Laboratório de Sistemas Digitais – 2025.1

Relatório 6 – Circuitos Sequenciais

João Pedro Fernandes Santos (222025342)

Turma 09

Introdução/Objetivos

Neste Experimento 6 de Sistemas Digitais, desenvolvemos e analisamos circuitos sequenciais em VHDL a partir do uso estruturado de blocos process. Na primeira etapa, projetamos um flip-flop JK sensível à borda de subida do clock, implementado por meio de quatro condições dentro de um único process contemplando manutenção do estado (J=0/K=0), set (J=1/K=0), reset (J=0/K=1) e inversão (J=1/K=1). Em seguida, ampliamos nossa abordagem para construir um registrador de deslocamento bidirecional de 4 bits, que suporta operações de reset síncrono, load paralelo e deslocamento serial nas duas direções, respeitando a prioridade de controle (reset > load > shift). Durante o desenvolvimento, enfatizamos a codificação modular, e a clareza na definição dos sinais de controle.

Para validar o comportamento dos circuitos, elaboramos um testbench automatizado no ModelSim que gera combinações de entradas cobrindo todas as operações possíveis do registrador e do flip-flop. O testbench compara os resultados esperados com as saídas simuladas, sinalizando qualquer discrepância e permitindo inspeção detalhada das formas de onda. Com isso, exercitamos habilidades fundamentais: projetar e simular circuitos sequenciais em VHDL, manipular o fluxo de controle dentro de process, criar testbenches que garantam cobertura completa de casos de teste e interpretar os resultados de simulação para assegurar a correção funcional dos módulos.

1. Atividade 1

1.1 Descrição da atividade

A primeira atividade propôs, em VHDL e simulação no ModelSim, a implementação de um flip-flop tipo JK sensível à borda de subida do clock, conforme especificado no roteiro. O componente deve incluir entradas síncronas J, K, CLK, além de sinais de preset (PR) e clear (CLR), e produzir a saída Q de acordo com as funções lógicas da tabela-verdade:

- **PR = 1** → Q ← 1 (override set)
- **CLR = 1** → Q ← 0 (override reset)
- **na borda de subida de CLK** (PR = CLR = 0):

- J = 0, K = 0 → mantém o estado atual de Q
- J = 0, K = 1 → Q ← 0 (reset)
- J = 1, K = 0 → Q ← 1 (set)
- J=1, K=1 → Q ← ¬Q (toggle)
  Em todas as outras condições (fora da transição de 0→1 em CLK), as variáveis marcadas como "X" não afetam a saída.

Para exercitar conceitos de modelagem comportamental, a implementação deve usar um bloco process com lista de sensibilidade adequada e a detecção de borda de subida pelo "*rising_edge*"(clk). Ao final, espera-se, por meio da análise das formas de onda no ModelSim, confirmar que cada linha da tabela-verdade foi corretamente implementada, validando o funcionamento do flip-flop JK em todas as condições de entrada.

Tabela 1: Tabela-verdade do Flip-Flop JK:

PR	CLR	CLK	J	K	Q
1	X	X	X	X	1
0	1	X	X	X	0
0	0	0→1	0	0	Mantém
0	0	0→1	0	1	0
0	0	0→1	1	0	1
0	0	0→1	1	1	Inverte
0	0	0	X	X	Mantém

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 FLIPFLOP_JK IS
 PORT(
  PRESET  : IN  STD_LOGIC;
  CLEAR  : IN  STD_LOGIC;
  CLOCK  : IN  STD_LOGIC;
  J : IN  STD_LOGIC;
  K : IN  STD_LOGIC;
  SAIDA   : OUT STD_LOGIC );
END FLIPFLOP_JK;
ARCHITECTURE FLIPFLOP_JK_ARCH OF
FLIPFLOP_JK IS
 SIGNAL REG : STD_LOGIC := '0';
```

```
BEGIN
 SAIDA <= REG;
PROCESS(PRESET, CLEAR, CLOCK)
 BEGIN
  IF PRESET = '1' THEN
   REG <= '1';
  ELSIF CLEAR = '1' THEN
   REG <= '0';
  ELSIF RISING_EDGE(CLOCK) THEN
   IF J = '0' AND K = '0' THEN
    REG <= REG;
   ELSIF J = '0' AND K = '1' THEN
    REG <= '0';
   ELSIF J = '1' AND K = '0' THEN
```

