Laboratório de Sistemas Digitais – 2025.1

Relatório 8 – Contadores BCD

João Pedro Fernandes Santos (222025342)

Turma 09

Introdução/Objetivos

Neste Experimento 8 de Sistemas Digitais, dedicamo-nos ao desenvolvimento e à análise de contadores BCD em VHDL, enfatizando duas etapas principais. Na primeira, projetamos um contador BCD módulo 10 como uma máquina de estados finita do tipo Moore, consolidando em um único bloco process as condições de reset síncrono, load paralelo e incremento sequencial, além da geração adequada do sinal de ripple-carry-out. Em seguida, estendemos esse design para implementar um contador BCD módulo 100, composto por duas instâncias em cascata do contador módulo 10, de modo que o sinal de ripple-carry-out da unidade alimente o estágio de dezena, assegurando a contagem coordenada de unidades e dezenas.

Para validar o comportamento funcional desses circuitos, construiremos um testbench no ModelSim capaz de exercitar todas as transições de estado e as operações de controle, comparando automaticamente os resultados esperados com as saídas simuladas e permitindo a inspeção das formas de onda para confirmação da corretude. Com este experimento, buscamos consolidar a habilidade de projetar máquinas de estado finitas em VHDL, aplicar prioridades de controle (reset > load > contagem), compor módulos hierarquicamente, criar testbenches de cobertura completa e interpretar sinais de simulação para garantir a funcionalidade correta dos sistemas digitais desenvolvidos.

1. Atividade 1

1.1 Descrição da atividade

Nesta primeira atividade, desenvolvemos em VHDL e validamos em ModelSim um contador BCD de 0 a 9, seguindo os requisitos do roteiro. Os principais pontos são:

- Entity e Portas
    - clk : sinal de clock síncrono.
    - reset : reset síncrono, que força o contador a 0.
    - enable (ativo baixo): habilita a contagem quando '0'.
    - rci (ripple carry-in, ativo baixo): ativo em '0' para permitir incremento.

    - load : carrega simultaneamente o valor presente em D.
    - D (4 bits): valor paralelo de entrada para load.
    - Q (4 bits): saída que representa o valor atual do contador.
    - rco : ripple carry-out, que vai a '0' apenas quando o contador alcança 9.
- Máquina de Estados Moore
    - Estados: S0…S9 correspondendo aos valores 0…9.
    - loadState: gerado por um with D select, mapeando cada código binário em seu estado correspondente.
    - sync_proc: processo síncrono que, na borda de subida de clk, atualiza currentState ← nextState.
    - comb_proc: processo combinacional que, a cada mudança em currentState, reset, enable, rci ou load, define:
        1. Saídas (Q e rco) de acordo com currentState.
        2. Próximo estado (nextState) obedecendo à prioridade
            - Reset (reset = '1') → S0
            - Load (load = '1') → loadState
            - Contagem (enable='0' e rci='0') → estado sucessor circular (S9→S0)
            - Espera (caso contrário) → mantém currentState.
- Validação em ModelSim
    - Construção de um testbench que aplica sequências de:
        - Pulso de clk para percorrer todos os estados.
        - Ativação de reset e load em diferentes instantes.
        - Variação de enable e rci para testar a contagem e o bloqueio.
    - Verificação automática: comparação entre o valor esperado e a saída simulada.
    - Análise das formas de onda: confirmação visual das transições de 0→1, do carregamento paralelo e da geração correta de rco.

Com isso, garantimos que o módulo bcd_counter10 atende aos requisitos funcionais de um contador BCD síncrono com controle de load e ripple carry, servindo de base para a composição em estágios do contador módulo 100.

1.2 Implementação em VHDL

A Listagem 1 e 2 apresentam o código criado para a execução desta atividade.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bcd_counter10 is
 port (
  clk   : in  STD_LOGIC;
  reset  : in  STD_LOGIC;
  enable : in  STD_LOGIC;
  rci   : in  STD_LOGIC;  -- ripple carry-in
  load   : in  STD_LOGIC;
  D     : in  STD_LOGIC_VECTOR(3 downto 0);
  Q     : out STD_LOGIC_VECTOR(3 downto 0);
  rco   : out STD_LOGIC     -- ripple carry-out: '0' em
9, '1' caso contrário );
end bcd_counter10;
architecture arch of bcd_counter10 is
 type state_type is (S0, S1, S2, S3, S4, S5, S6, S7,
S8, S9);
 signal currentState, nextState, loadState :
state_type;
begin
with D select
  loadState <= S0 when "0000",  S1 when "0001",  S2
when "0010",S3 when "0011",  S4 when "0100",
 S5 when "0101",S6 when "0110",S7 when "0111", S8
when "1000", S9 when "1001",
    S0 when others;
 sync_proc: process(clk)
 begin
  if rising_edge(clk) then
   currentState <= nextState;
  end if;
end process sync_proc;
```

