Programa para Excelência em Microeletrônica

Desenvolvimento do Segundo Exercício de Systemverilog

Aluno: Gustavo Vilar de Farias

SONY make.believe
FUNDAÇÃO DE APOIO AO IFPB
FUNETEC-PB Fundação de Educação Tecnológica e Cultural da Paraíba
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PARAÍBA

Sumário

1.

Registrador.sv

```
module Registrador(input Clock, input[3:0] Entrada, output logic[3:0] Saida);
    always_ff @(posedge Clock)
        Saida <= Entrada;
endmodule
```

UpDownCount.sv

```
module UpDownCount(input Swap, Enable, Clock, output logic[3:0] DownCountS,
UpCountS);
    logic[3:0] DownCountE, UpCountE;

                Registrador   DownCount(.Clock(Clock),   .Entrada(DownCountE),
.Saida(DownCountS)),
                                      UpCount(.Clock(Clock), .Entrada(UpCountE),
.Saida(UpCountS));

    always_comb
        if(Enable)
            if(Swap) begin
                DownCountE = UpCountS;
                UpCountE = DownCountS;
            end
            else begin
                DownCountE = DownCountS - 4'd1;
                UpCountE = UpCountS + 4'd1;
            end
        else begin
            DownCountE = DownCountS;
            UpCountE = UpCountS;
        end
endmodule
```

Teste.sv

```
module Teste;
    logic Swap, Enable, Clock;
    logic[3:0] DownCountS, UpCountS;

            UpDownCount   Test(.Swap(Swap),   .Enable(Enable),   .Clock(Clock),
.DownCountS(DownCountS), .UpCountS(UpCountS));

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

        Swap = 0;
```

```
            Enable = 0;
            Clock = 0;

            repeat(1000) #10 Clock = ~Clock;
        end

        initial begin
            #15 Enable = 1;
            #30 Enable = 0;
            #20 Enable = 1;
        end

        initial repeat(1000) begin
            #100 Swap = 1;
            #15 Swap = 0;
        end
    endmodule
```

2.

Registrador.sv

```
    module Registrador(input Clock, input[8:0] Entrada, output logic[8:0] Saida);
        always_ff @(posedge Clock)
            Saida <= Entrada;
    endmodule
```

Circuito.sv

```
    module Circuito(input Clock, output logic f);
        logic[8:0] count, EntradaCount, EntradaSaida;

        Registrador Count(.Clock(Clock), .Entrada(EntradaCount), .Saida(count)),
                                        Saida(.Clock(Clock), .Entrada(EntradaSaida),
.Saida(f));

        always_comb begin
            if(count==9'd499)
                EntradaCount = 9'd0;
            else
                EntradaCount = count + 9'd1;

            if(count>9'd19 && count<9'd90)
                EntradaSaida = 9'd0;
            else
                EntradaSaida = 9'd1;
        end
    endmodule
```

