Universidade de Brasília
Departamento de Engenharia Elétrica

Aluna: Mariana Soares Oliveira
Matrícula: 231013663
Turma 01
06/01/2024

Relatório Experimento 6

1. Introdução

O intuito do seguinte experimento é, usando a arquitetura *process*, desenvolver um Flip Flop JK gatilhado pela borda de subida e um registrador de deslocamento bidirecional de 4 bits, utilizando a linguagem de descrição de hardware VHDL, e simular o seu comportamento por meio de *testbenchs* realizados no *software* ModelSim.

2. Teoria

2.1 Flip Flop tipo JK

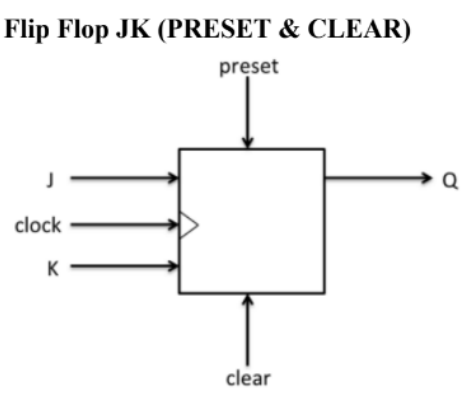

Figura 1. Flip Flop JK

O Flip-Flop JK, apresentado na **Figura 1**, é um circuito digital utilizado para armazenar e manipular informações de forma sequencial. Além de operar na borda de subida do clock (CLK) e depender das entradas J e K para determinar seu comportamento, ele conta com as entradas PR (preset) e CLR (clear), que têm papel crucial no controle direto de sua saída. O preset (PR) permite forçar a saída Q para o estado lógico 1, independentemente do clock ou das entradas J e K. O clear (CLR), por sua vez, força a saída Q para o estado lógico **0**, ignorando também o

De acordo com a Tabela Verdade, como mostrada na Tabela 1:

- Quando *PR=1*, a saída *Q* é forçada a *1*.
- Quando *CLR=1*, *Q* é forçada a *0*.
- Com *PR=0* e *CLR=0*, o estado da saída depende de *J* e *K*:
    - *J=0, K=0*: mantém o estado atual.

- *J=1, K=0J: Q=1.*
- *J=0, K=1: Q=0*
- *J=1, K=1*: a saída inverte seu valor atual.

entradas					saída
PR	*CLR*	*CLK*	*J*	*K*	*Q*
1	x	x	x	x	1
0	1	x	x	x	0
0	0	↓	0	0	mantém
0	0	↓	0	1	0
0	0	↓	1	0	1
0	0	↓	1	1	inverte
0	0	outros	x	x	mantém

Tabela 1. Tabela Verdade do Flip Flop tipo JK

2.2 Registrador de Deslocamento Bidirecional de 4 Bits

O registrador de deslocamento bidirecional é um circuito capaz de armazenar 4 bits e realizar operações de carregamento ou deslocamento dos dados armazenados, conforme o controle das entradas.

Conforme a Tabela Verdade, como mostrada na Tabela 2:

- *RST=1*: o registrador é zerado *(Q=0000)*.
- *LOAD=1*: carrega diretamente os valores *D3,D2,D1,D0* no registrador.
- Quando *RST=0* e *LOAD=0*, o registrador realiza deslocamentos:
  - *DIR=0*: desloca os bits para a esquerda.
  - *DIR=1*: desloca os bits para a direita.
- Caso nenhuma dessas condições seja atendida, o registrador mantém o valor armazenado.

entradas							saída
CLK	*RST*	*LOAD*	*D*	*DIR*	*L*	*R*	*Q*
↓	1	x	xxxx	x	x	x	0000
↓	0	1	$D_3D_2D_1D_0$	x	x	x	$D_3D_2D_1D_0$
↓	0	0	xxxx	0	0	x	$Q_2Q_1Q_0$ 0
↓	0	0	xxxx	0	1	x	$Q_2Q_1Q_0$ 1
↓	0	0	xxxx	1	x	0	0 $Q_3Q_2Q_1$
↓	0	0	xxxx	1	x	1	1 $Q_3Q_2Q_1$
outros	x	x	xxxx	x	x	x	$Q_3Q_2Q_1Q_0$