```
comb_proc: process(currentState, reset, enable,
rci, load, loadState)
 begin
  case currentState is
   when S0 => Q <= "0000"; rco <= '1';
   when S1 => Q <= "0001"; rco <= '1';
   when S2 => Q <= "0010"; rco <= '1';
   when S3 => Q <= "0011"; rco <= '1';
   when S4 => Q <= "0100"; rco <= '1';
   when S5 => Q <= "0101"; rco <= '1';
   when S6 => Q <= "0110"; rco <= '1';
   when S7 => Q <= "0111"; rco <= '1';
   when S8 => Q <= "1000"; rco <= '1';
   when S9 => Q <= "1001"; rco <= '0'; end case;
if reset = '1' then
   nextState <= S0;
  elsif load = '1' thennextState <= loadState;
  elsif (enable = '0') and (rci = '0') then
   case currentState is
    when S0 => nextState <= S1; when S1 =>
nextState <= S2; when S2 => nextState <= S3; when
S3 => nextState <= S4; when S4 => nextState <= S5;
when S5 => nextState <= S6; when S6 => nextState
<= S7; when S7 => nextState <= S8;
     when S8 => nextState <= S9;
     when S9 => nextState <= S0; end case;
  else
   nextState <= currentState;
  end if;
end process comb_proc;
end architecture arch;
```

Listagem 1: Implementação do circuito Contador BCD 10 em Vhdl.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY TB_BCD_COUNTER10 IS
END ENTITY TB_BCD_COUNTER10;
ARCHITECTURE BEHAVIOR OF TB_BCD_COUNTER10 IS
COMPONENT BCD_COUNTER10
  PORT (CLK   : 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 CLK_TB   : STD_LOGIC := '0';
 SIGNAL RESET_TB  : STD_LOGIC := '0';
 SIGNAL ENABLE_TB : STD_LOGIC := '1';  -- INIBE
CONTAGEM
 SIGNAL RCI_TB   : STD_LOGIC := '1';  -- INIBE RIPPLE IN
 SIGNAL LOAD_TB  : STD_LOGIC := '0';
 SIGNAL D_TB     : STD_LOGIC_VECTOR(3 DOWNTO 0)
:= (OTHERS => '0');
 SIGNAL Q_TB     : STD_LOGIC_VECTOR(3 DOWNTO 0);
 SIGNAL RCO_TB   : STD_LOGIC;
CONSTANT CLK_PERIOD : TIME := 10 NS;
BEGIN
UUT: BCD_COUNTER10
  PORT MAP (CLK   => CLK_TB,
    RESET  => RESET_TB,
    ENABLE => ENABLE_TB,
    RCI    => RCI_TB,
    LOAD   => LOAD_TB,  D     => D_TB,  Q     =>
Q_TB,
    RCO    => RCO_TB );
 CLK_PROC: PROCESS
 BEGIN
  LOOP
   CLK_TB <= '0';
   WAIT FOR CLK_PERIOD/2;
   CLK_TB <= '1';
   WAIT FOR CLK_PERIOD/2;
  END LOOP;
 END PROCESS CLK_PROC;
STIM_PROC: PROCESS
 BEGIN
  -- Reset Síncrono
  RESET_TB <= '1';
  WAIT FOR CLK_PERIOD;
  RESET_TB <= '0';
  WAIT FOR CLK_PERIOD;

  -- Teste de Load Paralelo
  D_TB   <= "0101";  -- CARREGA 5
  LOAD_TB <= '1';
  WAIT FOR CLK_PERIOD;
  LOAD_TB <= '0';
  WAIT FOR CLK_PERIOD;
ENABLE_TB <= '0';  -- ATIVA CONTAGEM
  RCI_TB   <= '0';  -- HABILITA RIPPLE
FOR I IN 0 TO 15 LOOP
   WAIT FOR CLK_PERIOD;
  END LOOP;
ENABLE_TB <= '1'; RCI_TB   <= '1';
WAIT FOR 5 * CLK_PERIOD;
  WAIT;  -- Fim da simulação
 END PROCESS STIM_PROC;

END ARCHITECTURE BEHAVIOR;
```

Listagem 2: Implementação do circuito do Testbench para o Contador BCD 10 em Vhdl.

