Projeto de Excelência de Microeletrônica

Segundo Exercício de System Verilog

Aluno: Laerty Santos da Silva

Respostas

Questão 01:

Q1.sv

```
module Q1(input swap, enable, clock,
                  output logic[3:0] outputUp, outputDown);
      logic[3:0] inputUp, inputDown, inputEnableUp, inputEnableDown;

      always_ff @(posedge clock) begin
            outputUp <= inputUp;
            outputDown <= inputDown;
      end

      always_comb begin
            if(swap) begin
                  inputEnableUp <= outputDown;
                  inputEnableDown <= outputUp;
            end
            else begin
                  inputEnableUp <= outputUp + 4'd1;
                  inputEnableDown <= outputDown - 4'd1;
            end

            if(enable) begin
                  inputUp <= inputEnableUp;
                  inputDown <= inputEnableDown;
            end
            else begin
                  inputUp <= outputUp;
                  inputDown <= outputDown;
            end
      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
```

Questão 02:

Q2.sv

```
module Q2(input clock,  output logic f, output logic[8:0] count);
      logic[8:0] inputCount;
      logic inputF;
            always_ff @(posedge clock) begin
            f <= inputF;
            count <= inputCount;
      end

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

            if(count>9'd19 && count<9'd90)
                  inputF = 1'd0;
            else
                  inputF = 1'd1;
      end
endmodule
```

/* Se a frequência de entrada for 100 MHz, como são 500 ciclos de clock, então a frequência de saída será de 0,2 MHz*/

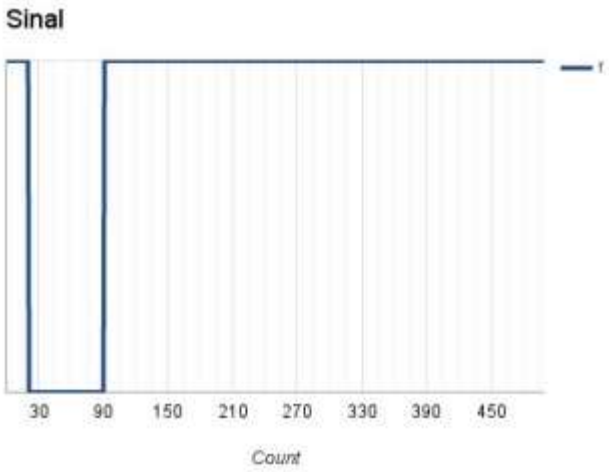
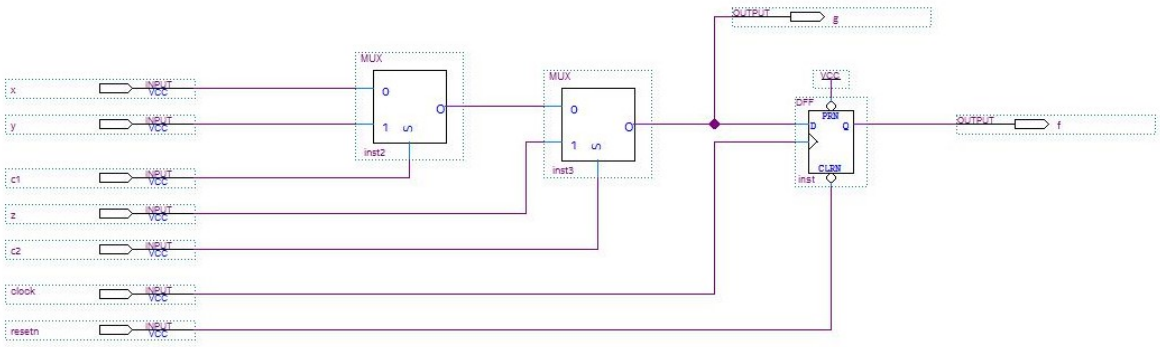


Figura 1 - Sinal f em função do número de contagem

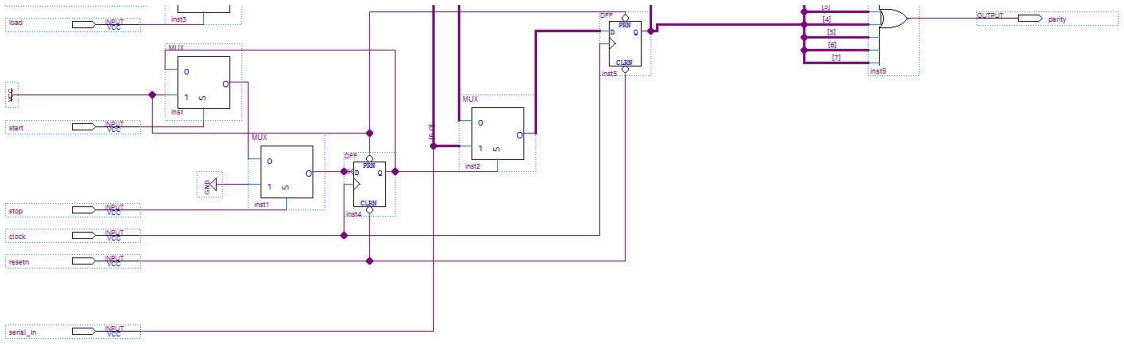
Tabela 1 - Sinal f em função do número de contagem

