Programa para Excelência em Microeletrônica
SystemVerilog

Resposta da Lista 2

Aluna: Cicera Maria Marinho Rodrigues

10 de dezembro de 2016

Campina Grande – PB

Resposta da 1ª Questão

Código do Registrador:

```
module Registrador (
input Clock ,
input [ 3 : 0 ] Entrada ,
output logic [ 3 : 0 ] Saida );

always_ff @( posedge Clock)
Saida <= Entrada;
Endmodule
```

Codigo do UpDownCout:

```
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
```

Código de Teste:

```
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
```

Resposta da 2ª Questão

Código do Registrador:

```
module Registrador (
input Clock , input [8:0] Entrada ,
output logic [8:0] Saida );

always_ff @( posedge Clock)
Saida <= Entrada;
Endmodule
```

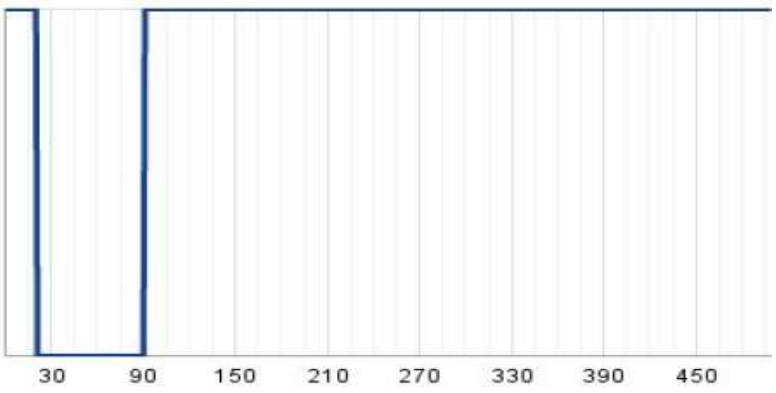
Código do Circuito:

```
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
```