```
      REG <= '1';
    ELSIF J = '1' AND K = '1' THEN
      REG <= NOT REG;
    END IF;
  END IF;
 END PROCESS;
END FLIPFLOP_JK_ARCH;
```

Listagem 1: Implementação do circuito Flip-Flop JK em Vhdl.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tb_flipflop_jk IS
END tb_flipflop_jk;
ARCHITECTURE behavior OF tb_flipflop_jk IS
 COMPONENT FLIPFLOP_JK
  PORT(
   PRESET : IN  STD_LOGIC;
   CLEAR  : IN  STD_LOGIC;
   CLOCK  : IN  STD_LOGIC;
   J      : IN  STD_LOGIC;
   K      : IN  STD_LOGIC;
   SAIDA  : OUT STD_LOGIC  );
 END COMPONENT;
SIGNAL PRESET_s, CLEAR_s, CLOCK_s, J_s,
K_s : STD_LOGIC := '0';
 SIGNAL SAIDA_s                  : STD_LOGIC;
BEGIN
 DUT: FLIPFLOP_JK
  PORT MAP (
   PRESET => PRESET_s,
   CLEAR  => CLEAR_s,
   CLOCK  => CLOCK_s,
   J      => J_s,
   K      => K_s,
   SAIDA  => SAIDA_s   );
```

```
 clk_gen: PROCESS
 BEGIN
  CLOCK_s <= '0';  WAIT FOR 5 ns;
  CLOCK_s <= '1';   WAIT FOR 5 ns;
 END PROCESS clk_gen;
stim_proc: PROCESS
 BEGIN
  PRESET_s <= '0';
  CLEAR_s  <= '1';
  J_s      <= '0';
  K_s      <= '0';
  WAIT FOR 20 ns;
  CLEAR_s <= '0';
  WAIT FOR 10 ns;
  WAIT UNTIL rising_edge(CLOCK_s);
  J_s <= '1'; K_s <= '0';
  WAIT UNTIL rising_edge(CLOCK_s);
  J_s <= '0'; K_s <= '1';
  WAIT UNTIL rising_edge(CLOCK_s);
  J_s <= '1'; K_s <= '1';
  WAIT UNTIL rising_edge(CLOCK_s);
  J_s <= '0'; K_s <= '0';
  PRESET_s <= '1';
  WAIT FOR 20 ns;
  PRESET_s <= '0';
  WAIT;
 END PROCESS stim_proc;
END behavior;
```

Listagem 2: Implementação do circuito do Testbench para Flip-Flop JK em Vhdl.

O código do **FLIPFLOP_JK** é organizado em duas partes principais: a declaração da entidade (entity) e sua implementação comportamental (architecture). A entidade FLIPFLOP_JK expõe cinco entradas: PRESET , CLEAR , CLOCK, J e K e uma saída SAIDA , todas do tipo STD_LOGIC. Dentro da arquitetura, declara-se um sinal interno REG que armazena o estado atual do flip-flop e que é atribuído à saída a cada ciclo. O bloco PROCESS (P
RESET , CLEAR , CLOCK) implementa primeiro os resets assíncronos: se PRESET = '1' Então força REG <= '1'; senão, se CLEAR = '1', força REG <= '0'. Caso nenhum dos dois esteja ativo, o PROCESS aguarda a borda de subida do relógio (rising_edge(CLOCK)) para decidir, com base nos valores de J e K, se mantém (J=0,K=0), zera (J=0,K=1), seta (J=1,K=0) ou inverte (J=1,K=1) o conteúdo de REG. Esse estilo concentra toda a lógica sequencial num único process, seguindo as boas práticas de modelagem comportamental em VHDL.