//Se o clock de entrada for de 100MHz o sinal de saída será de 0.2MHz

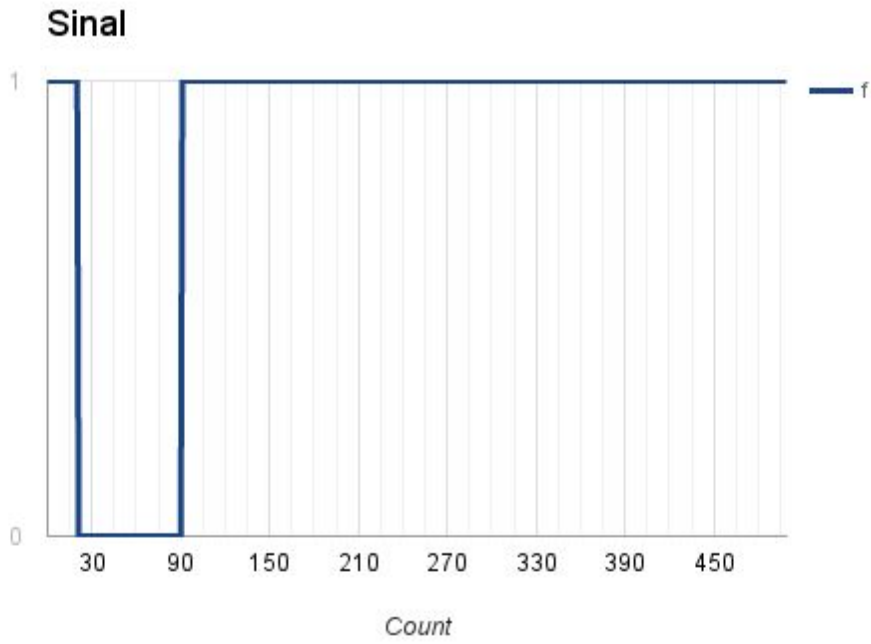

Figura 1 - Sinal f em função do número de contagem

Tabela 1 - Sinal f em função do número de contagem

Count	f
0	1
20	1
21	0
90	0
91	1
499	1

3.

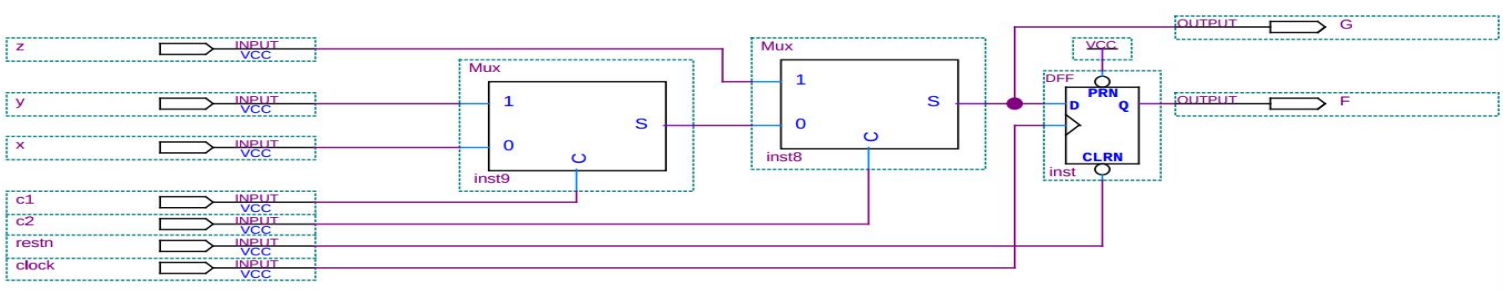

Figura 2 - Circuito da terceira questão

4.