// entrada do clock é de 100MHz, logo a saída será de 0.2MHz

Gráfico do Sinal F



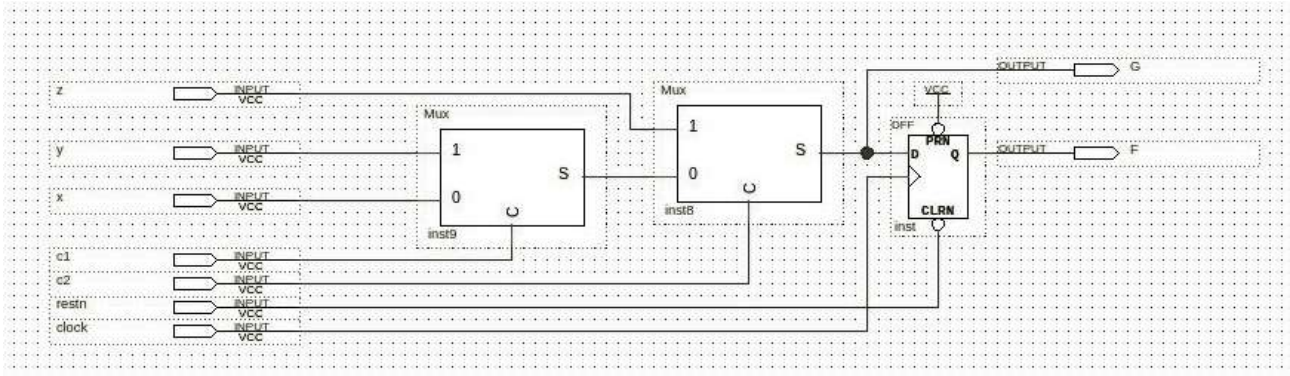
Count

Tabela do sinal F

Count	*F*
0	*1*
20	*1*
21	*0*
90	*0*
91	*1*
499	*1*

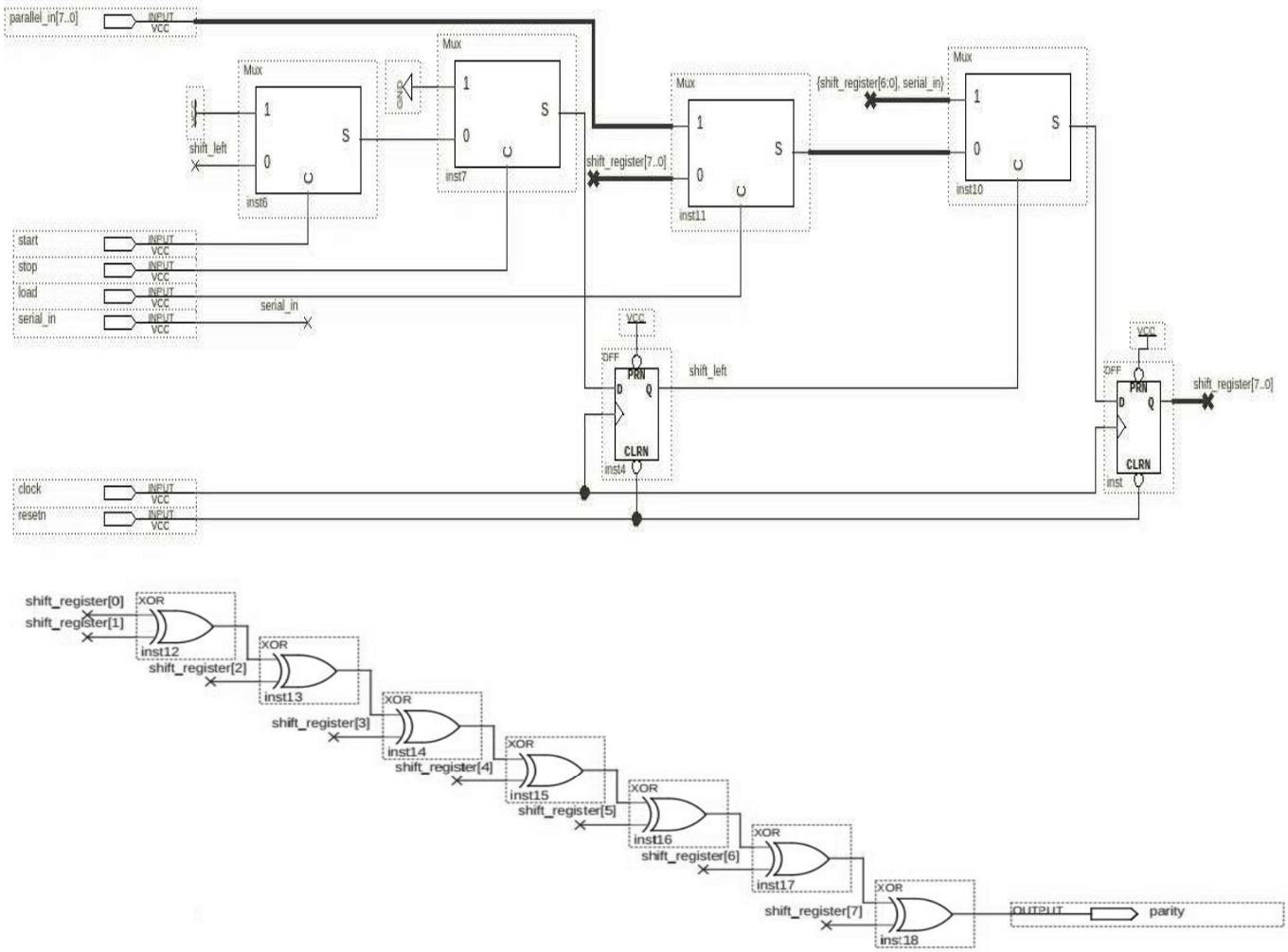
Resposta da 3ª Questão

Circuito



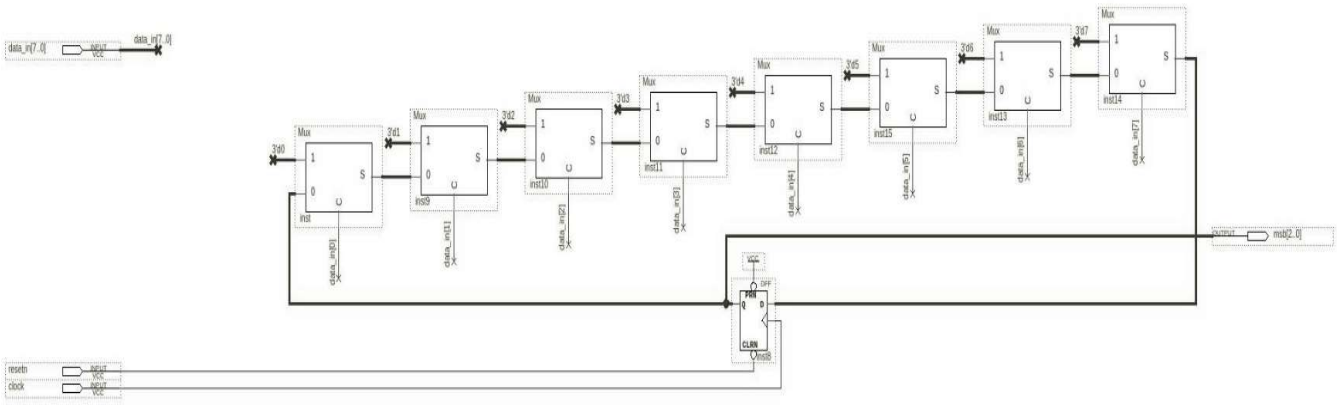
Resposta da 4ª Questão

Circuito



Resposta da 5ª Questão

Circuito

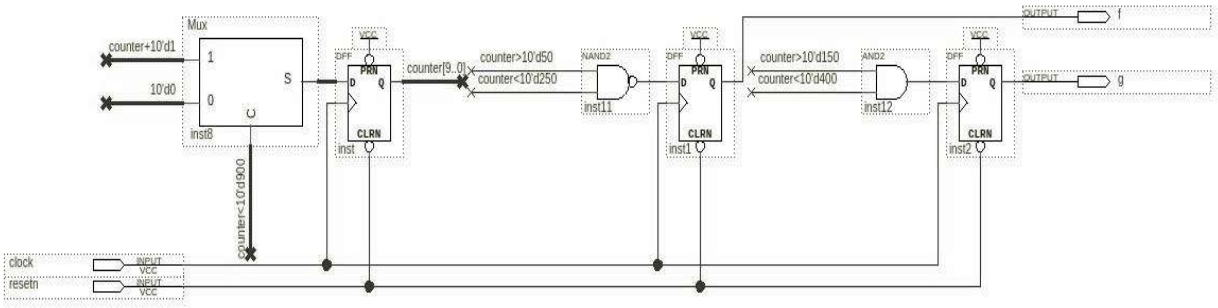


```
module Q5 (
input logic resetn , clock ,
input logic [7:0] data_in ,
output logic [2:0] msb );
integer i;
always_ff @( posedge clock or negedge resetn ) begin
if (! resetn)
msb <= 3 'b000;
else begin
i = 0;
while ( i < 8 ) begin
if ( data_in [ i ])
msb <= i;
i = i + 1;
end
end
end
endmodule
```

//Sim, pois como temos uma fixa quantidade de repetição é possível a substituição do for pelo while.