O **testbench** complementa o DUT (Device Under Test) fornecendo estímulos de forma automatizada e reprodutível. Ele começa com a declaração de uma entidade tb_flipflop_jk sem portas — pois não há interação externa — e sua arquitetura behavior, que inclui: a re-declaração do componente FLIPFLOP_JK para que possa ser instanciado; sinais internos (PRESET_s, CLEAR_s, CLOCK_s, J_s, K_s e SAIDA_s) que interligam o DUT ao testbench; e (3) a instância DUT: FLIPFLOP_JK PORT MAP(...) que faz o mapeamento de portas. Em seguida, o processo clk_gen gera um clock periódico de 10 ns de período, alternando CLOCK_s entre '0' e '1'. O processo stim_proc aplica, em sequência e com WAIT FOR, um conjunto de vetores de controle: primeiro o CLEAR assíncrono, depois as condições de set, reset, toggle e hold por combinações de J e K, e por fim o PRESET assíncrono. Cada etapa dura 10 ns, garantindo tempo suficiente para observar o resultado no ModelSim.

Juntos, esses dois blocos permitem que, ao se rodar no ModelSim apenas o testbench, visualizemos todas as transições de SAIDA conforme a tabela-verdade do flip-flop JK. A vantagem do testbench é automatizar a geração de estímulos, produzir formas de onda consistentes e facilitar a verificação de cobertura de todos os casos sem intervenção manual.

1.3 Simulação

Para verificar o comportamento do flip-flop JK, no testbench geramos um clock periódico de 10 ns de período (5 ns em ‘0’, 5 ns em ‘1’) e aplicamos sistematicamente todos os vetores de controle, alinhando-os às bordas de subida. Logo no início, mantivemos CLEAR_s = '1' por 20 ns (com PRESET_s, J_s e K_s em ‘0’), assegurando que o registrador interno partisse de ‘0’. Em seguida, voltamos CLEAR_s a ‘0’ e, em cada WAIT UNTIL rising_edge(CLOCK_s), mudamos apenas J_s e K_s para os quatro casos síncronos:

- **Set (J=1,K=0)** no primeiro rising edge,
- **Reset (J=0,K=1)** no segundo,
- **Toggle (J=1,K=1)** no terceiro,
- **Hold (J=0,K=0)** no quarto.

Por fim, aplicamos um PRESET_s = '1' por 20 ns, retornando depois a ‘0’.

Na janela de formas de onda do ModelSim, vê-se claramente:

- **0–20 ns**: CLEAR_s força SAIDA_s para ‘0’.
- **Rising edge #1**: com J/K = 0/0, SAIDA_s mantém o valor anterior (hold).
- **Rising edge #2**: com J/K = 1/0, SAIDA_s sobe para ‘1’ no mesmo ciclo de clock.
- **Rising edge #3**: com J/K = 0/1, SAIDA_s desce para ‘0’.
- **Rising edge #4**: com J/K = 1/1, SAIDA_s inverte (toggle).
- **~60–80 ns**: PRESET_s alto força SAIDA_s de volta a ‘1’.

Esse sequenciamento comprovou exaustivamente todos os modos de operação (clear, set, reset, toggle, hold e preset) exatamente na borda de subida do clock, sem necessidade de ajustes manuais na waveform, garantindo cobertura completa da tabela-verdade do flip-flop JK.

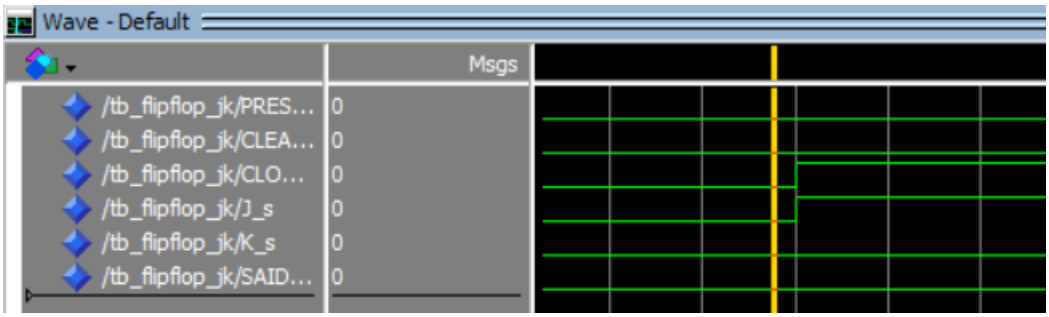

Figura 1: Simulação do FLIPFLOP_JK, no primeiro *Rising Edge #1,* é possível observar que para J/K=0/0 a saida após o clock é 0, mantendo seu valor anterior(HOLD) .

