Universidade de Brasília
Departamento da Engenharia Elétrica
Laboratório de Sistemas Digitais

Relatório 08

Contadores BCD

José Antônio Alcântara da Silva de Andrade Mat: 232013031

Professor:
Eduardo B. R. F. Paiva
Turma 08

Brasília, DF

8 de fevereiro de 2025

1 Objetivos

1. Implementar um contador BCD módulo 10 como uma máquina de estados do tipo Moore.
2. Implementar um contador BCD módulo 100 usando contadores BCD módulo 10 em cascata.

2 Atividade

Essa sessão do laboratório possui dois objetivos: construir um contador BCD (*Binary Coded Decimal*) módulo 10 e outro módulo 100.

2.1 Contador BCD 10

2.1.1 Modelagem

Um contador BCD módulo 10 é um contador de 4 bits usual, com o diferencial de retornar ao zero quando atinge nove. Para o contador requisitado, serão necessárias seis entradas e duas saídas.

As seis entradas são `clock`, para funcionalidade síncrona, `reset`, para o retorno ao estado inicial (aqui, o valor 0), `enable` e `rci`, ambas ativas em nível baixo e controla se o contador deve ou não contar, `load` e `d`, ambas permitindo o carregamento de um estado específico (aqui, o vetor `d`) no contador.

Para as duas saídas, `Q` será um vetor representando o estado atual e `RCO` indica que o contador está em seu último ciclo, ou seja, o valor 9.

Em questões de prioridade de ações, `reset` tem prioridade máxima, em seguida `load` e finalmente a contagem.

2.1.2 Implementação

Todas as entradas e saídas serão de apenas um bit, com excessão de `Q` e `D`, ambas vetores de 4 bits.

Listagem 1: Requisitos para o contador 10.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity CONTADORBCD10 is
5      port (
6          clock: in std_logic;
7          reset: in std_logic;
8          enable: in std_logic;
9          rci: in std_logic;
10         load: in std_logic;
11         d: in std_logic_vector(3 downto 0);
12         Q: out std_logic_vector(3 downto 0);
13         RCO: out std_logic
14     );
15 end CONTADORBCD10;
```

O código da Listagem 1 implementa todo o básico para a funcionalidade do contador. As bibliotecas para simulação de circuitos nas linhas 1 e 2, e as entradas e saídas da linha 6 até a linha 13.

Listagem 2: Arquitetura básica do contador 10.

```
1  architecture CONTADORBCD10_ARCH of CONTADORBCD10 is
2
3      type estado is (ST0,ST1,ST2,ST3,ST4,ST5,ST6,ST7,ST8,ST9);
4
5      signal currState, nextState : estado;
6      signal nextQ : std_logic_vector(3 downto 0);
7      signal nextRCO : std_logic;
8
9  begin
10     -- ...
11 end CONTADORBCD10_ARCH;
```

Para a arquitetura do circuito, primeiro define-se um novo tipo, `estado` (linha 3 da Listagem 2), que indicará todos os possíveis estados do contador. Para indicar qual estado ele se encontra em, e qual é o próximo, cria-se dois sinais do tipo `estado`, `currState` e `nextState`, respectivamente (linha 5). Finalmente, para realizar as mudanças sincronamente, cria-se um sinal `nextQ` (linha 6) e outro `nextRCO` (linha 7) para salvar os novos valores de `Q` e `RCO`, respectivamente, antes da subida do clock.

Após iniciar a arquitetura, serão definidos dois processos separados, um síncrono e outro combacional. O processo síncrono (Listagem 3) cuidará de atualizar as saídas apenas na subida do clock, enquanto o processo combinacional (Listagem 4) irá, assíncronamente, atualizar o próximo estado.

Listagem 3: Processo síncrono do contador 10.

```
1  sync_proc: process(clock)
2  begin
3      if rising_edge(clock) then
4          currState <= nextState;
5          Q <= nextQ;
6          RCO <= nextRCO;
7      end if;
8  end process sync_proc;
```

Na Listagem 3, primeiro define-se, na linha 1, o processo síncrono que ativa sempre que o clock muda de valor. Em seguida, verifica-se se o clock está em subida (linha 3) e, então, insere-se o próximo estado no estado atual e atualiza-se as saídas (linhas 4 a 6).

Listagem 4: Processo combinacional do contador 10.

```
1  comb_proc: process(currState, reset, load, d, enable, rci)
2  begin
3      if (reset = '1') then
4          nextState <= ST0;
5          nextQ <= "0000";
6          nextRCO <= '1';
7      elsif (load = '1') then
8          case (d) is
9              when "0000" =>
10                 nextState <= ST0;
11                 nextQ <= "0000";
12                 nextRCO <= '1';
13
14             -- ...
15
16             when "1001" =>
17                 nextState <= ST9;
18                 nextQ <= "1001";
19                 nextRCO <= '0';
20             when others =>
21                 nextState <= ST0;
22                 nextQ <= "0000";
23                 nextRCO <= '1';
24         end case;
25     elsif (enable = '0') and (rci = '0') then
26         case (currState) is
27             when ST0 =>
28                 nextState <= ST1;
```