O código de bcd_counter10 inicia com a declaração da entidade, expondo as entradas clk, reset, enable (ativo em ‘0’), rci (ripple carry-in, ativo em ‘0’), load e o vetor de dados D(3 downto 0), além das saídas Q(3 downto 0) e rco. Essa interface reflete exatamente as necessidades de um contador BCD: sinal de clock para sincronismo, reset síncrono para retornar a zero, controle de habilitação de contagem e de cascata, e a capacidade de carregar um valor paralelo. Cada porta tem nome e polaridade escolhidos para deixar explícita sua função no circuito.

Na arquitetura, definimos um tipo state_type com dez símbolos (S0…S9) que representam cada valor decimal. O sinal loadState é calculado por um with D select, mapeando diretamente cada código binário ao seu estado correspondente, o que simplifica o carregamento paralelo. Em seguida, o processo síncrono sync_proc, sensível apenas a clk, atualiza currentState na borda de subida, garantindo comportamento determinístico. Complementarmente, o processo combinacional comb_proc — sensível a currentState, reset, enable, rci, load e loadState — primeiro atribui as saídas Q e rco com base no estado atual (Moore), depois calcula nextState seguindo a hierarquia de prioridades: primeiro reset, depois load, em seguida contagem (quando enable='0' e rci='0') e, por fim, manutenção de estado.

O testbench define uma entidade sem portas e sua arquitetura behavior, redeclarando o componente bcd_counter10 e criando sinais de estímulo para cada uma de suas portas. Um processo gera o clock de 10 ns em loop, enquanto outro aplica, em sequência, um reset síncrono, um carregamento paralelo (por exemplo, “0101”), e então habilita enable e rci para percorrer o contador através de mais de dez ciclos, observando o retorno a zero e a geração de rco. Essa abordagem automatizada produz formas de onda reprodutíveis que cobrem reset, load, contagem e ripple-carry, permitindo verificar a corretude sem intervenção manual.

1.3 Simulação

Para verificar o funcionamento do bcd_counter10, o testbench inicializa um clock periódico de 10 ns (5 ns em ‘0’, 5 ns em ‘1’) gerado em loop. Logo no começo, aplica-se um reset síncrono (RESET_TB = '1') durante um ciclo completo de clock e retorna-se a ‘0’, garantindo que o contador parte de “0000” (S0). Em seguida, faz-se um carregamento paralelo (LOAD_TB = '1') com o vetor D_TB definido para “0101” durante um ciclo de clock, e volta-se LOAD_TB a ‘0’, posicionando o estado inicial em 5 para demonstrar o load.

Na fase de contagem, ativam-se simultaneamente ENABLE_TB = '0' e RCI_TB = '0', liberando tanto o incremento síncrono quanto o ripple-carry. O process de estímulo então itera por 16 ciclos de clock usando um loop FOR I IN 0 TO 15, o que faz o contador avançar de 5→6→...→9→0→1→... até cobrir toda a sequência esperada. Durante essa etapa, observamos em ModelSim não apenas a progressão de 0 a 9, mas também a transição de

rco para ‘0’ exclusivamente ao passar de “1001” (9) para “0000” (0), demonstrando o ripple-carry-out correto.

Por fim, o testbench desativa ENABLE_TB e RCI_TB retornando-os a ‘1’, mantém o contador estável por mais cinco ciclos de clock e, em seguida, executa um WAIT; para encerrar a simulação. No visor de formas de onda, fica evidente que todas as operações de reset, load e contagem circular foram exercitadas automaticamente, confirmando que o módulo acata a prioridade de controle (reset > load > contagem > espera) e que o sinal rco é gerado adequadamente.