Tabela 2. Tabela Verdade do Registrador

3. Códigos

Neste experimento utilizamos a linguagem de descrição de hardware VHDL por meio do software Modelsim para desenvolver o Flip Flop tipo JK e o registrador de deslocamento bidirecional conforme as figuras 2 e 3. Posteriormente, foi desenvolvido um código auxiliar chamado *testbench* para cada circuito, descrito nas figuras 4 e 5.

C:/Users/maria/Desktop/231013663_Projeto6/questao1/flipflopJK.vhd - Default

```
Ln#
 1  -- biblioteca
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.all;
 4  -- entidade
 5  entity flipflopJK is
 6      port (
 7          PR, CLR, CLK, J, K : in std_logic;
 8          Q : out std_logic
 9      );
10  end flipflopJK;
11  -- arquitetura
12  architecture arch_flipflopJK of flipflopJK is
13      signal Q_int : std_logic := '0';
14      signal JK_comb : std_logic_vector(1 downto 0);
15  begin
16      -- combina J e K em um vetor de dois bits
17      JK_comb <= J & K;
18      process(PR, CLR, CLK)
19      begin
20          if PR = '1' then
21              Q_int <= '1'; -- set
22          elsif CLR = '1' then
23              Q_int <= '0'; -- reset
24          elsif rising_edge(CLK) then
25              -- comportamento com PR = 0 e CLR = 0
26              case JK_comb is
27                  when "00" =>
28                      Q_int <= Q_int; -- mantem o estado atual
29                  when "01" =>
30                      Q_int <= '0'; -- Q = 0
31                  when "10" =>
32                      Q_int <= '1'; -- Q = 1
33                  when "11" =>
34                      Q_int <= not Q_int; -- inverte o estado atual
35                  when others =>
36                      Q_int <= Q_int; -- mantem o estado atual
37              end case;
38          end if;
39      end process;
40      Q <= Q_int;
41  end arch_flipflopJK;
```

Figura 2. Codificação do flip flop tipo JK (questão 1)

C:/Users/maria/Desktop/231013663_Projeto6/questao2/registrador.vhd - Default

```
Ln#
 1  -- biblioteca
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  -- entidade
 5  entity registrador is
 6      port (
 7          CLK, RST, LOAD, DIR, L, R : in STD_LOGIC;
 8          D : in STD_LOGIC_VECTOR (3 downto 0);
 9          Q : out STD_LOGIC_VECTOR (3 downto 0)
10      );
11  end registrador;
12  -- arquitetura
13  architecture rtl of registrador is
14      signal reg: STD_LOGIC_VECTOR (3 downto 0) := "0000";
15  begin
16      process (CLK)
17      begin
18          if rising_edge(CLK) then
19              if RST = '1' then
20                  reg <= "0000";  -- resetar o registrador
21              elsif LOAD = '1' then
22                  reg <= D;  -- carregar valor de entrada
23              elsif DIR = '0' and L = '1' then
24                  reg <= reg(2) & reg(1) & reg(0) & L;  -- deslocar para a esquerda
25              elsif DIR = '1' and R = '1' then
26                  reg <= R & reg(3) & reg(2) & reg(1);  -- deslocar para a direita
27              else
28                  reg <= reg;  -- manter valor atual
29              end if;
30          end if;
31      end process;
32      Q <= reg;
33  end rtl;
```

Figura 3. Codificação do registrador de deslocamento (questão 2)

