

**CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS –
CEFET/MG**

**DIOGO EMANUEL ANTUNES SANTOS
GUILHERME AUGUSTO DE OLIVEIRA
LUIZ EDUARDO LEROY SOUZA**

Circuito Sequencial — Contador Síncrono de 4 bits
Relatório 9

**BELO HORIZONTE
2021**

1. Objetivos

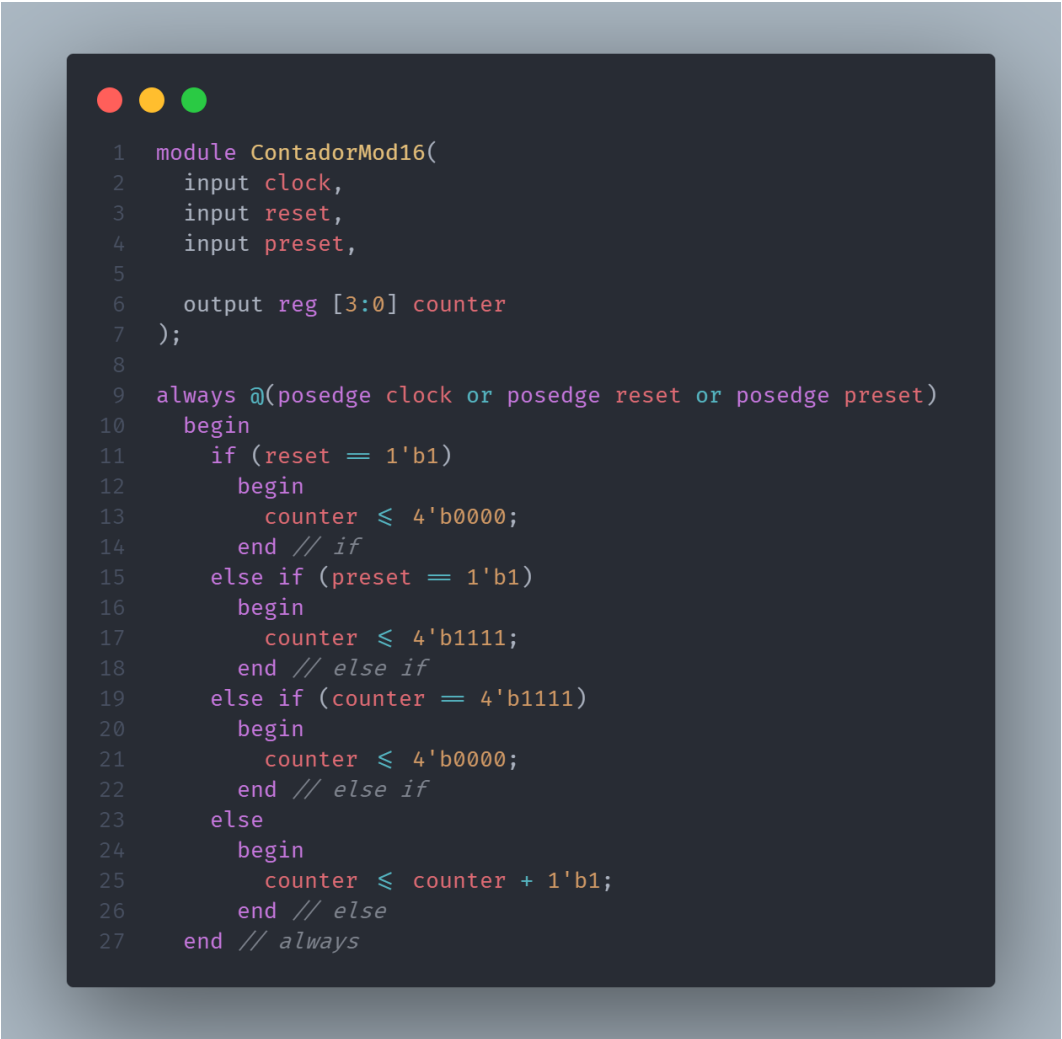
O objetivo do relatório é criar um circuito sequencial utilizando o modelo de hardware comportamental da linguagem Verilog HDL para criar dois contadores síncronos de quatro bits. O primeiro deles possui módulo 16, enquanto o segundo possui módulo 13.

2. Desenvolvimento

Para esse trabalho não foi necessário utilizar tabela verdade e as suas respectivas expressões lógicas. Tudo foi desenvolvido utilizando as funcionalidades da lógica comportamental.