```
29                  nextQ <= "0001";
30                  nextRCO <= '1';
31
32              -- ...
33
34              when ST8 =>
35                  nextState <= ST9;
36                  nextQ <= "1001";
37                  nextRCO <= '0';
38              when ST9 =>
39                  nextState <= ST0;
40                  nextQ <= "0000";
41                  nextRCO <= '1';
42              when others =>
43                  nextState <= ST0;
44                  nextQ <= "0000";
45                  nextRCO <= '1';
46          end case;
47      end if;
48  end process comb_proc;
```

Após definir que o processo combinacional deve ser acionado na mudança das variáveis de estado (`currState`, `reset`, `load`, `d`, `enable` e `rci`), cria-se um *if-statement* para seguir os requisitos de prioridade, na linha 3 da Listagem 4.

Começa-se com a maior prioridade, o `reset`, na linha 3 da Listagem 4, checando se o valor é 1 e, então, inserindo o estado ST0 (o estado inicial) junto de suas saídas. Similarmente, checa-se se `load` é 1 na seguinte condição (linha 7) e, então, usa-se de um case (linhas 8 a 23) para inserir o novo estado atual a partir de `d`, junto das saídas associadas àquele estado. Adicionalmente, casos não definidos de `d` resultam no carregamento do estado inicial.

Finalmente, após checar se tanto `enable` como `rci` estão em baixa na linha 25 da Listagem 4, a lógica do próximo estado é estabelecida. Usa-se de um *case-statement* na linha 26 para verificar o estado atual, e em seguida insere-se o próximo estado e os próximos valores das saídas em seus respectivos sinais. Cria-se um caso excepcional na linha 42 para capturar sinais incorretos e reestabelecê-los no sistema — isso é importante para os casos de sinais fracos ou indefinidos.

Alguns casos foram omitidos por questões de legibilidade e inseridos na Listagem 4 apenas implicitamente. A fim de ver o código completo do contador módulo 10, verifique a Listagem 7 no fim desse documento.

2.1.3 Simulação

Para as simulações do contador BCD módulo 10, realizam-se três: uma que testa a funcionalidade de contagem (Figura 1), outra que testa o `load` (Figura 2) e, finalmente, uma que testa o `reset` (Figura 3).

O teste de contagem, na Figura 1 é simples: inicia-se com um pequeno pulso no `reset` a fim de colocar o sistema em seu estado inicial, e, em seguida, apenas estabelece-se todas as entradas como 0. Durante os dez ciclos seguintes do `clock`, o valor de `Q` aumenta de um em um, finalmente atingindo o nove e retornando ao zero. A saída `RCO` ativa apenas quando `Q` é nove, como esperado.

O teste do load, na Figura 2, também inicia o teste com um pequeno pulso no `reset`. Após a realização de um ciclo do clock, carrega-se na máquina o estado seis, e então realiza-se mais cinco ciclos do clock, provando não só que o contador faz o ciclo como que `RCO` ativará sem problema.

Finalmente, na Figura 3, temos a simulação do reset. Duas vezes ele é testado: no pulso inicial do sistema e durante o uso do sistema. No ínicio da simulação, o resultado é corretamente carregado. Em seguida, após carregar o valor quatro e avançá-lo até seis,

Figura 1: Simulação de contagem do contador 10.

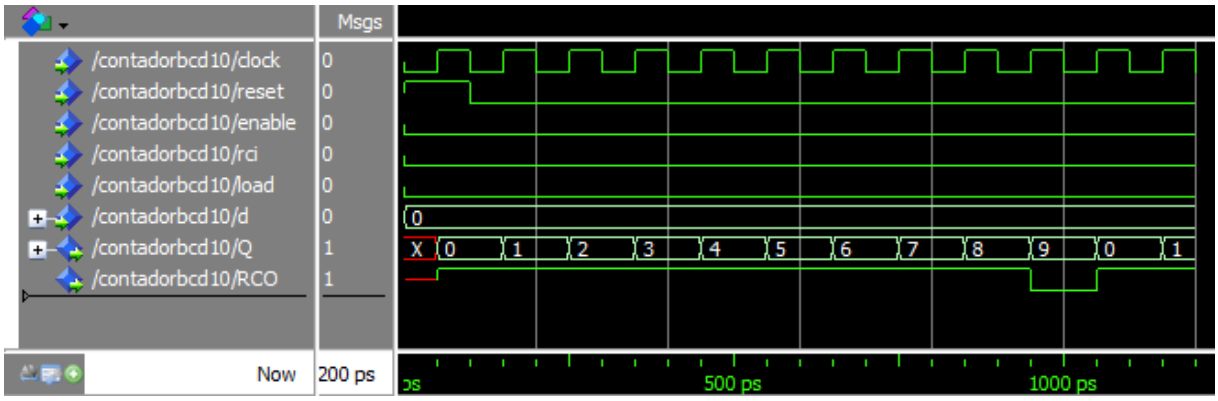
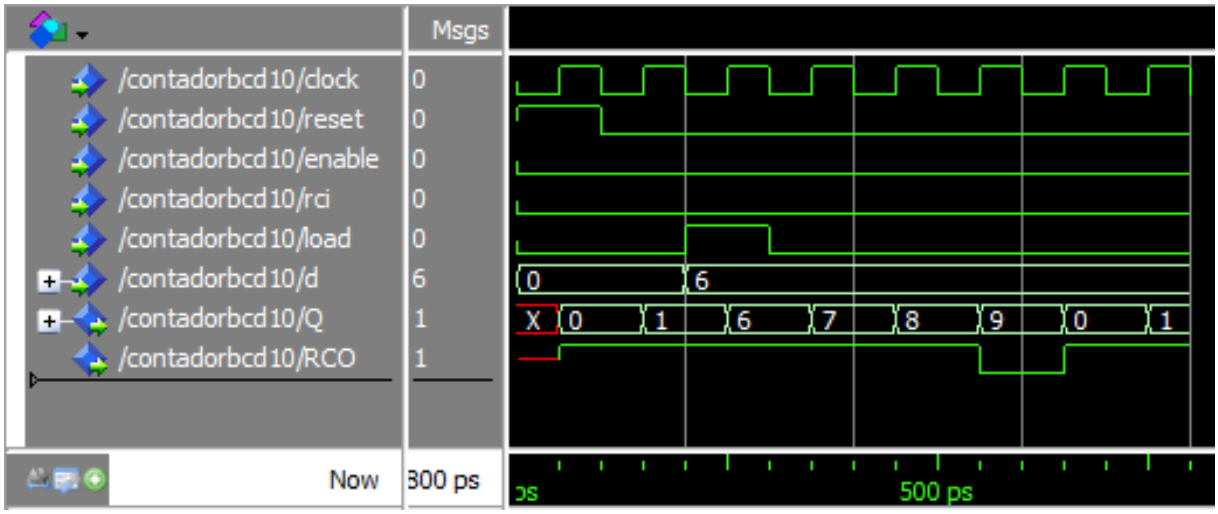
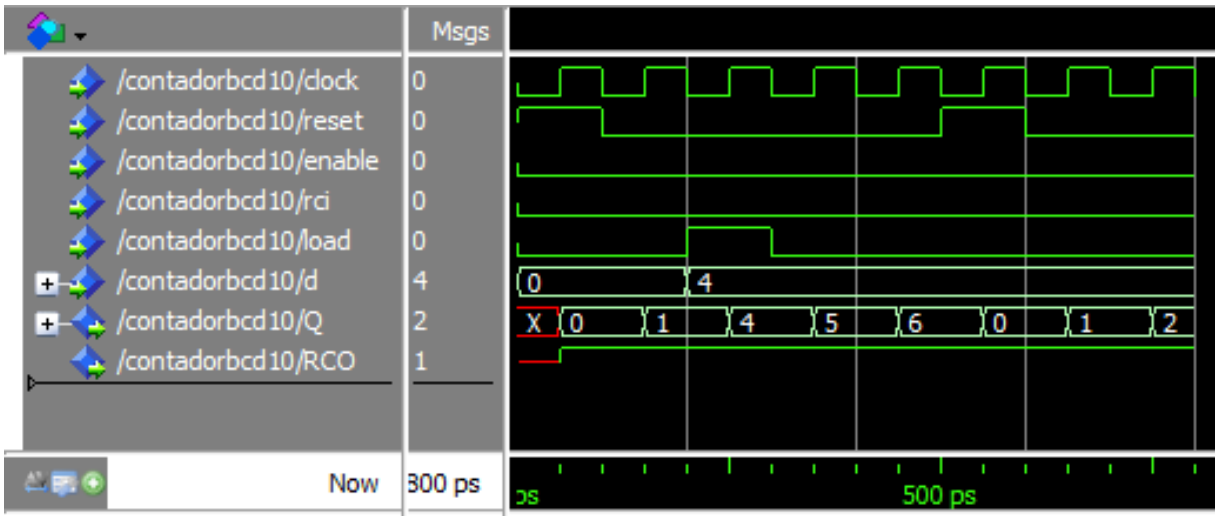


Figura 2: Simulação do load do contador 10.



novamente acionar o `reset` causa o retorno do valor para o zero, sem acionar o `RCO`.

Figura 3: Simulação do reset do contador 10.



2.2 Contador BCD 100

2.2.1 Modelagem

Um contador BCD módulo 100 é um contador similar ao BCD módulo 10, apenas agora atuando em dois dígitos ao invés de apenas um. Ou seja, ele possui uma saída para a dezena atual, e outro para a unidade atual.

Por requisito do experimento, serão necessárias as mesmas entradas e saídas do contador módulo 10, com algumas alterações. Para os valores de carregamento, `load` permanece igual, mas agora o contador aceita uma entrada de carregamento para as unidades, `d_uni`, e outra para as dezenas `d_dez`. Similarmente, não há apenas uma saída, mas duas, `Q_uni` para as unidades e `Q_dez` para as dezenas.

Além disso, o experimento requisita que o contador módulo 100 seja construído por cascateamento, ou seja, deverá ser usado dois contadores módulo 10 para simular cada dígito do contador. No ponto de vista lógico, a saída `RCO` do contador das unidades deverá ser usada para indicar quando que o contador das dezenas deve incrementar seu valor, enquanto a saída `RCO` das dezenas servirá para indicar que o contador atualmente se encontra no estado 99.

2.2.2 Implementação

Todas as entradas e saídas serão de apenas um bit, como no contador módulo 10, com excessão das entradas `D_uni` e `D_dez`, e as saídas `Q_uni` e `Q_dez`, todas as quatro vetores de 4 bits.

Listagem 5: Requisitos para o contador 100.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity CONTADORBCD100 is
5      port (
6          clock: in std_logic;
7          reset: in std_logic;
8          enable: in std_logic;
9          rci: in std_logic;
10         load: in std_logic;
11         d_uni: in std_logic_vector(3 downto 0);
12         d_dez: in std_logic_vector(3 downto 0);
13         Q_uni: out std_logic_vector(3 downto 0);
14         Q_dez: out std_logic_vector(3 downto 0);
15         RCO: out std_logic
16     );
17 end CONTADORBCD100;
```