Resposta da 6ª Questão

Circuito



Sinal de f e g

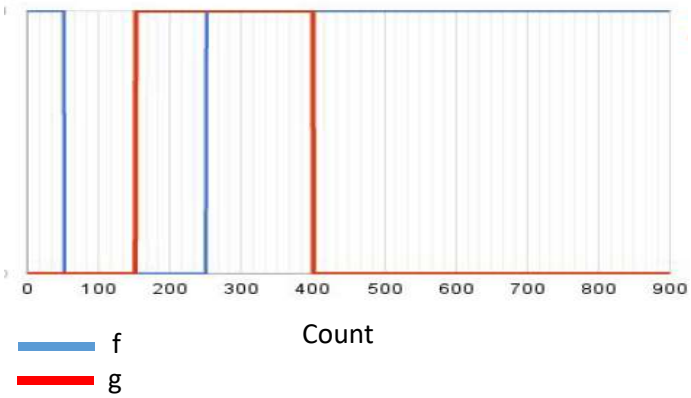


Tabela em função dos sinais e o Count

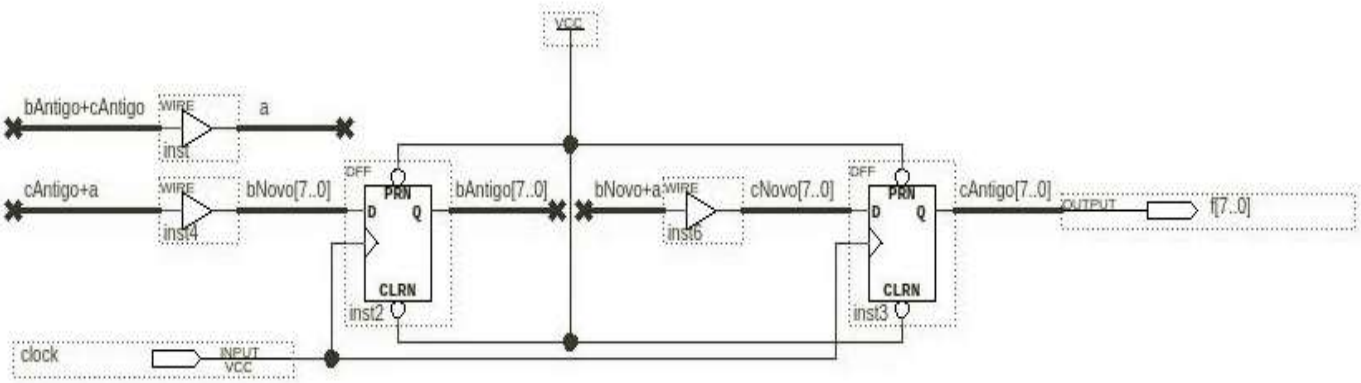
Count	*f*	*g*
0	*1*	*0*
51	*1*	*0*
52	*0*	*0*
151	*0*	*0*
152	*0*	*1*
250	*0*	*1*
251	*1*	*1*
400	*1*	*1*
401	*1*	*0*
900	*1*	*0*

//Caso a entrada do clock for de 200MHz a frequência gerada de saída dos sinais f e g será de 0.22MHz

Resposta da 7ª Questão

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.

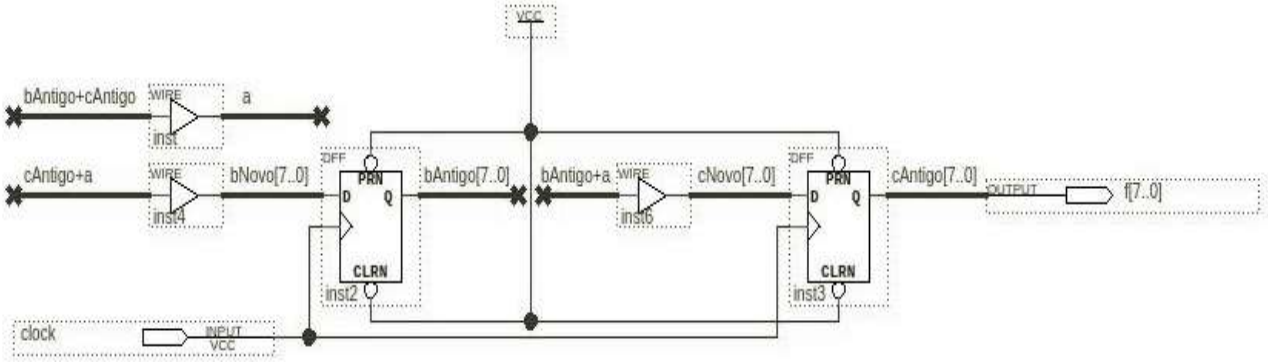
Circuito



Resposta da 8ª Questão

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'.