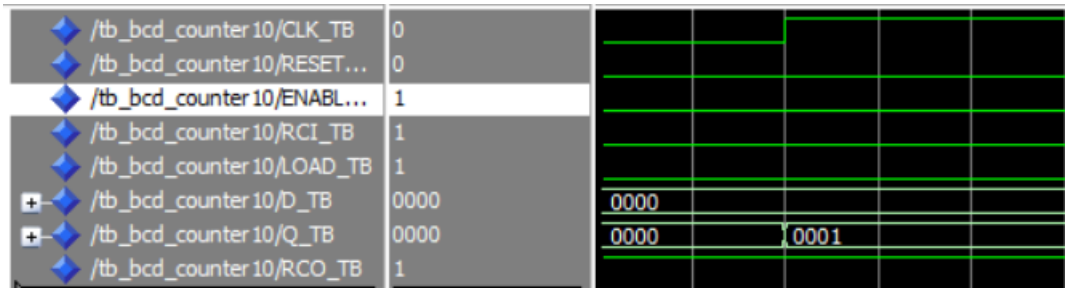

Figura 1: Simulação do Contador BCD 10, na subida do sinal de clock, é possível observar que o valor da varável Q se altera de 0000 (0) para 0001 (1).

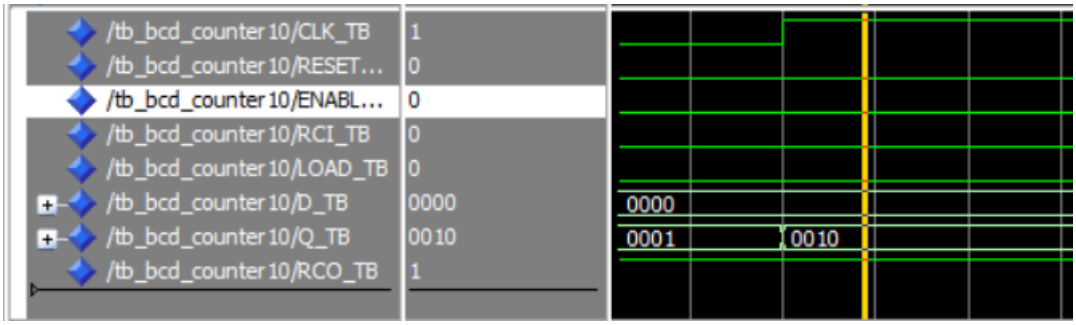

Figura 2: Simulação do Contador BCD 10, na subida do sinal de clock, é possível observar que o valor da varável Q se altera de 0001 (1) para 0010 (2).

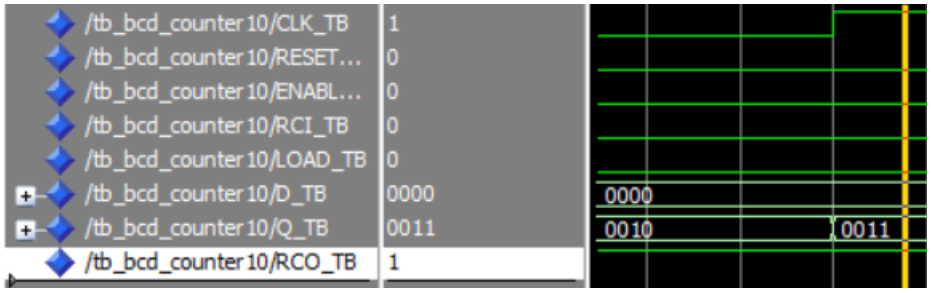

Figura 3: Simulação do Contador BCD 10, na subida do sinal de clock, é possível observar que o valor da varável Q se altera de 00010 (2) para 0011 (3), e assim, a simulação contua até que Q assuma o valor de 1001 (9).