O código da Listagem 5 implementa as bibliotecas necessárias (linha 1 e 2), e as portas anteriormente citadas (linhas 6 a 15).

Listagem 6: Arquitetura do contador 100.

```
1  architecture CONTADORBCD100_ARCH of CONTADORBCD100 is
2      component CONTADORBCD10 is
3          port (
4              clock: in std_logic;
5              reset: in std_logic;
6              enable: in std_logic;
7              rci: in std_logic;
8              load: in std_logic;
9              d: in std_logic_vector(3 downto 0);
10             Q: out std_logic_vector(3 downto 0);
11             RCO: out std_logic
12         );
13     end component;
14
15     signal uRCO, dRCO : std_logic;
```

```
16
17 begin
18     INT1 : CONTADORBCD10 port map(clock, reset, enable, rci, load, d_uni, Q_uni, uRCO);
19     INT2 : CONTADORBCD10 port map(clock, reset, enable, uRCO, load, d_dez, Q_dez, dRCO);
20
21     RCO <= uRCO or dRCO;
22
23 end CONTADORBCD100_ARCH;
```

Já, para a arquitetura do contador 100, o processo é bem simples. Basta, primeiramente, importar o contador 10 como um componente, aqui nas linhas 2 a 13 da Listagem 6. Em seguida, define-se dois sinais de bit simples, `uRCO` e `dRCO`, na linha 15, para conectar a saída `RCO` de ambos contadores, unidade e dezena.

Finalmente, as conexões são realizadas nas linhas 18 e 19, com `INT1` representando o contador das unidades e `INT2` representando o contador das dezenas. As entradas `clock`, `reset`, `enable` e `load` são conectadas identicamente em ambos contadores. A entrada `rci` do contador 100 é inserida no contador das unidades, e usa-se o sinal `uRCO` para receber a saída das unidades e conectá-la com a entrada `rci` do valor das dezenas. Já para os valores de `d` e `Q`, usa-se `d_uni` e `Q_uni` para as unidades, e `d_dez` e `Q_dez` para as dezenas.

Por fim, realiza-se a lógica da saída de `RCO` para o contador 100 na linha 21 da Listagem 6. Dessa forma, o contador BCD módulo 100 está completamente implementado.

O código completo do contador BCD módulo 100 pode ser visualizado na Listagem 8, no final desse documento.