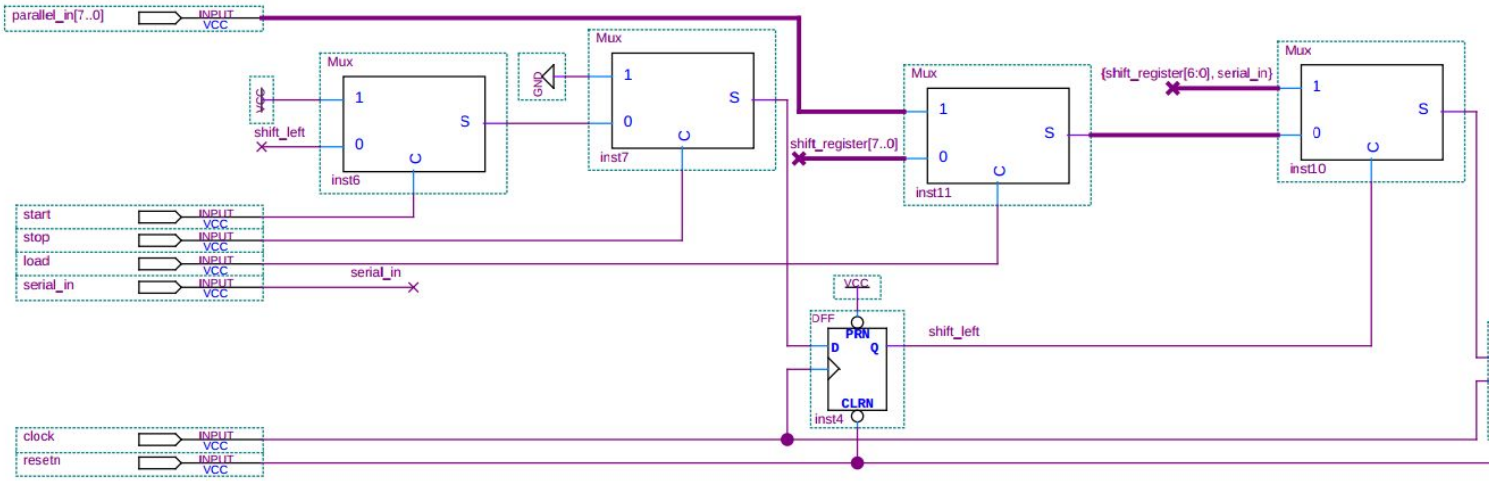

Figura 3 - Primeira parte do circuito da quarta questão

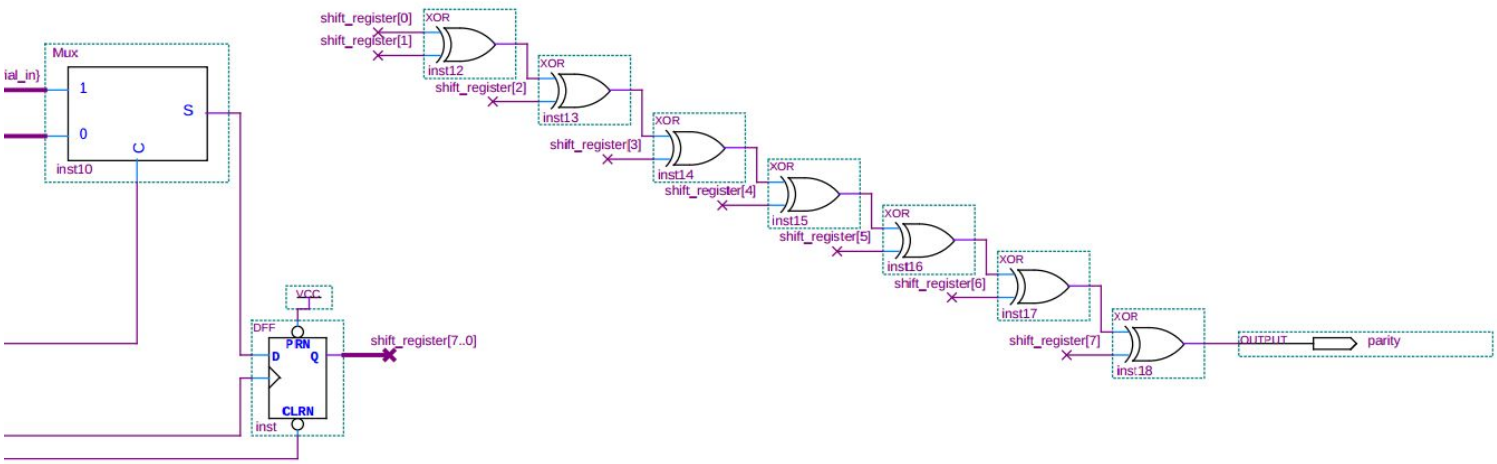

Figura 4 - Segunda parte do circuito da quarta questão

5.