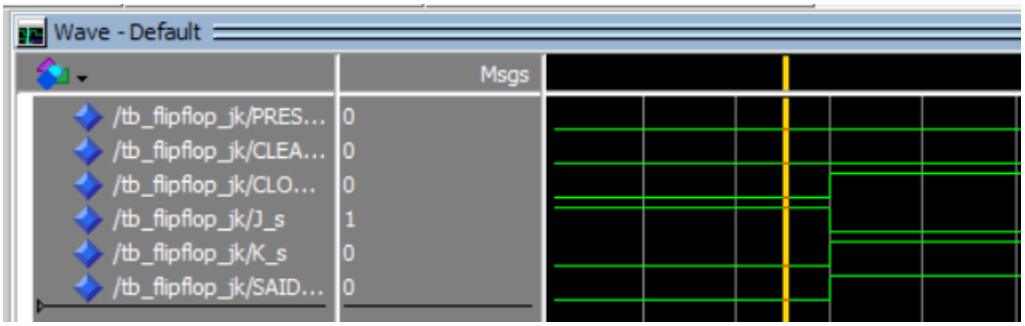

Figura 2: Simulação do FLIPFLOP_JK, no segundo *Rising Edge #2,* é possível observar que para J/K=1/0 a saida após o clock é 1.

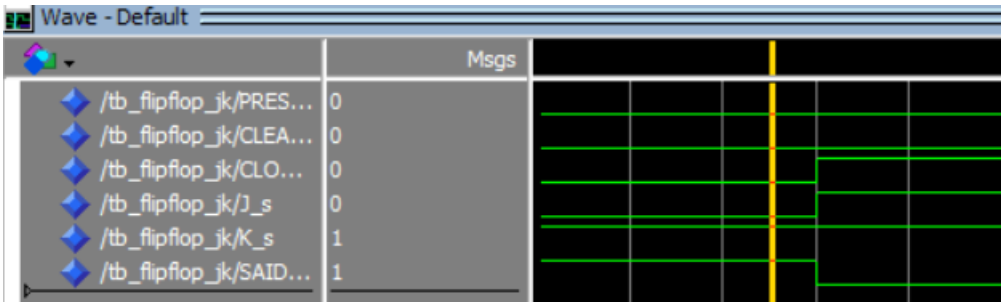

Figura 3: Simulação do FLIPFLOP_JK, no terceiro *Rising Edge #3,* é possível observar que para J/K=0/1 a saida após o clock é 0.

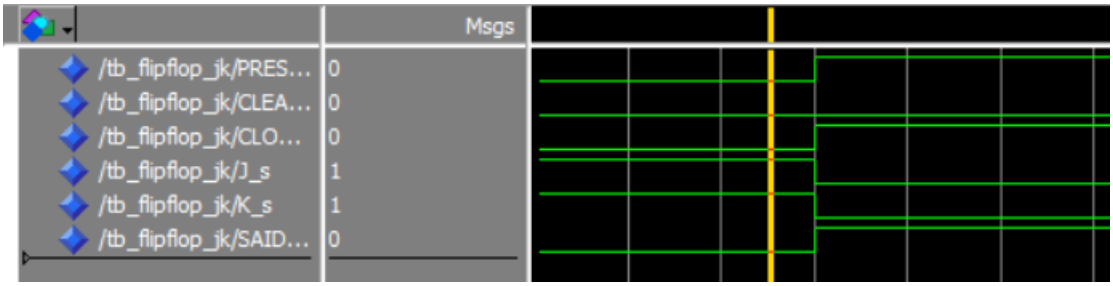

Figura 4: Simulação do FLIPFLOP_JK, no Quarto *Rising Edge #4,* é possível observar que para J/K=1/1 a saida após o clock é invertida (Toggle).