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

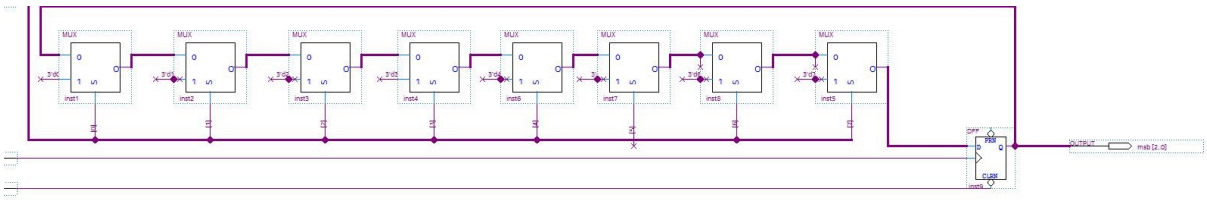
Questão 03:



Questão 04:



Questão 05:

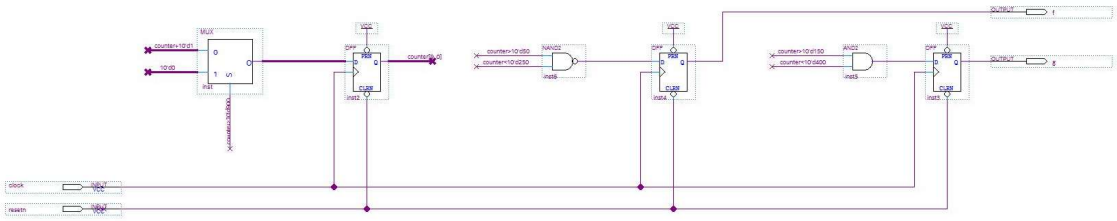


Q5.sv

```
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) begin
                  msb <= 3'b000;
            end
            else begin
            i = 3'd0;
            while (i<=3'd7) begin
                  if (data_in[i]) msb <= i;
                  i = i + 3'd1;
            end
        end
   end
endmodule
```

/* O código da questão é sintetizável e como pode ser visto, o laço for pode ser trocado por um while porque se tem um número limitado de repetições.*/

Questão 06:



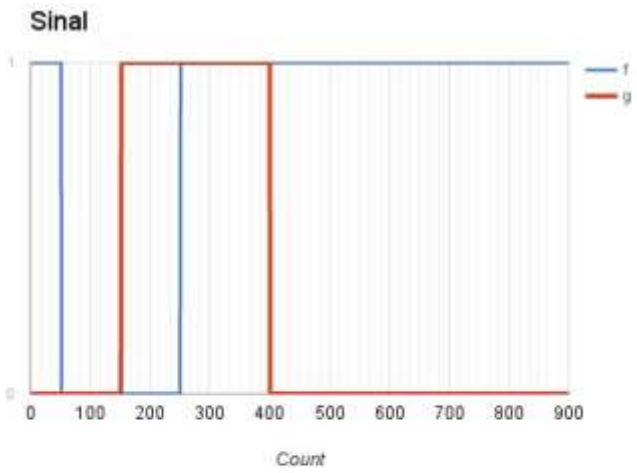
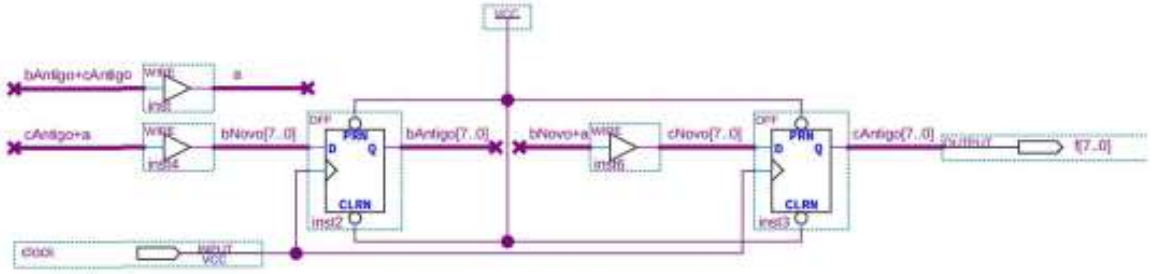