2.2.3 Simulação

Para o contador módulo 100, serão realizados dois testes. Um que verifica a funcionalidade de contagem (Figuras 4 até 13) e outro que verifica a funcionalidade do `load` (Figura 14).

Inicia-se o teste da contagem com um pequeno pulso em `reset` (visualizado na Figura 4), e, em seguida, a permanência de todas as entradas em zero. O resultado da simulação segue nas Figuras 4 até 13. Nota-se, na Figura 13, que a saída `RCO` corretamente entra em baixa apenas no estado 99.

Figura 4: Simulação da contagem do contador 100 quando as dezenas é 0.

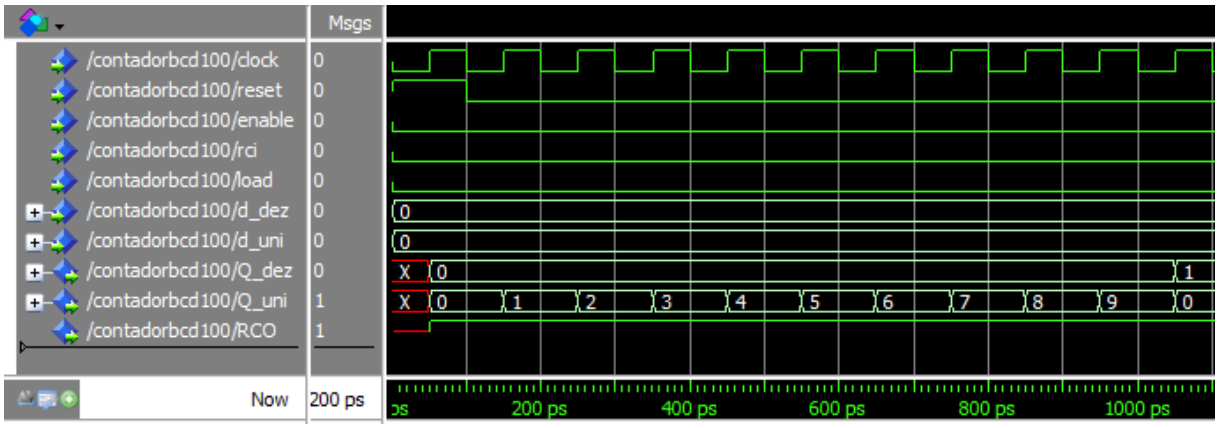


Figura 5: Simulação da contagem do contador 100 quando as dezenas é 1.

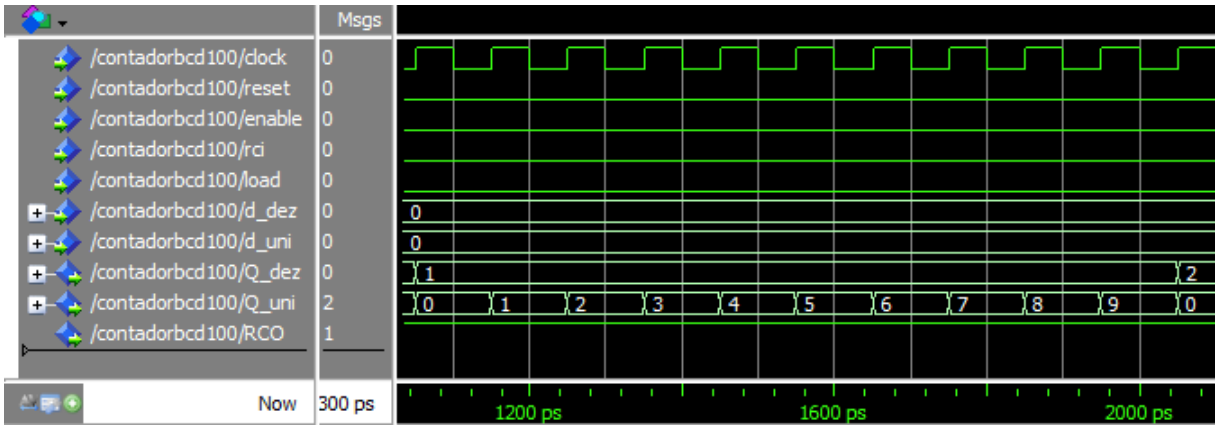


Figura 6: Simulação da contagem do contador 100 quando as dezenas é 2.

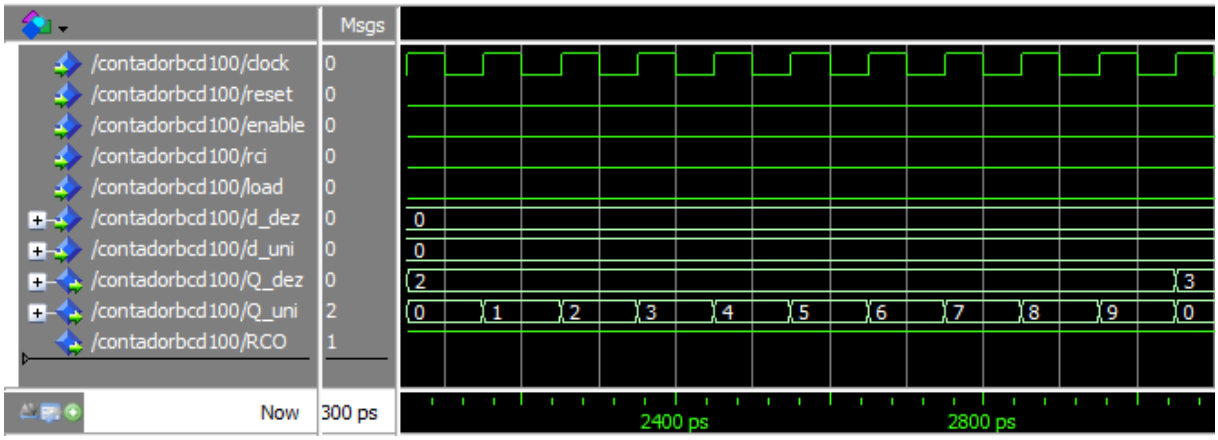


Figura 7: Simulação da contagem do contador 100 quando as dezenas é 3.