2. Atividade 2

2.1 Descrição da atividade

A segunda atividade propôs, em VHDL e simulação no ModelSim, a implementação de um registrador de deslocamento bidirecional de 4 bits usando um único bloco PROCESS. A *entity* expõe as entradas de controle — RESET (síncrono), LOAD (paralelo), DIR (direção do deslocamento) e SHIFT (habilita deslocamento) — além do sinal de clock CLK e do vetor de dados paralelos de 4 bits D_IN(3 downto 0). A saída é o vetor Q_OUT(3 downto 0). Na *architecture*, declara-se um sinal interno REG(3 downto 0) := (others => '0') que mantém o estado corrente do registrador e é ligado diretamente a Q_OUT.

Dentro do PROCESS(RESET, CLK) modela-se primeiramente o reset síncrono: se RESET = '1' então REG <= (others => '0'). Caso contrário, ao detectar rising_edge(CLK), o bloco executa uma escolha ordenada de controles:

1. **LOAD = '1'** → carrega D_IN em REG;
2. **SHIFT = '1'** → desloca REG um bit, para esquerda ou direita conforme DIR (‘1’=left, ‘0’=right), preenchendo o bit extremo com ‘0’;
3. **caso contrário** → REG mantém seu valor (hold).

Esse encadeamento de prioridades (RESET > LOAD > SHIFT > HOLD) garante que, mesmo com múltiplos controles ativos, o registrador se comporte de forma determinística. Ao simular no ModelSim, instancia-se o DUT num testbench que gera um clock periódico e aplica sequencialmente: um pulso de reset, um ciclo de load com um vetor de dados de teste, vários ciclos de deslocamento para cada direção e, por fim, um período sem habilitar shift, para verificar o hold. Na janela de formas de onda, verifica-se que, após o load, Q_OUT assume corretamente o valor de D_IN; então, em cada borda de subida subsequente, o padrão de bits se desloca à esquerda ou à direita conforme esperado, confirmando o funcionamento completo do registrador bidirecional.

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 REG_BIDIRECIONAL_4B IS
 PORT(
  CLOCK   : IN  STD_LOGIC;
  RESET    : IN  STD_LOGIC;
  LOAD  : IN  STD_LOGIC;
  DADOS    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
  DIRECAO   : IN  STD_LOGIC;
  DADO_ESQ  : IN  STD_LOGIC;
  DADO_DIR  : IN  STD_LOGIC;
  SAIDAS   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END REG_BIDIRECIONAL_4B;
ARCHITECTURE REG_BIDIRECIONAL_4B_ARCH OF
REG_BIDIRECIONAL_4B IS
 SIGNAL REGISTRO : STD_LOGIC_VECTOR(3 DOWNTO
0);
```