3. Código em Verilog

O código utilizado para desenvolver os dois circuitos é muito semelhante. Ambos representam contadores de quatro bits, mudando apenas o valor máximo armazenado em cada contador.



```
1 module ContadorMod16(
2     input clock,
3     input reset,
4     input preset,
5
6     output reg [3:0] counter
7 );
8
9 always @(posedge clock or posedge reset or posedge preset)
10 begin
11     if (reset == 1'b1)
12     begin
13         counter <= 4'b0000;
14     end // if
15     else if (preset == 1'b1)
16     begin
17         counter <= 4'b1111;
18     end // else if
19     else if (counter == 4'b1111)
20     begin
21         counter <= 4'b0000;
22     end // else if
23     else
24     begin
25         counter <= counter + 1'b1;
26     end // else
27 end // always
```

Imagem 01: Circuito contador de módulo 16

O módulo conta com uma variável *clock* para representar a passagem de tempo no circuito, além das variáveis *reset* e *preset*, que servem para zerar e posicionar o contador na última posição, respectivamente. Além disso, há também uma variável que representa a saída atual do contador.

Toda a lógica do circuito é desenvolvida dentro de um bloco *always*, tendo em vista que é necessário utilizar as funcionalidades do modelo comportamental. Foi definido que as entradas do circuito funcionam em borda de subida.

Dentro do *always* existem quatro estruturas condicionais. A primeira verifica se o *reset* foi acionado. Em caso positivo, o contador volta para a posição inicial. A segunda verifica se o *preset* foi ativado. Caso isso ocorra, o contador vai para a última posição, que é o número 15. Também foi implementada uma maneira de realizar um ciclo no contador. Se em alguma circunstância o contador chegar em 15 (seu valor máximo), o mesmo volta para o número 0, sem necessitar do acionamento da entrada *clear*. Por fim, ocorre a situação padrão e o contador aumenta seu valor de forma crescente.

A screenshot of a code editor with a dark background and light-colored text. The code is Verilog for a 4-bit counter module. It features a single `always` block triggered by the positive edges of `clock`, `reset`, and `preset`. Inside the block, there are four conditional statements: 1) If `reset` is high, set `counter` to 0. 2) If `preset` is high, set `counter` to 12 (4'b1100). 3) If `counter` reaches 12, set it back to 0. 4) Otherwise, increment `counter` by 1. The code is numbered from 1 to 19.

```
1  always @(posedge clock or posedge reset or posedge preset)
2      begin
3          if (reset == 1'b1)
4              begin
5                  counter <= 4'b0000;
6              end // if
7          else if (preset == 1'b1)
8              begin
9                  counter <= 4'b1100;
10             end // else if
11         else if (counter == 4'b1100)
12             begin
13                 counter <= 4'b0000;
14             end // else if
15         else
16             begin
17                 counter <= counter + 1'b1;
18             end // else
19     end // always
```

Imagem 02: Circuito contador de módulo 13

O segundo projeto possui a mesma lógica do primeiro. O que muda são as condições de parada. Como o contador possui módulo 13, uma vez que o valor 12 é computado, o contador retorna para o número 0. Além disso, a variável *preset* muda o valor do mesmo para 12.

```

1  reg TB_clock, TB_reset, TB_preset;
2  wire [3:0] TB_counter;
3
4  integer i = 0;
5
6  ContadorMod13 dut(TB_clock, TB_reset, TB_preset, TB_counter);
7
8  initial
9  begin
10
11     TB_reset = 1'b1; TB_preset = 1'b0; TB_clock = 1'b0; # 25;
12
13     for (i = 0; i < 30; i = i + 1)
14     begin
15         TB_reset = 1'b0;
16         TB_preset = 1'b0;
17
18         if(TB_clock == 1'b1)
19         begin
20             TB_clock ≤ 1'b0;
21         end // if
22     else
23     begin
24         TB_clock ≤ 1'b1;
25     end // else
26
27     # 25;
28
29     end // for
30
31     TB_reset = 1'b1; TB_preset = 1'b0; TB_clock = 1'b1; # 25;
32
33     for(i = 0; i < 10; i = i + 1)
34     begin
35         TB_reset = 1'b0;
36
37         if(TB_counter == 4'b0010)
38         begin
39             TB_preset = 1'b1;
40         end // if
41     else
42     begin
43         TB_preset = 1'b0;
44     end // else
45
46         if(TB_clock == 1'b1)
47         begin
48             TB_clock ≤ 1'b0;
49         end // if
50     else
51     begin
52         TB_clock ≤ 1'b1;
53     end // else
54
55     # 25;
56
57     end // for
58
59 end // initial

```

Imagem 03: Testbench para os dois projetos

As ondas geradas para a simulação dos dois projetos foram produzidas utilizando a mesma *testbench*. Primeiramente foram declaradas as variáveis de entrada e saída para criar o *dut*. Após isso é feito um bloco *initial* para que seja possível utilizar as funcionalidades de *loop* condição.

Na linha 11, o sinal de entrada *reset* é ativado, com o objetivo de iniciar o contador em 0. Após isso, inicia-se na linha 13 uma estrutura de repetição que a cada alterna o valor do clock entre 0 e 1 a cada iteração, que dura *25ps*. Assim, o clock terá uma frequência de *50ps*.

