividade apresentou a diferença entre as atribuições blocantes (=) e não blocantes (<=). Como foi observado, nas atribuições blocantes, a ordem dos comandos é importante para a obtenção do sinal, pois considera o valor já atualizado para o comando seguinte, enquanto que nas atribuições não-blocantes, a ordem não interfere nos resultados, pois são utilizados os valores anteriores (ainda não atualizados) nos comandos a serem executados.

Essa distinção foi observada na realização dos dois casos, em que o Caso 2, por ser não-blocante, precisou de um ciclo a mais de clock para atualizar o sinal “a” para 6 que no Caso 1, recebendo o valor 3 antes, que equivale a soma dos sinais “b” e “c” anteriores, enquanto que, no Caso 1, o sinal “c” já foi atualizado no comando anterior. Os resultados dos demais ciclos de clock se mantêm constantes por conta dos sinais “b” e “d” não serem alterados, assim, o sinal “c” é atualizado somente uma vez e o sinal “a” apenas uma vez no Caso 1 e duas vezes no Caso 2.

Ao inverter a ordem das sentenças, a atualização do sinal “c” no Caso 1 foi deixada para o final, não interferindo no resultado do sinal “a” no mesmo ciclo de clock, por isso foi observado a semelhança de resultados entre os dois casos, tendo em vista que ao realizar a soma b+c, o sinal “c” ainda apresenta o mesmo valor anterior, pois ainda não foi atualizado.

5. Referências Bibliográficas:

1.Projeto Parte A - https://www.edaplayground.com/x/u3Gn

2.Projeto Parte B - https://www.edaplayground.com/x/8fqK

3.Projeto Parte B com sentenças alternada Eda Playground - https://www.edaplayground.com/x/dTFN

Tocci, Ronald J. - Sistemas digitais : princípios e aplicações / Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss ; revisão técnica Renato Giacomini ; tradução Jorge Ritter. – 11. ed. – São Paulo : Pearson Prentice Hall, 2011.