C:/Users/maria/Desktop/231013663_Projeto6/questao1/tb_flipflopJK.vhd - Default

```
Ln#
 1  -- biblioteca
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  use IEEE.NUMERIC_STD.ALL;
 5  -- entidade
 6  entity tb_flipflopJK is end;
 7  -- arquitetura
 8  architecture tb_flipflopJK_arch of tb_flipflopJK is
 9      -- declaração de flipflopJK
10      component flipflopJK is
11          port (
12              PR, CLR, CLK, J, K : in std_logic;
13              Q : out std_logic
14          );
15      end component;
16      -- sinais auxiliares
17      signal s  : std_logic_vector(1 downto 0) := "00";
18      signal clk : std_logic := '0';
19      signal jk : std_logic_vector(1 downto 0) := "00";
20  begin
21      -- instância do flipflopJK
22      ul: flipflopJK
23          port map (
24              pr => s(1),
25              clr => s(0),
26              J => jk(1),
27              K => jk(0),
28              clk => clk,
29              Q => open
30          );
31      -- geração do clock
32      clk <= not clk after 5 ns;
33      -- processo de estímulo
34      estimulo: process
35      begin
36          for i in 0 to 4 loop
37              s <= std_logic_vector(to_unsigned(i, 2)); -- altera PR e CLR
38              jk <= "00"; -- mantém o estado
39              wait for 10 ns;
40              jk <= "01"; -- reseta o estado
41              wait for 10 ns;
42              jk <= "11"; -- inverte o estado
43              wait for 10 ns;
44              jk <= "10"; -- define o estado
45              wait for 10 ns;
46          end loop;
47          wait;
48      end process;
49  end tb_flipflopJK_arch;
50
```

Figura 4. *Testbench* da questão 1

C:/Users/maria/Desktop/231013663_Projeto6/questao2/tb_registrador.vhd - Default

```
Ln#
 1  -- biblioteca
 2  library IEEE;
 3  use IEEE.STD_LOGIC_1164.ALL;
 4  -- entidade do Testbench
 5  entity tb_registrador is end;
 6  -- arquitetura
 7  architecture sim of tb_registrador is
 8      -- Declaração do componente registrador
 9      component registrador
10          port (
11              CLK, RST, LOAD, DIR, L, R : in std_logic;
12              D : in std_logic_vector(3 downto 0);
13              Q : out std_logic_vector(3 downto 0)
14          );
15      end component;
16      signal clk : std_logic := '0';
17      signal rst, load, dir, l, r : std_logic := '0';
18      signal d : std_logic_vector(3 downto 0) := "0000";
19      signal q : std_logic_vector(3 downto 0);
20      constant clk_period : time := 10 ns;
21  begin
22      uut: registrador port map (CLK => clk, RST => rst, LOAD => load, DIR => dir, L => l, R => r, D => d, Q => q);
23  -- geração do clock
24      clk_process : process
25      begin
26          while true loop
27              clk <= '0';
28              wait for clk_period / 2;
29              clk <= '1';
30              wait for clk_period / 2;
31          end loop;
32      end process;
33      -- Testes
34      est_proc: process
35      begin
36          -- reset ativo
37          report "Teste 1: Reset ativo (RST = '1')";
38          rst <= '1';
39          wait for clk_period;
40          rst <= '0';
41          -- load ativo
42          load <= '1';
43          d <= "1010";
44          wait for clk_period;
45          load <= '0';
46          -- deslocar para a esquerda
47          dir <= '0';
48          l <= '1';
49          wait for clk_period;
50          -- deslocar para a direita
51          dir <= '1';
52          r <= '1';
53          wait for clk_period;
54          -- sem alteração nos sinais
55          dir <= '0';
56          l <= '0';
57          r <= '0';
58          wait for clk_period;
59          wait;
60      end process;
61  end sim;
```

Figura 5. *Testbench* da questão 2

4. Compilação

Os códigos gerados anteriormente foram submetidos a uma compilação com o intuito de garantir seu funcionamento, como mostrado nas figuras 6 e 7 ambos os códigos não apresentam erros de sintaxe.