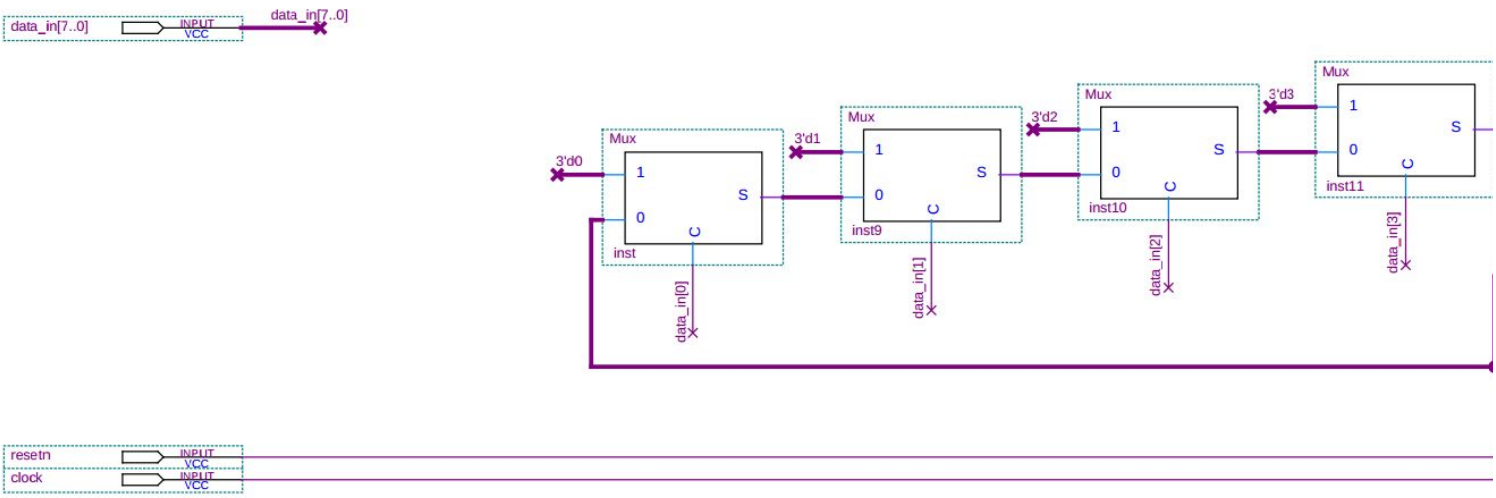

Figura 5 - Primeira parte do circuito da quinta questão

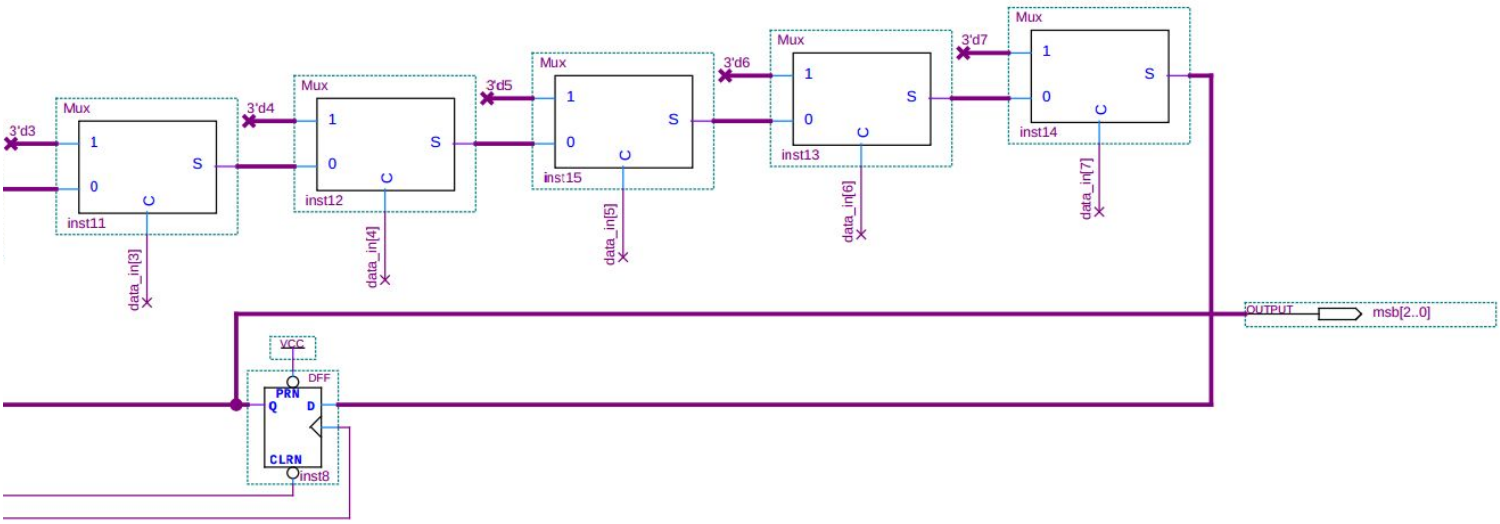

Figura 6 - Segunda parte do circuito da quinta questão

Q5.sv

```
    module Q5(input logic resetn, clock, input logic[7:0] data_in, output
logic[2:0] msb);
        integer i;

        always_ff @(posedge clock or negedge resetn) begin
            if(!resetn)
                msb <= 3'b000;
            else begin
                i = 0;

                while(i<8) begin
                    if (data_in[i])
                        msb <= i;

                    i = i + 1;
                end
            end
        end
    endmodule
```

//Sim, é possível a troca do for pelo o while pois temos uma quantidade fixa de repetições

6.