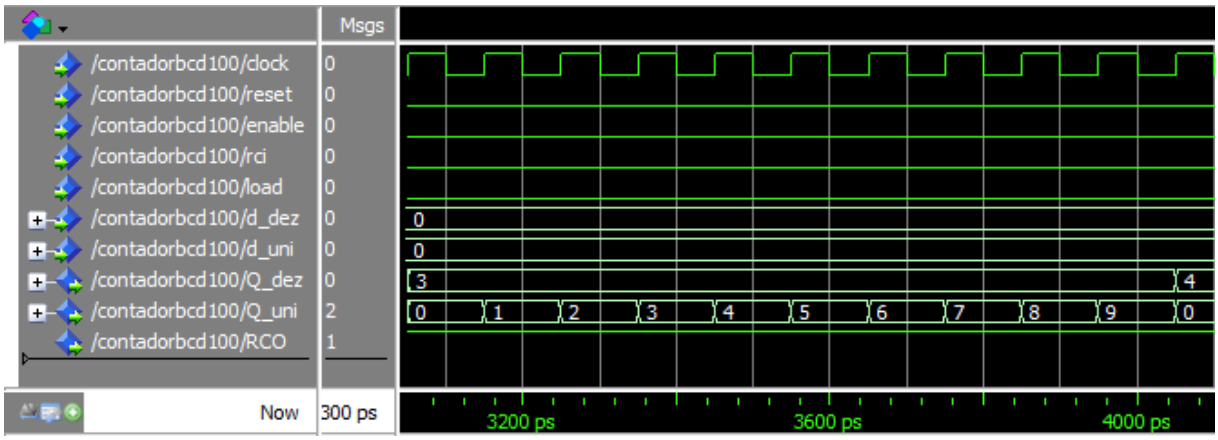


Figura 8: Simulação da contagem do contador 100 quando as dezenas é 4.

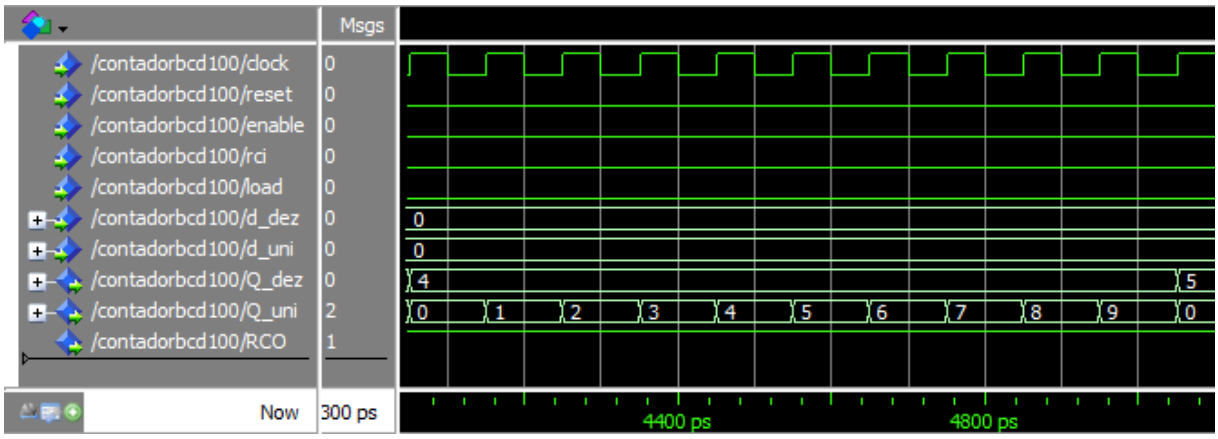


Figura 9: Simulação da contagem do contador 100 quando as dezenas é 5.

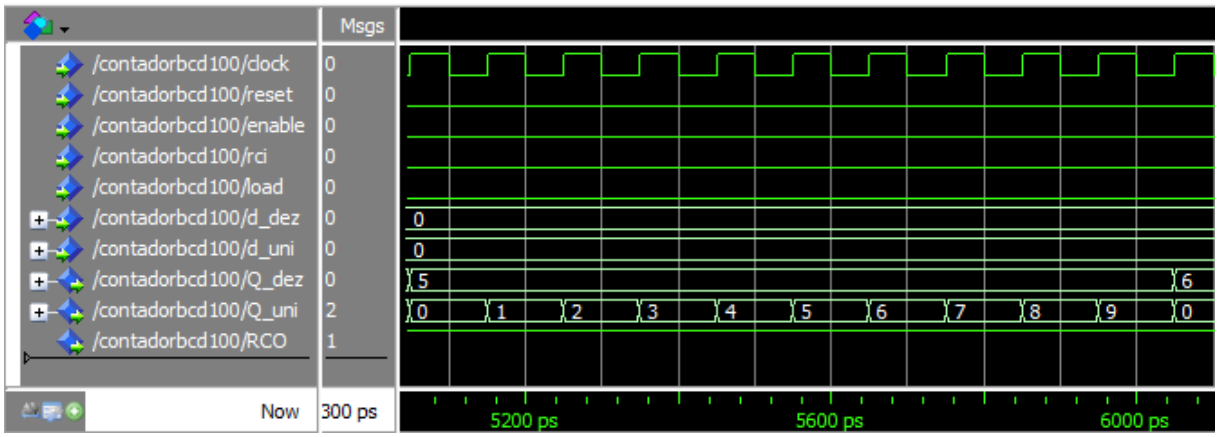


Figura 10: Simulação da contagem do contador 100 quando as dezenas é 6.