```
BEGIN
 SAIDAS <= REGISTRO;
 PROCESS(CLOCK)
 BEGIN
  IF RISING_EDGE(CLOCK) THEN
   IF RESET = '1' THEN
    REGISTRO <= (OTHERS => '0');
   ELSIF LOAD = '1' THEN
    REGISTRO <= DADOS;
   ELSIF DIRECAO = '0' THEN
    REGISTRO <=  REGISTRO(2 DOWNTO 0) &
DADO_ESQ ;
   ELSIF DIRECAO = '1' THEN
    REGISTRO <= DADO_DIR & REGISTRO(3
DOWNTO 1) ;
  ELSE
    REGISTRO <= REGISTRO;
   END IF;
  END IF;
 END PROCESS;END
REG_BIDIRECIONAL_4B_ARCH;
```

Listagem 3: Implementação do circuito do registrador de deslocamento bidirecional de 4 bits.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tb_reg_bidirecional_4b IS
END tb_reg_bidirecional_4b;
ARCHITECTURE behavior OF tb_reg_bidirecional_4b IS
 COMPONENT REG_BIDIRECIONAL_4B
  PORT (
   CLOCK   : IN  STD_LOGIC;
   RESET   : IN  STD_LOGIC;
   LOAD    : IN  STD_LOGIC;
   DADOS   : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
   DIRECAO  : IN  STD_LOGIC;
   DADO_ESQ : IN  STD_LOGIC;
   DADO_DIR : IN  STD_LOGIC;
   SAIDAS   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
 END COMPONENT;
 SIGNAL CLOCK_s   : STD_LOGIC := '0';
 SIGNAL RESET_s   : STD_LOGIC := '0';
 SIGNAL LOAD_s    : STD_LOGIC := '0';
 SIGNAL DADOS_s   : STD_LOGIC_VECTOR(3 DOWNTO 0)
:= (others => '0');
 SIGNAL DIRECAO_s  : STD_LOGIC := '0';
 SIGNAL DADO_ESQ_s : STD_LOGIC := '0';
 SIGNAL DADO_DIR_s : STD_LOGIC := '0';
 SIGNAL SAIDAS_s   : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
 DUT: REG_BIDIRECIONAL_4B
  PORT MAP (
CLOCK   => CLOCK_s,
   RESET   => RESET_s,
   LOAD    => LOAD_s,
   DADOS   => DADOS_s,
   DIRECAO  => DIRECAO_s,
   DADO_ESQ => DADO_ESQ_s,
   DADO_DIR => DADO_DIR_s,
   SAIDAS   => SAIDAS_s  );
clk_gen: PROCESS
 BEGIN
  LOOP
   CLOCK_s <= '0'; WAIT FOR 5 ns;
   CLOCK_s <= '1'; WAIT FOR 5 ns;
  END LOOP;
 END PROCESS clk_gen;
 stim_proc: PROCESS
 BEGIN
  RESET_s <= '1';
  WAIT UNTIL rising_edge(CLOCK_s);
  RESET_s <= '0';
  WAIT UNTIL rising_edge(CLOCK_s);
WAIT FOR 1 ns;    DADOS_s <= "1010";
  LOAD_s  <= '1';
  WAIT UNTIL rising_edge(CLOCK_s);
  LOAD_s  <= '0';
  WAIT UNTIL rising_edge(CLOCK_s);
  DIRECAO_s  <= '0';
  DADO_ESQ_s <= '1';
  WAIT UNTIL rising_edge(CLOCK_s);
  WAIT UNTIL rising_edge(CLOCK_s);
  DIRECAO_s  <= '1';
  DADO_DIR_s <= '0';
  WAIT UNTIL rising_edge(CLOCK_s);
  WAIT UNTIL rising_edge(CLOCK_s);
  DIRECAO_s  <= '0';
  DADO_ESQ_s <= '0';
  DADO_DIR_s <= '0';
  WAIT UNTIL rising_edge(CLOCK_s);
  WAIT UNTIL rising_edge(CLOCK_s);
  WAIT;
 END PROCESS stim_proc;
END behavior;
```

Listagem 4: Implementação do circuito do Testbench do registrador de deslocamento bidirecional de 4 bits.

O código do **REG_BIDIRECIONAL_4B** se divide em duas partes claras Entity e Architecture. A entity expõe uma entrada de relógio CLOCK, sinais de controle RESET (limpa os dados), LOAD (carga), DIRECAO (seleção de direção do deslocamento), bits de entrada DADO_ESQ e DADO_DIR e o vetor de dados DADOS, além da saída SAIDAS. Na architecture, o processo é sensível à borda de subida de CLOCK e trata, em ordem de prioridade, o reset síncrono (zera todo o vetor), a carga, o deslocamento à esquerda e à direita, e, por fim, o hold (mantém o valor atual). Isso concentra toda a lógica sequencial num único bloco.

O **testbench tb_reg_bidirecional_4b** automatiza a verificação do DUT. Ele declara internamente o componente REG_BIDIRECIONAL_4B, cria sinais "_s" para cada porta e instancia o DUT com um PORT MAP. O processo clk_gen gera um clock de 10 ns de período em loop contínuo. No stim_proc, alinhado a WAIT UNTIL rising_edge(CLOCK_s), o algoritmo é:

1. Um pulso de reset (RESET_s='1') para zerar o registrador.
2. Um ciclo de load com DADOS_s="1010" e LOAD_s='1'.
3. Dois ciclos de shift-left com DIRECAO_s='0' e DADO_ESQ_s='1'.
4. Dois ciclos de shift-right com DIRECAO_s='1' e DADO_DIR_s='0'.
5. Por fim, sem habilitar carga nem shift, verifica-se o hold.