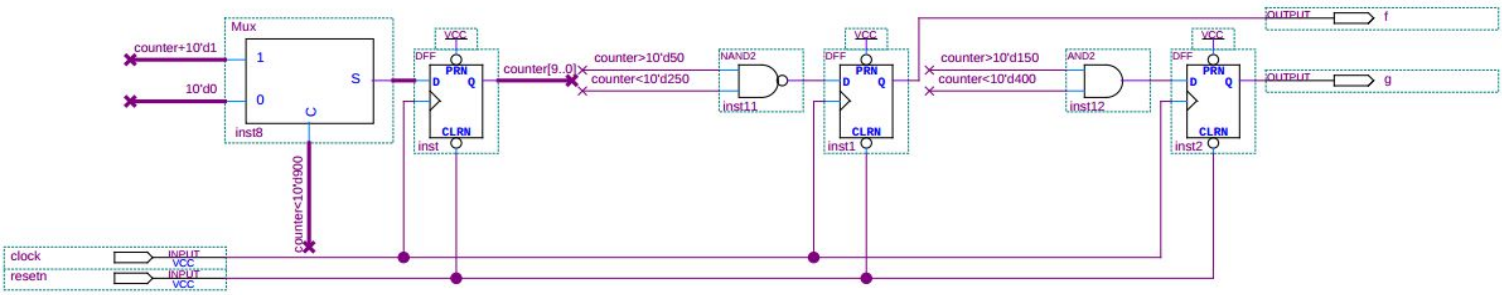

Figura 7 - Circuito da sexta questão

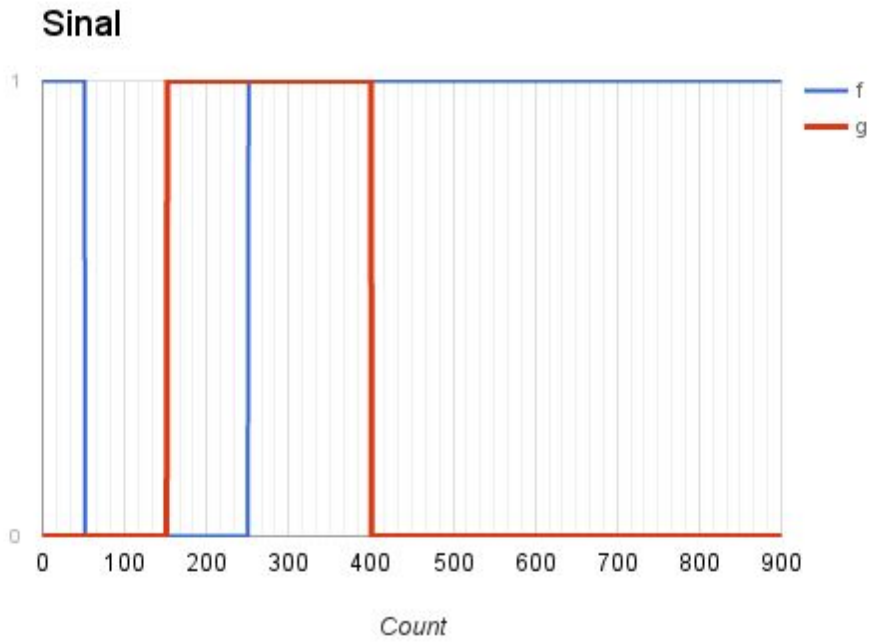

Figura 8 - Sinal f e g em função do número de contagem

Tabela 2 - Sinal f e g em função do número de contagem

Count	f	g
0	1	0
51	1	0
52	0	0
151	0	0
152	0	1
250	0	1
251	1	1

400	1	1
401	1	0
900	1	0

//Se o clock de entrada for de 200MHz a frequência de saída dos sinais f e g será de 0.22MHz

7.

A variável 'a' recebe o valor antigo do 'b' mais o valor antigo do 'c' pois como é bloqueante ele recebe os valores do 'b' e do 'c' antes de serem atualizados.