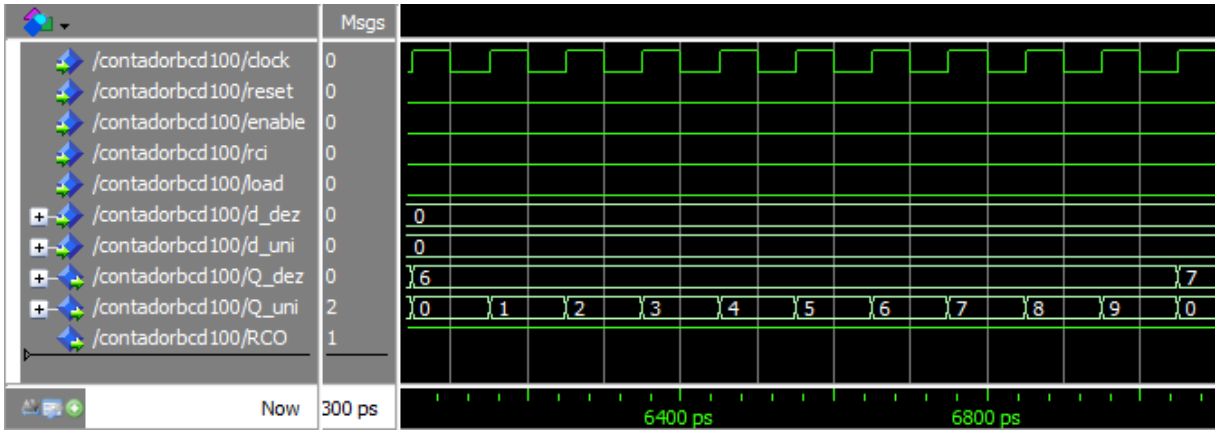


Figura 11: Simulação da contagem do contador 100 quando as dezenas é 7.

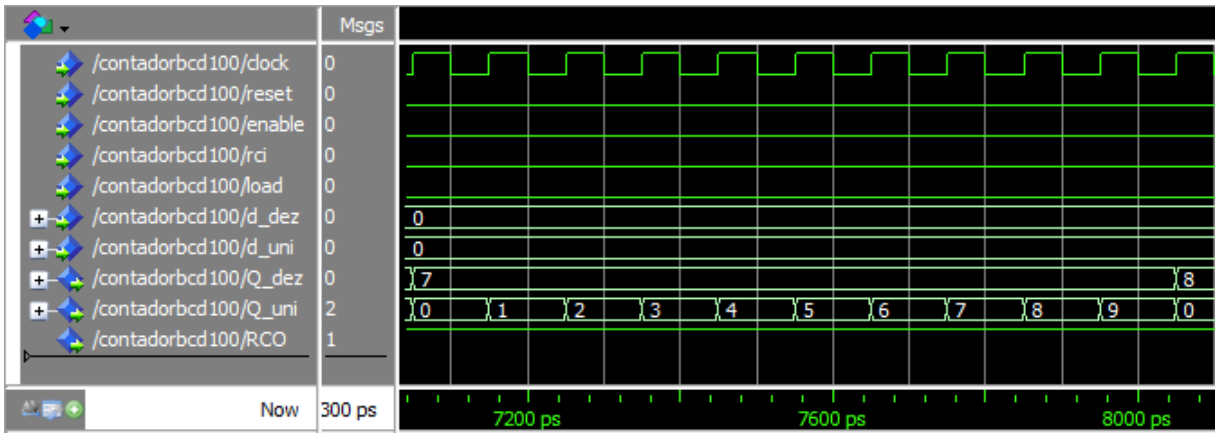


Figura 12: Simulação da contagem do contador 100 quando as dezenas é 8.

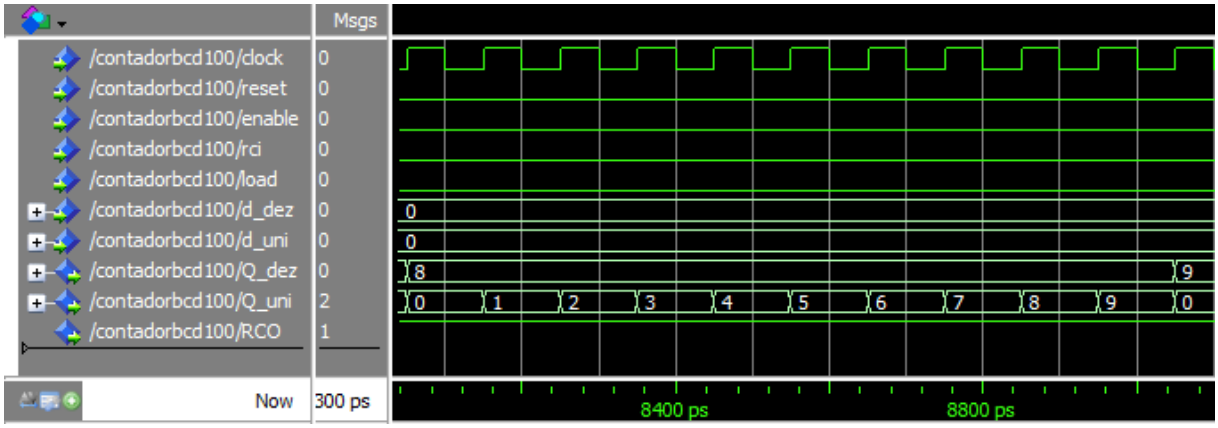
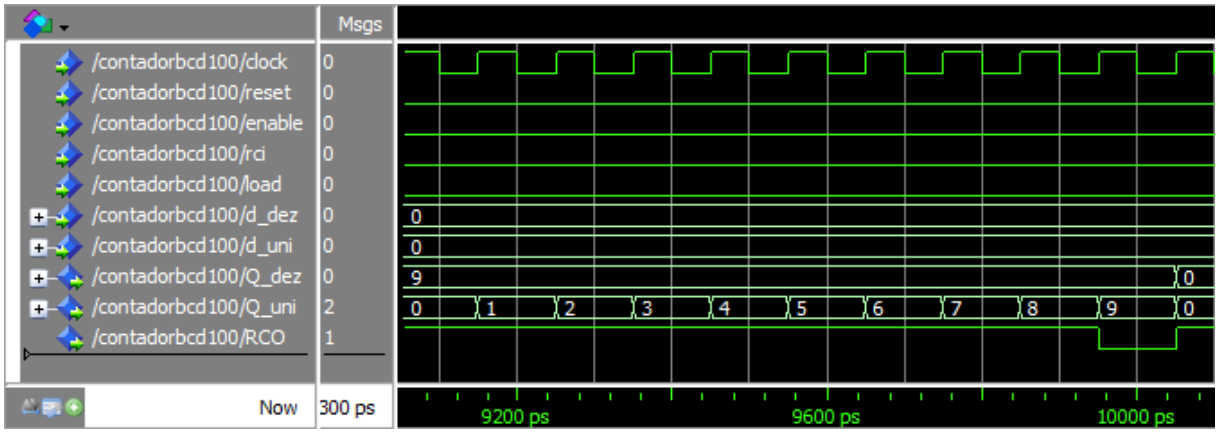
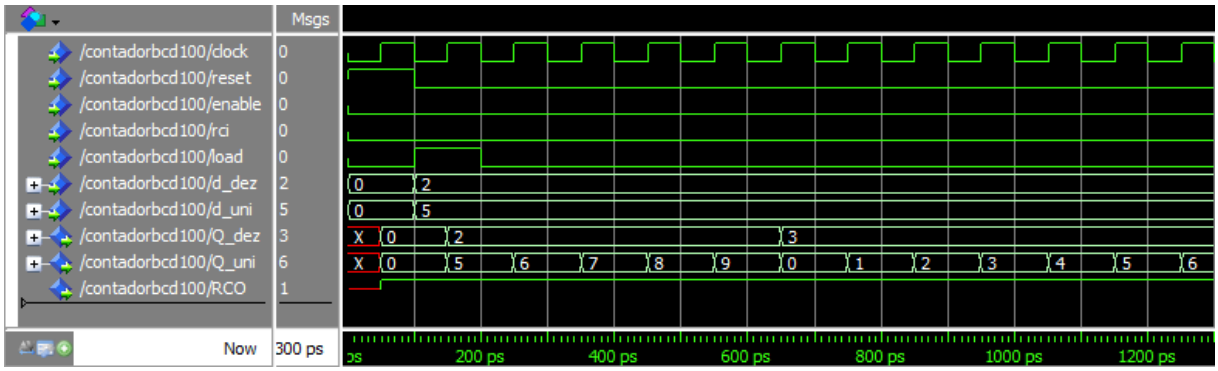