Com a finalidade de testar todas as possibilidades, na linha 44 é feita a ativação do sinal *reset*, zerando o contador. A seguir existe uma estrutura de repetição para testar a entrada *preset*. Para isso, inicialmente o contador conta até o dois e na iteração seguinte o sinal é ativado, trazendo o contador para o seu maior valor.

4. Esquema do circuito

A seguir é possível observar os dois circuitos sintetizados pelo software Quartus. Como foi gerado utilizando o modelo comportamental, o próprio compilador escolheu as melhores portas lógicas / estruturas pré-fabricadas para gerar o resultado final.

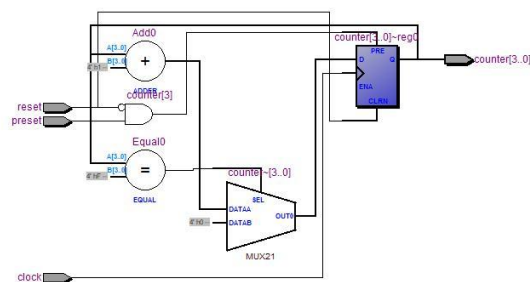


Imagem 04: Netlist Viewer do circuito de módulo 16

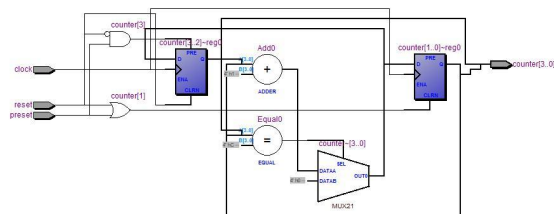


Imagem 05: Netlist Viewer do contador de módulo 13

5. Simulação das ondas no ModelSim

Simulação Contador Módulo 16

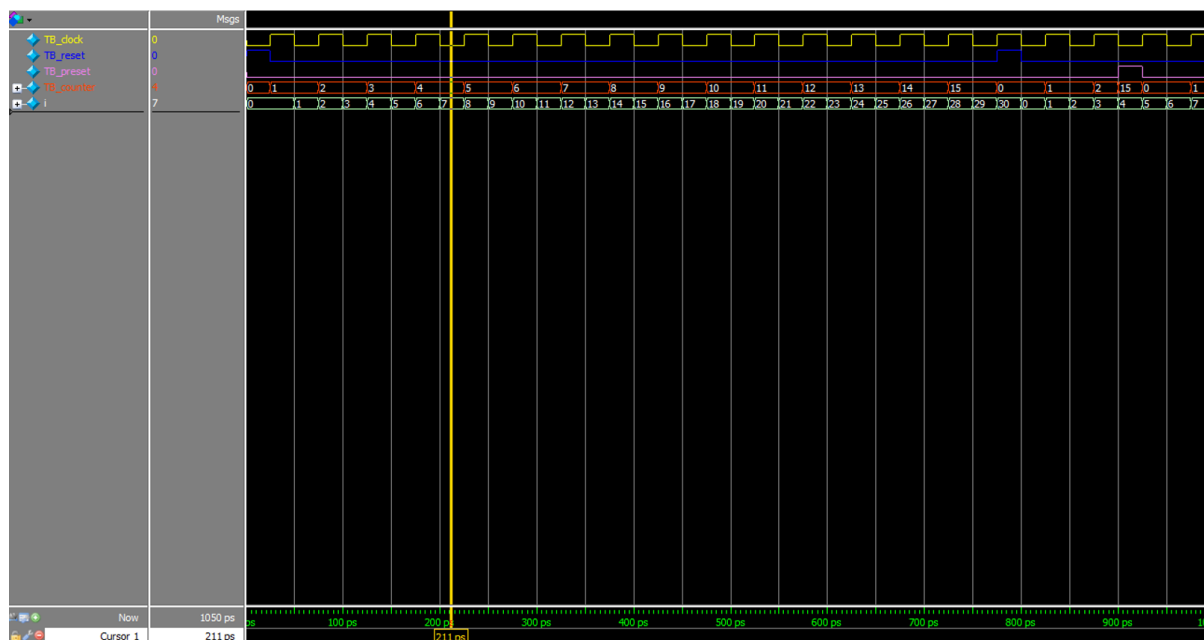


Imagem 6: Painel de ondas do Modelsim Altera Contador módulo 13

Acima temos a imagem referente a simulação do projeto de contador síncrono de módulo 16 na qual temos a seguinte situação, o clock, representado pela onda em amarelo que possui um período de 50 picossegundos. Abaixo deste, temos na cor azul e na cor branca respectivamente, as ondas representando os estímulos das entradas assíncronas reset e preset, o primeiro responsável por aplicar 0 às saídas e o outro responsável por aplicar 1 às saídas.