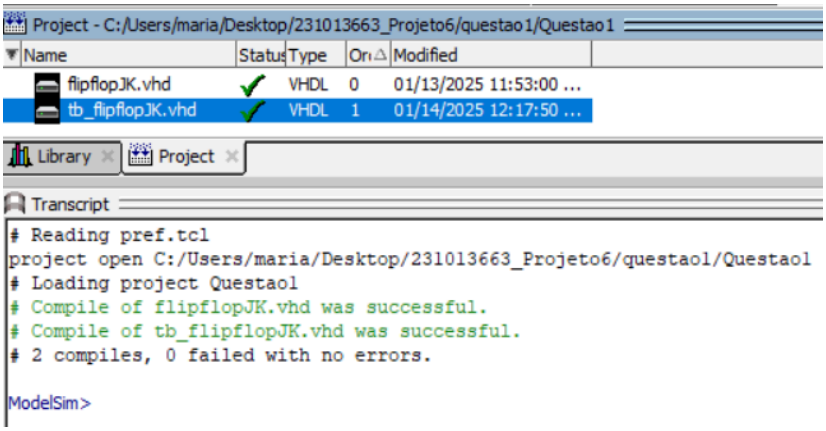

Figura 6. Compilação da questão 1

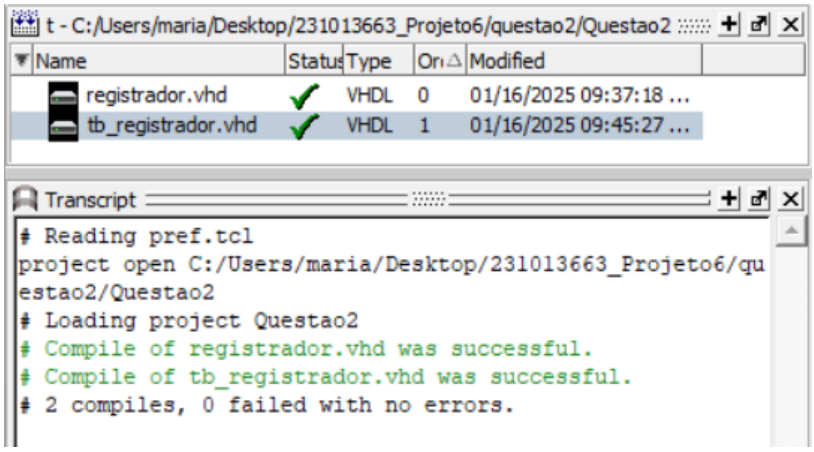

Figura 7. Compilação da questão 2

5. Simulação

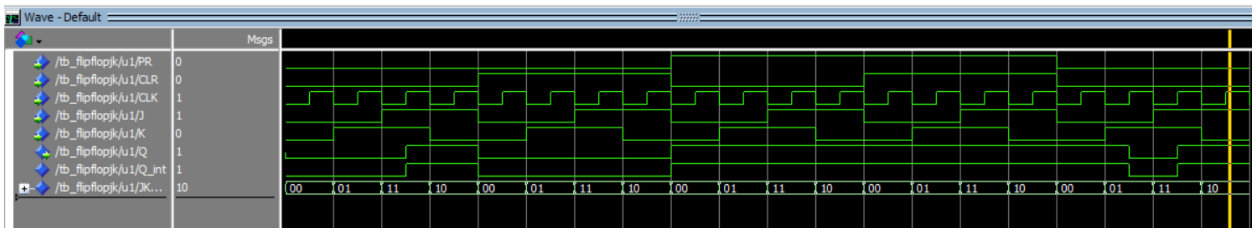

Figura 8. Simulação de onda do banco de testes da questão 1

- **Cursor 1 (em 3.913 ns): PR = 0 , CLR = 0 , CLK= 0 , J= 0 , K= 0, Q= 0**
- **Cursor 2 (em 28.043 ns): PR = 0 , CLR = 0 , CLK= 1 , J= 1 , K= 1, Q= 1**
- **Cursor 3 (em 38.478 ns): PR = 0 , CLR = 0 , CLK= 1 , J= 1 , K= 0, Q= 1**
- **Cursor 4 (em 77.609 ns): PR = 0 , CLR = 1 , CLK= 1 , J= 1 , K= 0, Q= 0**