Figura 13: Simulação da contagem do contador 100 quando as dezenas é 9.



A simulação da funcionalidade `load`, na Figura 14 é semelhante ao teste realizado ateriormente, no contador de módulo 100 (Figura 2). Inicia-se com um pequeno pulso a `reset` e, depois, insere-se o estado 25. Verifica-se, com alguns ciclos do clock, que o contador continua em funcionamento normal após o carregamento.

Figura 14: Simulação do load do contador 100.



2.3 Extra

2.3.1 Compilação

Como segue na Figura 15, nenhum dos códigos obteve erros de compilação.

Figura 15: Compilação dos códigos.

```
# Compile of contadorBCD10.vhd was successful.
# Compile of contadorBCD100.vhd was successful.
# 2 compiles, 0 failed with no errors.
```

2.3.2 Códigos Diversos

Listagem 7: Código completo do contador BCD módulo 10.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity CONTADORBCD10 is
    port (
        clock: in std_logic;
        reset: in std_logic;
        enable: in std_logic;
        rci: in std_logic;
        load: in std_logic;
        d: in std_logic_vector(3 downto 0);
        Q: out std_logic_vector(3 downto 0);
        RCO: out std_logic
    );
end CONTADORBCD10;

architecture CONTADORBCD10_ARCH of CONTADORBCD10 is

    type estado is (ST0,ST1,ST2,ST3,ST4,ST5,ST6,ST7,ST8,ST9);

    signal currState, nextState : estado;
    signal nextQ : std_logic_vector(3 downto 0);
    signal nextRCO : std_logic;

begin

    sync_proc: process(clock)
    begin
        if rising_edge(clock) then
            currState <= nextState;
            Q <= nextQ;
            RCO <= nextRCO;
        end if;
```

```
    end process sync_proc;

    comb_proc: process(currState, reset, load, enable, rci)
    begin
        if (reset = '1') then
            nextState <= ST0;
            nextQ <= "0000";
            nextRCO <= '1';
        elsif (load = '1') then
            case (d) is
                when "0000" =>
                    nextState <= ST0;
                    nextQ <= "0000";
                    nextRCO <= '1';
                when "0001" =>
                    nextState <= ST1;
                    nextQ <= "0001";
                    nextRCO <= '1';
                when "0010" =>
                    nextState <= ST2;
                    nextQ <= "0010";
                    nextRCO <= '1';
                when "0011" =>
                    nextState <= ST3;
                    nextQ <= "0011";
                    nextRCO <= '1';
                when "0100" =>
                    nextState <= ST4;
                    nextQ <= "0100";
                    nextRCO <= '1';
                when "0101" =>
                    nextState <= ST5;
                    nextQ <= "0101";
                    nextRCO <= '1';
                when "0110" =>
                    nextState <= ST6;
                    nextQ <= "0110";
                    nextRCO <= '1';
                when "0111" =>
                    nextState <= ST7;
                    nextQ <= "0111";
                    nextRCO <= '1';
                when "1000" =>
                    nextState <= ST8;
                    nextQ <= "1000";
                    nextRCO <= '1';
                when "1001" =>
                    nextState <= ST9;
                    nextQ <= "1001";
                    nextRCO <= '0';
                when others =>
                    nextState <= ST0;
                    nextQ <= "0000";
                    nextRCO <= '1';
            end case;
        elsif (enable = '0') and (rci = '0') then
            case (currState) is
                when ST0 =>
                    nextState <= ST1;
                    nextQ <= "0001";
                    nextRCO <= '1';
                when ST1 =>
                    nextState <= ST2;
                    nextQ <= "0010";
                    nextRCO <= '1';
                when ST2 =>
                    nextState <= ST3;
                    nextQ <= "0011";
                    nextRCO <= '1';
                when ST3 =>
                    nextQ <= "0100";
                    nextRCO <= '1';
                    nextState <= ST4;
```