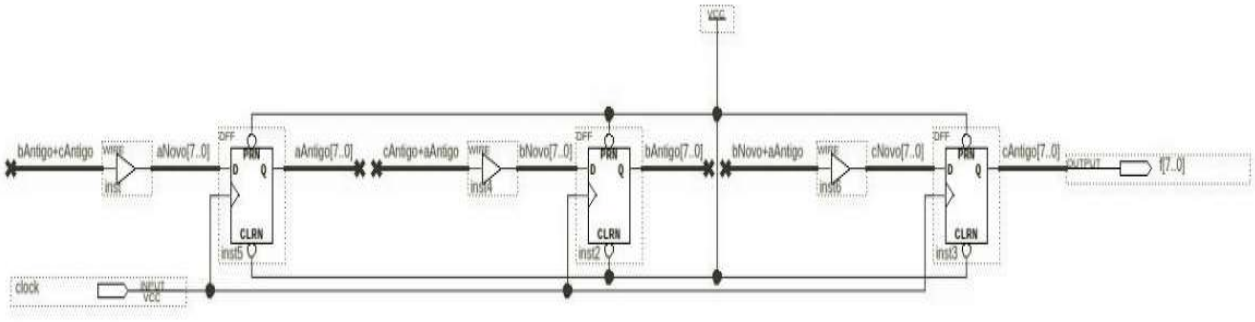
Circuito



Resposta da 9ª Questão

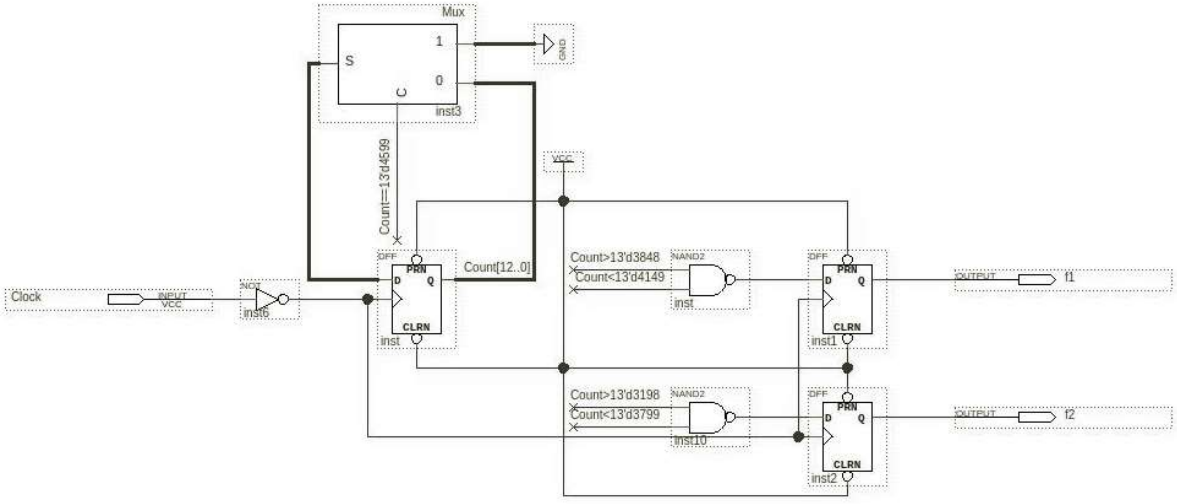
Para este circuito precisamos adicionar um flip-flop para armazenar a variável 'a', pois utilizaremos seu valor atualizado e desatualizado. Essa variável recebe o resultado de 'b' + 'c', mas somente ao final de todas as operações por isso as outras operações que usarem 'a' devem utilizar seu valor antes 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.

Circuito



Resposta da 10ª Questão

Circuito



Código do Contador de Descida:

```
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
```

Código do Circuito:

```
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 simulaçõeses o clock foi gerado com estado inicial igual a 0

Link do Github: https://github.com/CiceraRodrigues/Cicera_Ex02_SV