2. Atividade 2

2.1 Descrição da atividade

esta segunda atividade, estendemos o design do contador BCD módulo 10 para construir um contador de 0 a 99, integrando dois estágios em cascata. Os principais aspectos são:

- Entity e Portas
    - clk e reset: compartilhados entre os dois estágios, garantindo sincronismo e reset simultâneo.
    - enable (ativo baixo) e load: controles globais que afetam ambos os dígitos.
    - D_unidade e D_dezena (cada um 4 bits): valores paralelos de entrada para o carregamento de unidades e dezenas.
    - Q_unidade e Q_dezena (cada um 4 bits): saídas representando o valor atual de cada dígito.
- Composição Hierárquica
    - Dois componentes bcd_counter10 são instanciados:
        1. Unidade (U_UNIT): recebe rci='0' para sempre poder contar em nível de unidade.
        2. Dezena (U_DEZ): recebe rci igual ao rco do estágio de unidade, de modo que, quando a unidade transita de 9→0, dispara a contagem da dezena.
- Fluxo de Controle
    - Reset: se reset='1', ambos os estágios retornam a zero simultaneamente.
    - Load: quando load='1', cada contador carrega seu respectivo valor em D_unidade ou D_dezena.
    - Contagem: ativada com enable='0'; a unidade conta de 0→9 e, ao retornar a 0, gera rco='0', permitindo que a dezena incremente um passo.
    - Bloqueio: se enable='1', nenhum estágio avança, mantendo o valor atual.
- Validação em ModelSim
    - Testbench aplica:
        1. Reset síncrono de todo o contador 100.

2. Load paralelo para verificar Q_unidade←D_unidade e Q_dezena←D_dezena.
3. Contagem contínua para observar a cascata: unidade de 0→9→0 e dezena incrementando apropriadamente.

- Conferência das formas de onda para cada transição de unidade e dezena, garantindo que a lógica de ripple-carry esteja corretamente implementada.

Com essa atividade, consolidamos a habilidade de compor módulos VHDL hierarquicamente, reutilizando componentes bem testados e garantindo a correta interdependência de estágios em designs de maior escala.

2.2 Implementação em VHDL

A Listagem 3 e 4 apresentam o código criado para esta atividade.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bcd_counter100 is
 port (
  clk     : in  STD_LOGIC;
  reset    : in  STD_LOGIC;
  enable   : in  STD_LOGIC;  -- conta quando '0'
  load     : in  STD_LOGIC;
  D_unidade : in  STD_LOGIC_VECTOR(3 downto 0);
  D_dezena  : in  STD_LOGIC_VECTOR(3 downto 0);
  Q_unidade : out STD_LOGIC_VECTOR(3 downto 0);
  Q_dezena  : out STD_LOGIC_VECTOR(3 downto 0));
end bcd_counter100;
architecture arch of bcd_counter100 is
 signal rco_unidade, rco_dezena : STD_LOGIC;
begin
U_UNIT: entity work.bcd_counter10
  port map (clk   => clk, reset  => reset,
   enable => enable,
```

```
   rci   => '0',       -- sempre habilitado para unidade
   load   => load,
   D     => D_unidade,
   Q     => Q_unidade,
   rco    => rco_unidade );
 -- componente dezena, recebe rco_unidade como
rci
 U_DEZ: entity work.bcd_counter10
  port map (
   clk   => clk,
   reset  => reset,
   enable => enable,
   rci   => rco_unidade,  -- cascata
   load   => load,
   D     => D_dezena,
   Q     => Q_dezena,
   rco    => rco_dezena
  );

end architecture arch;
```

Listagem 3: Implementação do circuito Contador BCD 100 em Vhdl.

```
IBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;