```
                when ST4 =>
                    nextState <= ST5;
                    nextQ <= "0101";
                    nextRCO <= '1';
                when ST5 =>
                    nextState <= ST6;
                    nextQ <= "0110";
                    nextRCO <= '1';
                when ST6 =>
                    nextState <= ST7;
                    nextQ <= "0111";
                    nextRCO <= '1';
                when ST7 =>
                    nextState <= ST8;
                    nextQ <= "1000";
                    nextRCO <= '1';
                when ST8 =>
                    nextState <= ST9;
                    nextQ <= "1001";
                    nextRCO <= '0';
                when ST9 =>
                    nextState <= ST0;
                    nextQ <= "0000";
                    nextRCO <= '1';
                when others =>
                    nextState <= ST0;
                    nextQ <= "0000";
                    nextRCO <= '1';
            end case;
        end if;
    end process comb_proc;
end CONTADORBCD10_ARCH;
```

Listagem 8: Código completo do contador BCD módulo 100.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity CONTADORBCD100 is
    port (
        clock: in std_logic;
        reset: in std_logic;
        enable: in std_logic;
        rci: in std_logic;
        load: in std_logic;
        d_uni: in std_logic_vector(3 downto 0);
        d_dez: in std_logic_vector(3 downto 0);
        Q_uni: out std_logic_vector(3 downto 0);
        Q_dez: out std_logic_vector(3 downto 0);
        RCO: out std_logic
    );
end CONTADORBCD100;

architecture CONTADORBCD100_ARCH of CONTADORBCD100 is
    component CONTADORBCD10 is
        port (
            clock: in std_logic;
            reset: in std_logic;
            enable: in std_logic;
            rci: in std_logic;
            load: in std_logic;
            d: in std_logic_vector(3 downto 0);
            Q: out std_logic_vector(3 downto 0);
            RCO: out std_logic
        );
    end component;

    signal uRCO, dRCO : std_logic;

begin
    INT1 : CONTADORBCD10 port map(clock, reset, enable, rci, load, d_uni, Q_uni, uRCO);
    INT2 : CONTADORBCD10 port map(clock, reset, enable, uRCO, load, d_dez, Q_dez, dRCO);

    RCO <= uRCO or dRCO;
```

```

end CONTADORBCD100_ARCH;
```

Listagem 9: Comandos utilizados durante as simulações.

```
// teste 10 contagem
force -freeze sim:/contadorbcd10/clock 0 0, 1 {50 ps} -r 100
force -freeze sim:/contadorbcd10/reset 1 0
force -freeze sim:/contadorbcd10/enable 0 0
force -freeze sim:/contadorbcd10/rci 0 0
force -freeze sim:/contadorbcd10/load 0 0
force -freeze sim:/contadorbcd10/d 0000 0
run 100ps
noforce sim:/contadorbcd10/reset
force -freeze sim:/contadorbcd10/reset 0 0
run 1100ps



// teste 10 load
force -freeze sim:/contadorbcd10/clock 0 0, 1 {50 ps} -r 100
force -freeze sim:/contadorbcd10/reset 1 0
force -freeze sim:/contadorbcd10/enable 0 0
force -freeze sim:/contadorbcd10/rci 0 0
force -freeze sim:/contadorbcd10/load 0 0
force -freeze sim:/contadorbcd10/d 0000 0
run 100ps

noforce sim:/contadorbcd10/reset
force -freeze sim:/contadorbcd10/reset 0 0
run 100ps

noforce sim:/contadorbcd10/load
noforce sim:/contadorbcd10/d
force -freeze sim:/contadorbcd10/load 1 0
force -freeze sim:/contadorbcd10/d 0110 0
run 100ps

noforce sim:/contadorbcd10/load
force -freeze sim:/contadorbcd10/load 0 0
run 500ps



// teste 10 reset
force -freeze sim:/contadorbcd10/clock 0 0, 1 {50 ps} -r 100
force -freeze sim:/contadorbcd10/reset 1 0
force -freeze sim:/contadorbcd10/enable 0 0
force -freeze sim:/contadorbcd10/rci 0 0
force -freeze sim:/contadorbcd10/load 0 0
force -freeze sim:/contadorbcd10/d 0000 0
run 100ps

noforce sim:/contadorbcd10/reset
force -freeze sim:/contadorbcd10/reset 0 0
run 100ps

noforce sim:/contadorbcd10/load
noforce sim:/contadorbcd10/d
force -freeze sim:/contadorbcd10/load 1 0
force -freeze sim:/contadorbcd10/d 0100 0
run 100ps
noforce sim:/contadorbcd10/load
force -freeze sim:/contadorbcd10/load 0 0
run 200ps

noforce sim:/contadorbcd10/reset
force -freeze sim:/contadorbcd10/reset 1 0
run 100ps
noforce sim:/contadorbcd10/reset
force -freeze sim:/contadorbcd10/reset 0 0
run 200ps


```

```
70
71 // teste 100 contagem
72 force -freeze sim:/contadorbcd100/clock 0 0, 1 {50 ps} -r 100
73 force -freeze sim:/contadorbcd100/reset 1 0
74 force -freeze sim:/contadorbcd100/enable 0 0
75 force -freeze sim:/contadorbcd100/rci 0 0
76 force -freeze sim:/contadorbcd100/load 0 0
77 force -freeze sim:/contadorbcd100/d_uni 0000 0
78 force -freeze sim:/contadorbcd100/d_dez 0000 0
79 run 100ps
80 noforce sim:/contadorbcd100/reset
81 force -freeze sim:/contadorbcd100/reset 0 0
82 run 10100ps
83
84
85 // teste 100 load
86 force -freeze sim:/contadorbcd100/clock 0 0, 1 {50 ps} -r 100
87 force -freeze sim:/contadorbcd100/reset 1 0
88 force -freeze sim:/contadorbcd100/enable 0 0
89 force -freeze sim:/contadorbcd100/rci 0 0
90 force -freeze sim:/contadorbcd100/load 0 0
91 force -freeze sim:/contadorbcd100/d_uni 0000 0
92 force -freeze sim:/contadorbcd100/d_dez 0000 0
93 run 100ps
94 noforce sim:/contadorbcd100/reset
95 noforce sim:/contadorbcd100/load
96 noforce sim:/contadorbcd100/d_uni
97 noforce sim:/contadorbcd100/d_dez
98 force -freeze sim:/contadorbcd100/reset 0 0
99 force -freeze sim:/contadorbcd100/load 1 0
100 force -freeze sim:/contadorbcd100/d_uni 0101 0
101 force -freeze sim:/contadorbcd100/d_dez 0010 0
102 run 100ps
103 noforce sim:/contadorbcd100/load
104 force -freeze sim:/contadorbcd100/load 0 0
105 run 1100ps
```