- **Cursor 5 (em 96.522 ns): PR = 1 , CLR = 0 , CLK= 1 , J= 0 , K= 1, Q= 1**
- **Cursor 6 (em 137.391 ns): PR = 1 , CLR = 1 , CLK= 1 , J= 0 , K= 1, Q= 1**
- **Cursor 7 (em 168.696 ns): PR = 0 , CLR = 0 , CLK= 1 , J= 0 , K= 0, Q= 1**
- **Cursor 8 (em 183.261 ns): PR = 0 , CLR = 0 , CLK= 0 , J= 1 , K= 1, Q= 0**

    - Q_int e JK (últimas duas linhas), são apenas sinais auxiliares usados na codificação do flipflop e seus valores estão expressos em J, K e Q, respectivamente.

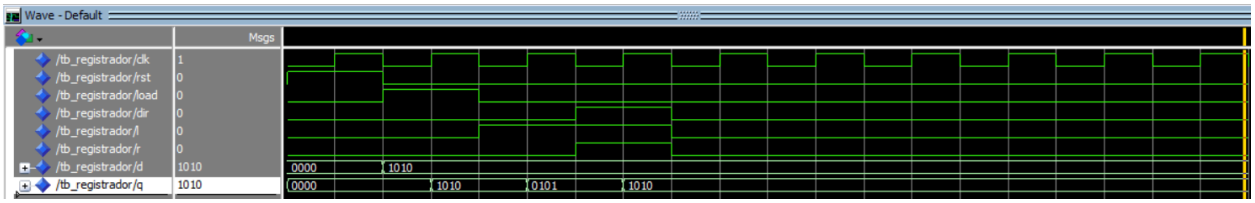

Figura 9. Simulação de onda do banco de testes da questão 2

- **Cursor 1 (3.066 ns): clk = 0 , rst = 1 , load = 0 , dir = 0 , l = 0 , r = 0 , d = 0000 , q = 0000;**
- **Cursor 2 (7.555 ns): clk = 1 , rst = 1 , load = 0 , dir = 0 , l = 0 , r = 0 , d = 0000 , q = 0000;**
- **Cursor 3 (12.591 ns): clk = 0 , rst = 0 , load = 1 , dir = 0 , l = 0 , r = 0 , d = 1010, q = 0000;**
- **Cursor 4 (17.628 ns): clk = 1 , rst = 0 , load = 1 , dir = 0 , l = 0 , r = 0 , d = 1010 , q = 1010;**
- **Cursor 5 (23.321 ns): clk = 0 , rst = 0 , load = 0 , dir = 0 , l = 1 , r = 0 , d = 1010 , q = 1010;**
- **Cursor 6 (28.686 ns): clk = 1 , rst = 0 , load = 0 , dir = 0 , l = 1 , r = 0 , d = 1010 , q = 0101;**
- **Cursor 7 (33.504 ns): clk = 0 , rst = 0 , load = 0 , dir = 1 , l = 1 , r = 1 , d = 1010 , q = 0101;**
- **Cursor 8 (37.336 ns): clk = 1 , rst = 0 , load = 0 , dir = 1 , l = 1 , r = 1 , d = 1010 , q = 1010;**
- **Cursor 9 (42.482 ns): clk = 0 , rst = 0 , load = 0 , dir = 0 , l = 0 , r = 0 , d = 0000 , q = 0000.**

6. Análise

Neste experimento, foram analisadas duas estruturas: o Flip Flop tipo JK e o registrador de deslocamento bidirecional. Com base nas Tabelas Verdade 1 e 2, nas suas respectivas explicações na seção de **Teoria** e na simulação de de onda apresentada nas Figuras 8 e 9, é possível compreender o funcionamento esperado de ambos os circuitos. Dessa forma, pode-se

afirmar que os códigos desenvolvidos correspondem ao comportamento esperado, uma vez que os valores obtidos coincidem com aqueles descritos nas tabelas verdade

7. Conclusão

No experimento, foi possível descrever o comportamento das estruturas propostas e entender suas características. As simulações geraram os dados esperados, que foram comparados com as tabelas verdade dos circuitos (Tabelas 1 e 2). Não houveram erros ou divergências observados durante a realização do experimento.