Já variável 'b' recebe o valor do 'a' mais o valor antigo do 'c' e a variável 'c' recebe o novo valor do 'b' e do 'a'.

Precisamos de um flip-flop para a variável 'b' porque para as contas utilizamos seu valor atualizado e desatualizado.

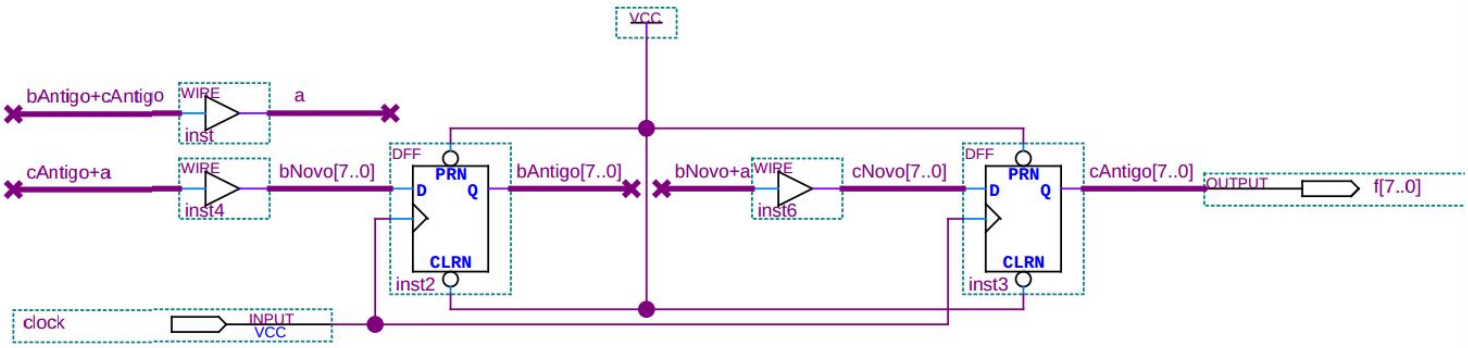

Figura 9 - Circuito da sétima questão

8.

A única diferença dessa questão para a questão 7 é que o valor do 'c' será o 'b' antigo no lugar do novo, pois como o 'b' é não-bloqueante a operação de atualização da variável só é feita após todas as outras operações serem feitas, então o valor que vale na operação de soma na atribuição da variável 'c' é o valor não atualizado da variável 'b'.