2.3 Simulação

Para validar o funcionamento do registrador bidirecional de 4 bits, simulamos o testbench tb_reg_bidirecional_4b no ModelSim, gerando um clock de 10 ns de período (5 ns em '0', 5 ns em '1'). Inicialmente, aplicamos um pulso de reset síncrono (RESET_s = '1') logo numa borda de subida, garantindo que o vetor interno fosse zerado para "0000". Em seguida, estabilizamos o clock por um ciclo extra e, no meio do período seguinte após 1ns, ativamos o sinal de carga paralela (LOAD_s = '1') junto com DADOS_s = "1010". No próximo rising edge, observou-se a saída mudar para "1010", confirmando a correta operação de load, e em seguida desativamos LOAD_s para voltar ao modo de deslocamento.

Para testar o shift, mantivemos o clock em execução e, antes de cada borda, definimos os sinais de direção e os bits de entrada de extremidade. Com DIRECAO_s = '0' e DADO_DIR_s = '0', por exemplo, é adicionado um bit '0' à direita, e alternando para DIRECAO_s = '1' com DADO_ESQ_s = '1', por exemplo, é

adicionado um bit ‘1’ á esquerda. Por fim, todos os controles foram desativados e, em dois ciclos adicionais, constatou-se que SAIDAS_s permaneceu inalterado (hold), validando o comportamento de manutenção de estado. Essa sequência sistemática de reset, load, shift-left, shift-right e hold, alinhada às bordas de subida do clock e visualizada na waveform, garante cobertura completa dos modos de operação do registrador.

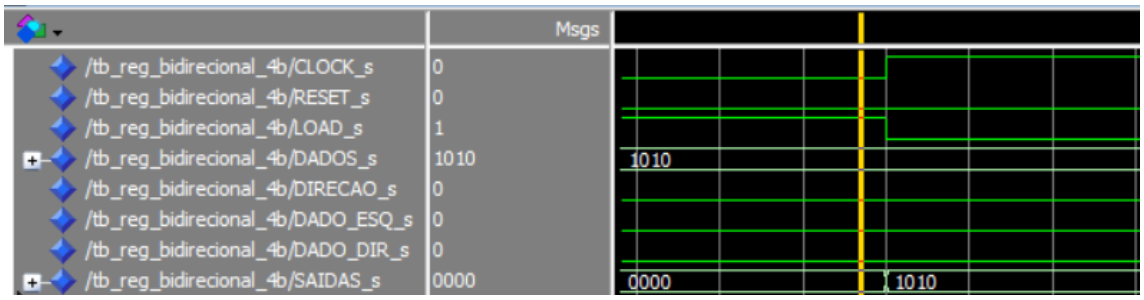

Figura 5: Na simulação do testbench do **REG_BIDIRECIONAL_4B,** após 1ns do primeiro ciclo de clock é possível observar a operação **LOAD,** na qual, quando a variável LOAD_s é igual a ‘1’ a função se ativa e a entrada de DADOS_s é passada para a SAIDA_s, que muda de ‘0000’ para ‘1010’, a entrada inserida em DADOS.