ENTITY TB_BCD_COUNTER100 IS
END ENTITY TB_BCD_COUNTER100;
ARCHITECTURE BEHAVIOR OF TB_BCD_COUNTER100 IS
COMPONENT BCD_COUNTER100
  PORT (
    CLK     : IN  STD_LOGIC;
    RESET   : IN  STD_LOGIC;
    ENABLE  : IN  STD_LOGIC;
    LOAD    : IN  STD_LOGIC;
    D_UNIDADE : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    D_DEZENA  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    Q_UNIDADE : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    Q_DEZENA  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)  );
 END COMPONENT;
SIGNAL CLK_TB      : STD_LOGIC := '0';
 SIGNAL RESET_TB    : STD_LOGIC := '0';
 SIGNAL ENABLE_TB   : STD_LOGIC := '1';  -- INIBE CONTAGEM
 SIGNAL LOAD_TB     : STD_LOGIC := '0';
 SIGNAL D_UNIDADE_TB  : STD_LOGIC_VECTOR(3 DOWNTO 0)
:= (OTHERS => '0');
 SIGNAL D_DEZENA_TB   : STD_LOGIC_VECTOR(3 DOWNTO 0)
:= (OTHERS => '0');
 SIGNAL Q_UNIDADE_TB  : STD_LOGIC_VECTOR(3 DOWNTO
0);
 SIGNAL Q_DEZENA_TB   : STD_LOGIC_VECTOR(3 DOWNTO 0);
CONSTANT CLK_PERIOD : TIME := 10 NS;
BEGIN
UUT: BCD_COUNTER100
  PORT MAP (CLK     => CLK_TB, RESET    => RESET_TB,
    ENABLE  => ENABLE_TB, LOAD    => LOAD_TB,
    D_UNIDADE => D_UNIDADE_TB,
    D_DEZENA  => D_DEZENA_TB,
    Q_UNIDADE => Q_UNIDADE_TB,
    Q_DEZENA  => Q_DEZENA_TB  );
CLK_PROC: PROCESS
 BEGIN
  LOOP
    CLK_TB <= '0';
    WAIT FOR CLK_PERIOD/2;
    CLK_TB <= '1';
    WAIT FOR CLK_PERIOD/2;
  END LOOP;
 END PROCESS CLK_PROC;
STIM_PROC: PROCESS
 BEGIN
RESET_TB <= '1';
  WAIT FOR CLK_PERIOD;
  RESET_TB <= '0';
  WAIT FOR CLK_PERIOD;
D_UNIDADE_TB <= "0011";  -- UNIDADE = 3
  D_DEZENA_TB  <= "0100";  -- DEZENA  = 4
  LOAD_TB    <= '1';
  WAIT FOR CLK_PERIOD;
  LOAD_TB    <= '0';
  WAIT FOR CLK_PERIOD;
ENABLE_TB <= '0';
FOR I IN 0 TO 25 LOOP
    WAIT FOR CLK_PERIOD;
  END LOOP;
ENABLE_TB <= '1';
WAIT FOR 5 * CLK_PERIOD;
  WAIT;  -- Fim da simulação
 END PROCESS STIM_PROC;