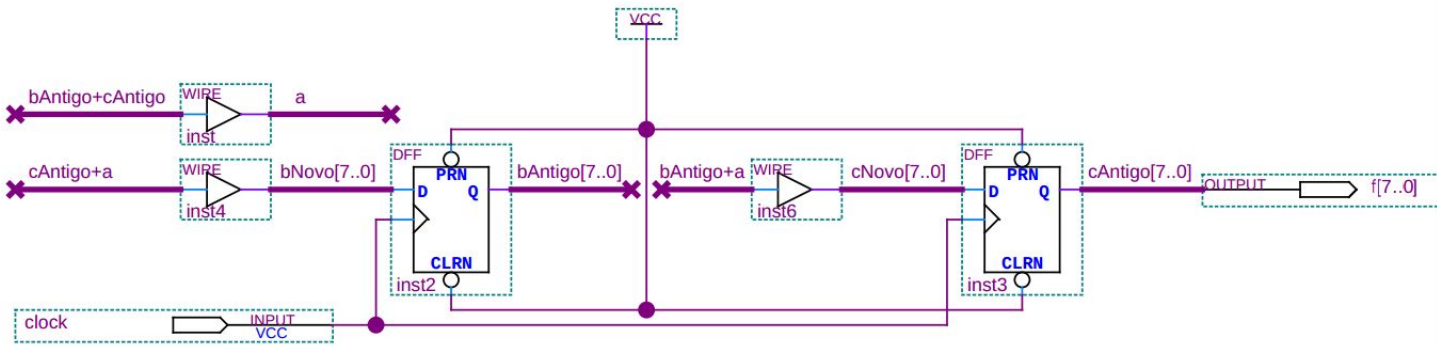

Figura 10 - Circuito da oitava questão

9.

Para este circuito precisamos adicionar um flip-flop para armazenar a variável 'a', pois utilizaremos seu valor atualizado e desatualizado.

Essa variável recebe o resultado de 'b' + 'c', mas somente ao final de todas as operações por isso as outras operações que usarem 'a' devem utilizar seu valor antes dessa soma.

Por isso 'b' recebe o valor de 'a' desatualizado mais o valor de 'c' desatualizado também, pois como mostrado no código no momento desta operação o valor de 'c' ainda não foi atualizado.

E por fim 'c' recebe o valor antigo de 'a' e o novo de 'b'.

Concluímos que quando precisamos dos valores atualizados e desatualizados de uma variável ou quando precisamos guardá-la usamos um flip-flop.

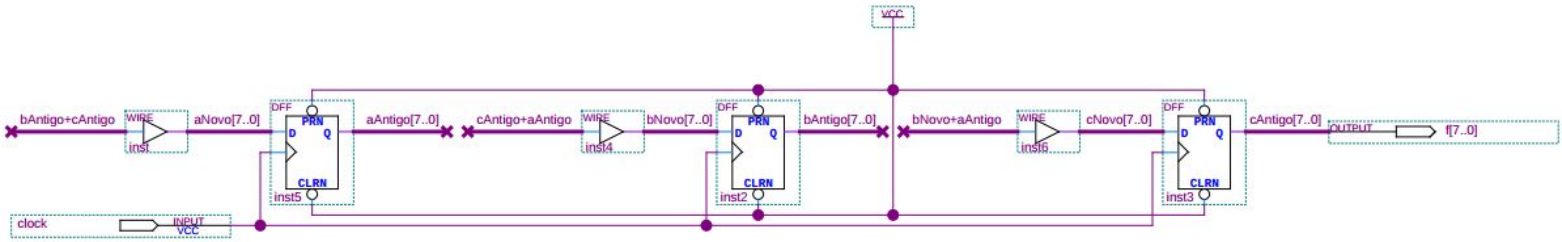

Figura 11 - Circuito da nona questão

10.