Em seguida temos os quatro bits que representarão o contador, assim podemos contar no intervalo de 0 a 15.

Analisando as resultados da simulação, podemos observar que nos primeiros picossegundos da mesma até a metade do primeiro ciclo do clock, a entrada assíncrona reset possui nível lógico 1, o que provoca a inicialização do contador com 0 que conta até atingir o número 14 em binário. Quando o número 15 é atingido, o contador é reiniciado, assim como ocorre posteriormente na simulação quanto o reset possui nível lógico até o momento de 800 picossegundos de simulação. por fim, aplicamos o nível lógico alto na entrada preset que podemos observar a partir do momento 900 picossegundos de simulação em que é aplicado 1 às saídas do contador , fazendo com que seja apresentado o último número da sequência do contador.

Simulação do Contador Módulo 13

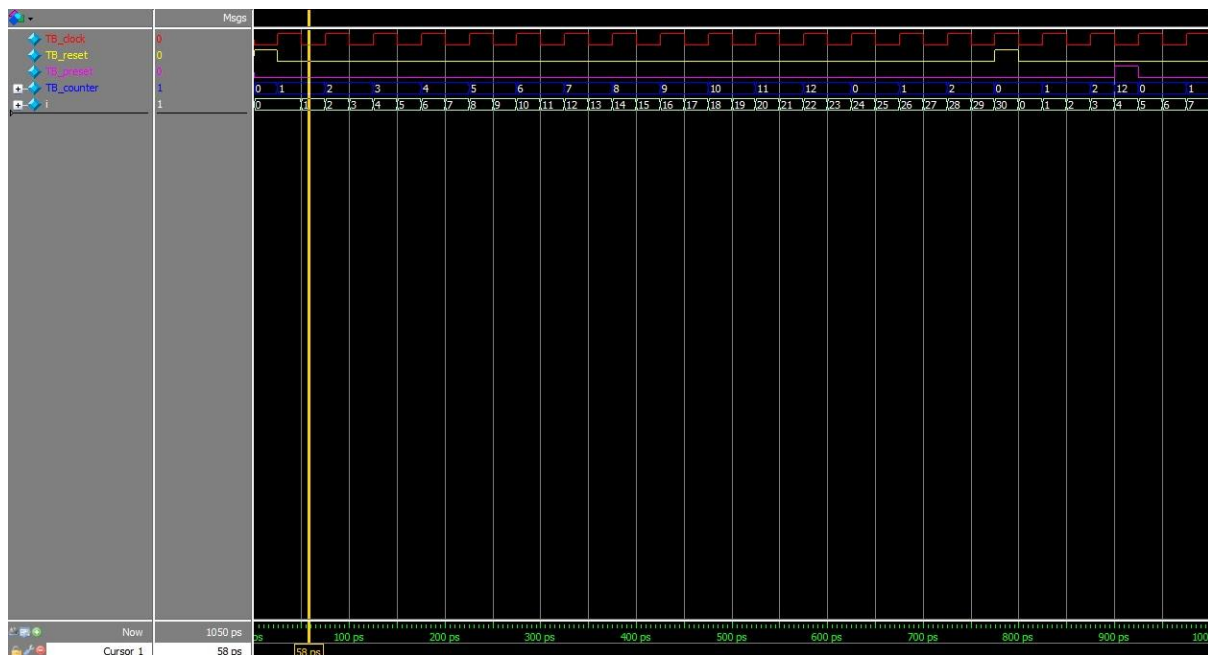


Imagem : Pannel de ondas do Modelsim Altera Contador módulo 13

Acima temos a imagem referente a simulação do projeto de contador síncrono de módulo 13 na qual temos a seguinte situação, o clock, representado pela onda em vermelho que possui um período de 50 picossegundos. Abaixo deste, temos na cor amarela e na cor rosa respectivamente, as ondas representando os estímulos das entradas assíncronas reset e preset, o primeiro responsável por aplicar 0 às saídas e o outro responsável por aplicar 1 às saídas.Em seguida temos os quatro bits que representarão o contador, assim podemos contar no intervalo de 0 a 15.

Analisando as resultados da simulação, podemos observar que nos primeiros picossegundos da mesma até a metade do primeiro ciclo do clock, a entrada assíncrona reset possui nível

lógico 1, o que provoca a inicialização do contador com 0 que conta até atingir o número 12 em binário. Quando o número 13 é atingido, o contador é reiniciado, assim como ocorre posteriormente na simulação quanto o reset possui nível lógico até o momento de 800 picossegundos de simulação. por fim, aplicamos o nível lógico alto na entrada preset que podemos observar a partir do momento 900 picossegundos de simulação em que é aplicado 1 às saídas do contador , fazendo com que seja apresentado o último número da sequência do contador.