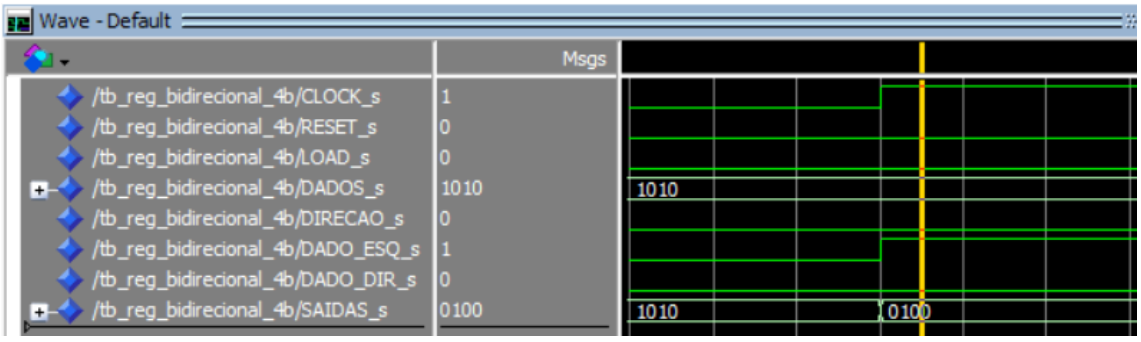

Figura 6: Na simulação do testbench do **REG_BIDIRECIONAL_4B,** é possível observar a operação **SHIFT DIREITA,** na qual, na subida do clock, quando DIRECAO_s assume valor ‘0’, que foi atribuído á direita, o bit ‘0’ de DADO_DIR assume a posição menos significativa da SAIDA_S e a SAIDA_s anterior é deslocada para a esquerda abrindo espaço á direita para o novo bit (de ‘1010’ para ‘0100’).

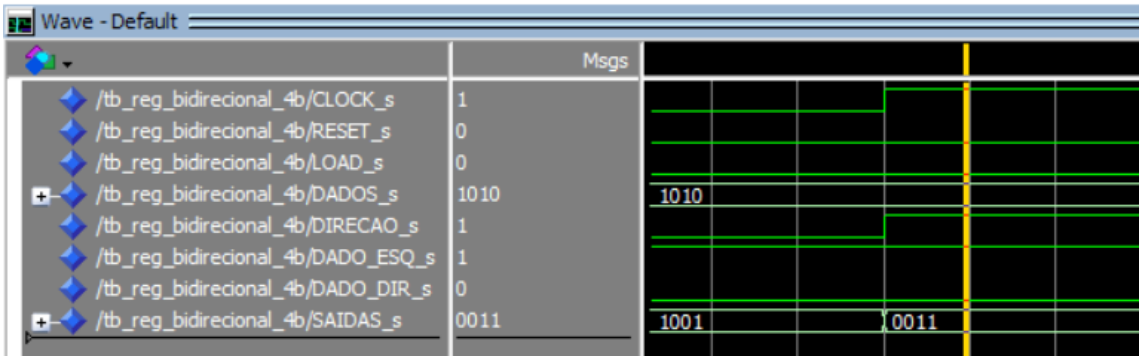

Figura 7: Na simulação do testbench do **REG_BIDIRECIONAL_4B,** é possível observar a operação **SHIFT ESQUERDA,** na qual, na subida do clock, quando DIRECAO_s assume valor '1', que foi atribuído à esquerda, o bit '1' de DADO_ESQ assume a posição mmais significativa da SAIDA_S e a SAIDA_s anterior é deslocada para a direita abrindo espaço á direita para o novo bit (de '1001' para '0011').

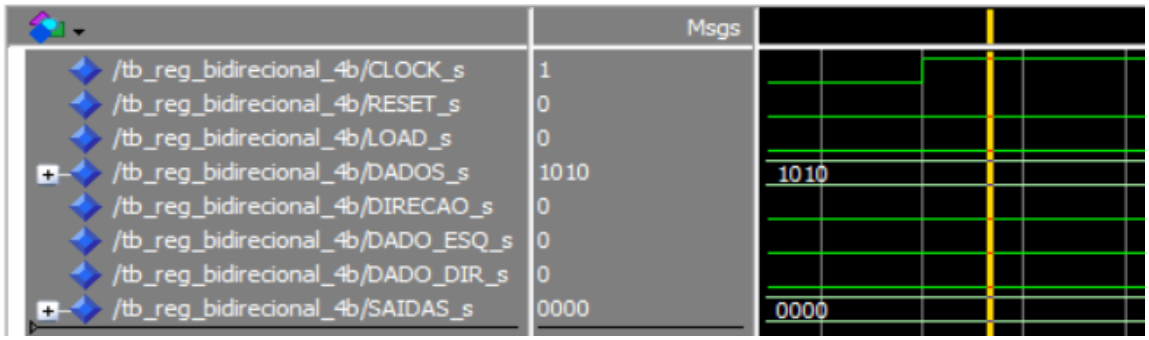

Figura 8: Na simulação do testbench do **REG_BIDIRECIONAL_4B,** após todos os controles serem desativados e, com dois ciclos adicionais, constatou-se que SAIDAS_s permaneceu inalterado, representando o **HOLD** validando o comportamento de manutenção de estado.