END ARCHITECTURE BEHAVIOR;
```

Listagem 4: Implementação do circuito do Testbench para o Contador BCD 100 em Vhdl.

O módulo bcd_counter100 é definido por uma entidade que expõe as entradas clk, reset, enable (ativo em ‘0’), load, e dois vetores de dados paralelos D_unidade(3 downto 0) e D_dezena(3 downto 0), além das saídas Q_unidade(3 downto 0) e Q_dezena(3 downto 0). Internamente, dois sinais rco_unidade e rco_dezena propagam o ripple carry entre os estágios. A arquitetura simplesmente instancia dois componentes bcd_counter10: o primeiro (U_UNIT) sempre recebe rci='0' para contar unidades, o segundo (U_DEZ) usa rco_unidade como seu rci para incrementar a dezena quando a unidade transborda.

Cada instância compartilha clk, reset, enable e load, garantindo que ambos os dígitos sejam resetados ou carregados em paralelo quando solicitado. No modo load (load='1'), cada estágio carrega o valor de D_unidade ou D_dezena via seu with-select. No modo contagem (enable='0'), a unidade avança de 0 a 9 produzindo rco_unidade='0' ao passar por 9→0, e esse pulso faz a dezena avançar um passo, respeitando o mesmo ciclo de controle.

O testbench redescreve BCD_COUNTER100, gera um clock de 10 ns em loop e aplica um estímulo ordenado: primeiro um reset síncrono de ambos os dígitos, depois um load paralelo (ex.: unidade="0011", dezena="0100"), seguido de habilitação de contagem para percorrer pelo menos 25 ciclos, observando a cascata 00→99→00. Em seguida, desativa enable, aguarda alguns ciclos e encerra a simulação com WAIT, produzindo formas de onda que validam reset, load, contagem e interdependência correta entre unidades e dezenas.

2.3 Simulação

O módulo bcd_counter100 organiza-se em duas partes bem definidas: a entidade, que expõe as entradas clk, reset, enable (ativo em ‘0’), load e os vetores paralelos D_unidade(3 downto 0) e D_dezena(3 downto 0), além das saídas Q_unidade(3 downto 0) e Q_dezena(3 downto 0); e sua arquitetura comportamental, que simplesmente instancia dois componentes bcd_counter10. Dois sinais internos, rco_unidade e rco_dezena, propagam o ripple carry: o primeiro sempre recebe rci='0' para contar unidades, o segundo usa rco_unidade como seu rci, de modo que quando a unidade transborda (9→0) dispara a contagem da dezena.

Cada instância compartilha clk, reset, enable e load, garantindo reset e load simultâneos em ambas as dezenas e unidades. No modo load='1', cada contador carrega via seu próprio with D select o valor em D_unidade ou D_dezena. Quando enable='0', a unidade avança de 0 a 9 e, somente ao completar o ciclo (passagem de “1001” para “0000”), gera rco_unidade='0', que permite à dezena incrementar um passo, mantendo a mesma lógica de máquina de estado Moore do módulo 10 em cada estágio.

O testbench TB_BCD_COUNTER100 re-declara o componente, gera um clock de 10 ns em loop e aplica um estímulo ordenado: primeiro um reset síncrono de ambos os dígitos, depois um load paralelo (por exemplo, unidade="0000", dezena="0000"), em seguida habilita enable='0' para contar e observar a cascata de 00→99→00, e por fim desativa a contagem e aguarda alguns ciclos antes de WAIT;. Essa simulação automatizada valida reset, load, contagem coordenada e propagação correta do ripple carry entre os estágios.

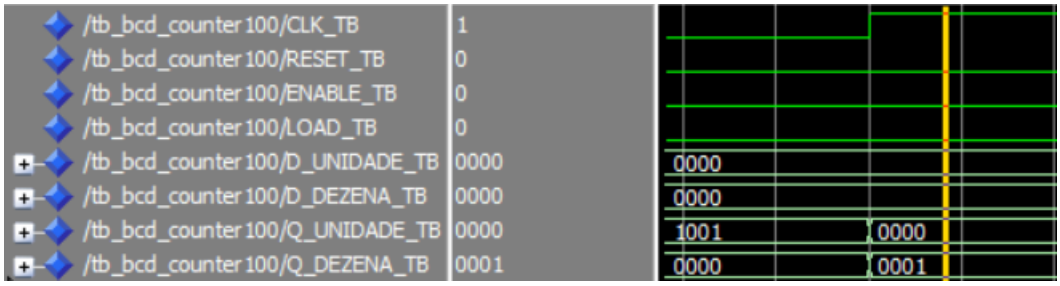

Figura 4: Simulação do Contador BCD 100, na subida do sinal de clock, é possível observar que o valor da varável Q_UNIDADE se altera de 1001 (9) para 0000 (0), e a variável Q_DEZENA se altera de 0000(0) para 0001 (1) demonstrando a mudança da contagem de 9 para 10 e continua de 1 em 1(unidade).

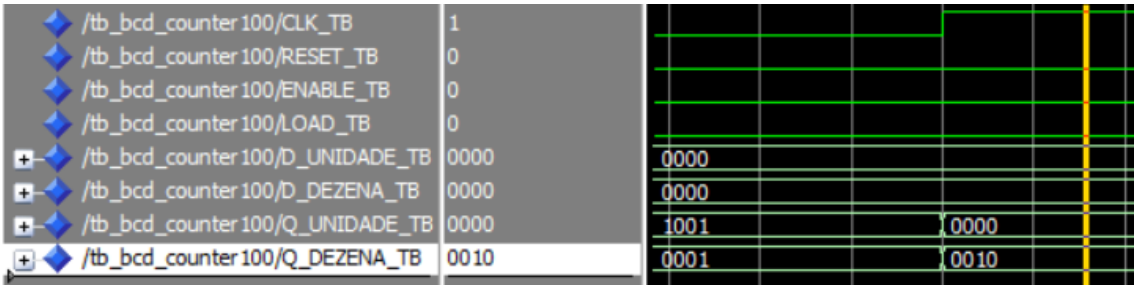

Figura 5: Simulação do Contador BCD 100, na subida do sinal de clock, é possível observar que o valor da varável Q_UNIDADE se altera de 1001 (9) para 0000 (0), e a variável Q_DEZENA se altera de 0001(0) para 0010 (2) demonstrando a mudança da contagem de 19 para 20, e assim, a simulação continua de unidade em unidade até que Q_UNIDADE e Q_DEZENA assumam o valor de 1001 (9), completando a contagem.