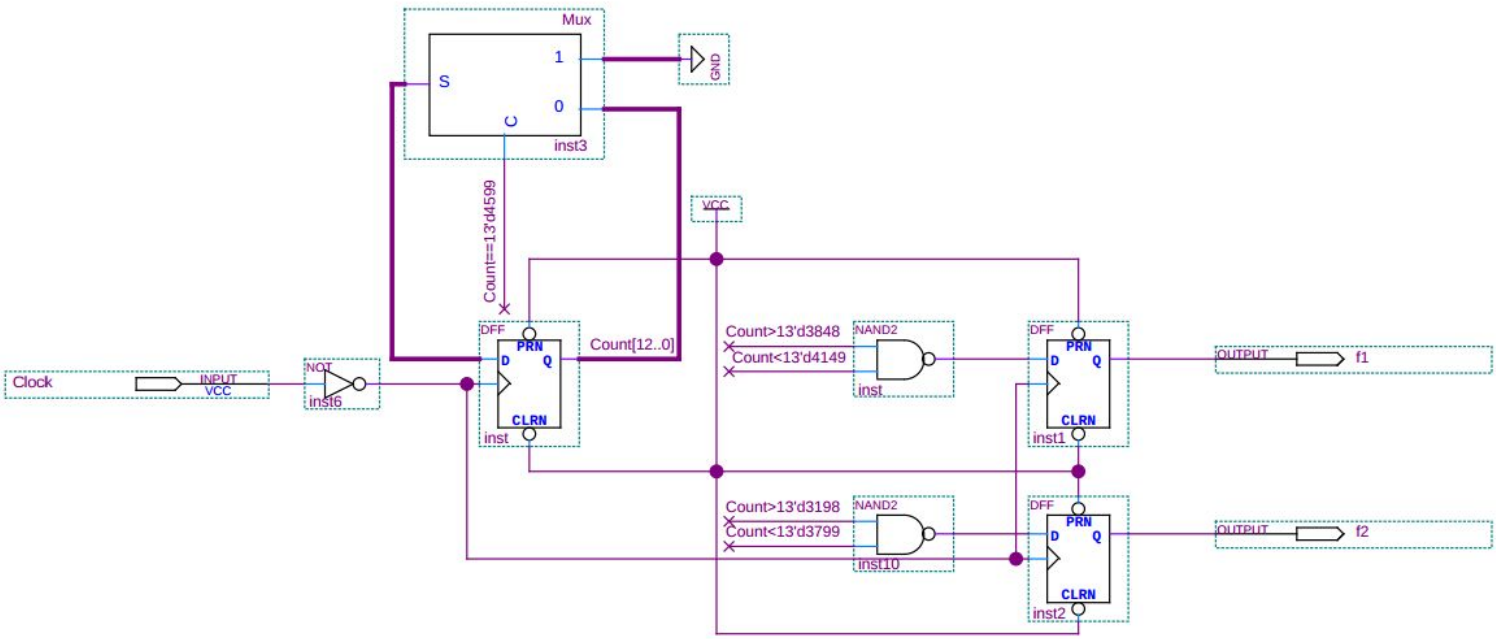

Figura 12 - Circuito da décima questão

ContadorDescida.sv

```
module ContadorDescida(input Clock, output logic[12:0] Saida);
    always_ff @(negedge Clock) begin
        if(Saida == 13'd4599)
            Saida <= 13'd0;
        else
            Saida <= Saida + 13'd1;
    end
endmodule
```

Circuito.sv

```
module Circuito(input Clock, output logic f1, f2);
    logic[12:0] contador;

    ContadorDescida Count(.Clock(Clock), .Saida(contador));

    always_ff @(negedge Clock) begin
        if(contador>=13'd3849 && contador<13'd4149)
            f1 <= 13'd0;
        else
            f1 <= 13'd1;

        if(contador>=13'd3199 && contador<13'd3799)
            f2 <= 13'd0;
        else
            f2 <= 13'd1;
    end
endmodule
```

//Para as simulações o clock foi gerado com estado inicial igual a 0

Considerações finais

Todos os arquivos e imagens contidos neste documento pode ser encontrado em melhor qualidade nesse link: <https://github.com/gvilardefarias/Gustavo_Ex02_SV>.