Figura 8 - Sinal f e g em função do número de contagem

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

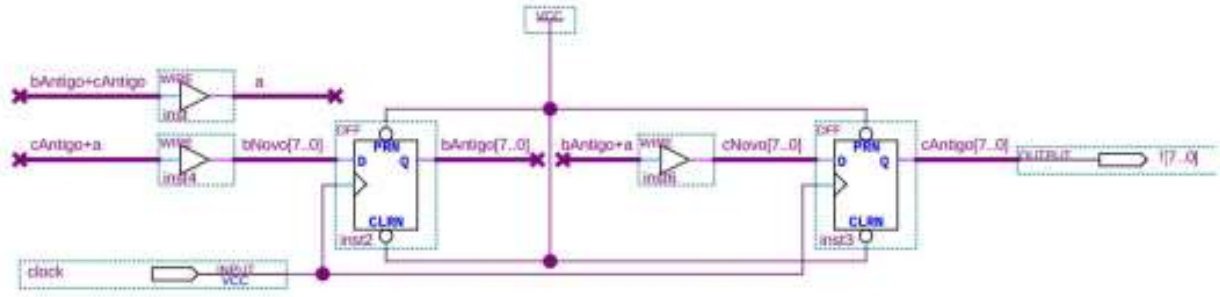
Questão 07:



No circuito em estudo, a variável "*a"* recebe o resultado soma dos valores antigos de "*b"* e "*c"* devido ao fato de ter-se utilizado uma atribuição não bloqueante. Por sua vez, a variável "*b"* recebe a soma de "*a"* e o "*c"* antigo e a variável *"c"* enfim recebe os valores atualizados de *"b"* e de *"a"*.

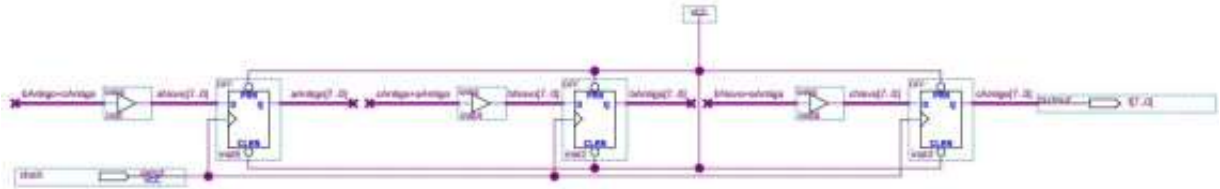
Sendo assim, será necessária a utilização de um flip-flop para a variável *"b"*, pois, em alguns momentos, será necessário usar tanto o valor antigo quanto o atualizado dessa mesma variável.

Questão 08:



Muito similar à questão anterior, com a diferença que a variável "c" receberá o valor antigo de "b" ao invés do atualizado.

Questão 09:



Novamente similar às anteriores, sendo que nesta questão foi preciso adicionar um flip-flop para servir com registrador da variável "a", para que seja possível utilizar o seu valor antigo e seu valor atualizado. Essa variável, só recebe a soma dos valores antigos de "b" e "c" ao final de todas as operações, exigindo que todas as operações com o "a" antigo sejam realizadas antes de armazenar o resultado da soma nessa variável.

Sendo assim, pudemos perceber que sempre que um circuito precise tanto do valor antigo quanto do atualizado de uma variável, devemos usar um flip-flop para armazenar essa variável.

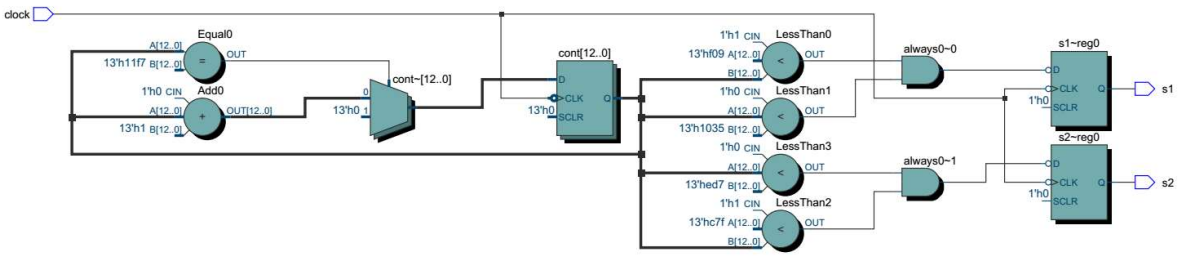
Questão 10:

Q10.sv

```
module Q10(input clock, output logic s1, s2);
    logic[12:0] cont;
    always_ff @(negedge clock) begin
     if(cont == 13'd4599)
              cont <= 13'd0;
        else
             cont <= cont + 13'd1;

        if(cont>=13'd3849 && cont<13'd4149)
              s1 <= 13'd0;
        else
              s1 <= 13'd1;

        if(cont>=13'd3199 && cont<13'd3799)
              s2 <= 13'd0;
        else
              s2 <= 13'd1;
    end
endmodule
```



Observações: Os arquivos referentes à resolução deste exercício podem ser encontrado no link: https://github.com/laerty/Laerty_Ex02_SV .