**Universidade de Brasília**
Departamento da Engenharia Elétrica
Laboratório de Sistemas Digitais

# Relatório 06

Circuitos Sequenciais

José Antônio Alcântara da Silva de Andrade Mat: 232013031

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

**Brasília, DF**

**18 de janeiro de 2025**

# 1 Objetivos

1. Utilizar a estrutura `process` do VHDL.
2. Implementar e simular circuitos sequenciais.

# 2 Atividades

Essa sessão do laboratório tinha como objetivo a construção de um flip flop do tipo JK e um circuito para realizar shift de bits. Ambos devem utilizar do sistema de `process` presente no VHDL.

## 2.1 Exercício 1

### 2.1.1 Modelagem

O modelo do flip flop JK a ser construído possui cinco entradas e uma saída. Isto é: duas entradas de controle, uma entrada de clock, duas entradas de dados, e a saída armazenando o dado.

As entradas de controle forçam um estado na saída. Quando PR (preset) for 1, a saída $Q$ deve também ser 1, independente de qualquer outro estado. Já a saída CLR (clear), ela esvazia a saída quando ativada, ou seja, quando CLR é 1, a saída é 0. A entrada de clock é usada apenas para transformar o circuito numa máquina síncrona, a qual faz com que apenas haja alterações de estado na subida de sinal (quando o clock sobe de 0 para 1).

As entradas J e K servem como as entradas padrões de um flip flop desse modelo. A tabela verdade resultante está representada na Tabela 1.

| **Entradas** | | | | | **Saída** |
|---|---|---|---|---|---|
| PR | CLR | CLK | J | K | $Q_{i+1}$ |
| 1 | x | x | x | x | 1 |
| 0 | 1 | x | x | x | 0 |
| 0 | 0 | ↑ | 0 | 0 | $Q_i$ |
| 0 | 0 | ↑ | 0 | 1 | 0 |
| 0 | 0 | ↑ | 1 | 0 | 1 |
| 0 | 0 | ↑ | 1 | 1 | $\bar{Q}_i$ |
| Outros | | | | | $Q_i$ |

**Tabela 1:** Tabela verdade do JK

### 2.1.2 Implementação

A implementação segue processo similar aos realizados anteriormente nos laboratórios: primeiramente delcara-se todas as portas de entrada e saída, e então desenvolve-se a arquitetura do circuito.

Para as portas do flip flop JK, usaremos cinco entradas e uma saída, todas de apenas um bit, como anteriormente discutido. A declaração dessas segue na Listagem 1.

**Listagem 1:** Requisitos para o flip flop JK.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity JK is
5     port (
```

```
6           J, K, clk, CLR, PR: in std_logic;
7           Q: out std_logic
8       );
9   end JK;
```

As linhas 1 e 2 da Listagem 1 importam as bibliotecas esseciais para o uso do VHDL. Em seguida, da linha 4 até a linha 9 o flip flop é declarado com suas portas, a linha 6 sendo as entradas e a linha 7 a saída.

**Listagem 2:** Arquitetura do flip flop JK.

```
1   architecture JK_ARCH of JK is
2       signal JK : std_logic_vector(1 downto 0);
3   begin
4       JK <= J & K;
5       process(clk, CLR, PR)
6           variable Qbuf : std_logic;
7       begin
8           if PR = '1' then
9               Qbuf := '1';
10          elsif CLR = '1' then
11              Qbuf := '0';
12          elsif rising_edge(clk) then
13              case JK is
14                  when "00" => Qbuf := Qbuf;
15                  when "01" => Qbuf := '0';
16                  when "10" => Qbuf := '1';
17                  when "11" => Qbuf := not(Qbuf);
18                  when others => Qbuf := Qbuf;
19              end case;
20          end if;
21          Q <= Qbuf;
22      end process;
23  end JK_ARCH;
```

Na Listagem 2, implementa-se a arquitetura do flip flop. Primeiramente, cria-se um sinal $JK$ na linha 2, vetor de dois bits, o qual é composto simplesmente da concatenação das entradas $J$ e $K$ (na linha 4). Ele será usado mais tarde para descreer os casos possíveis de entrada de dados no flip flop.

Similarmente define-se uma variável na linha 6, `Qbuf`, que será usada para processar o valor da saída antes de atribuí-lo a ela (na linha 21). Por conveniência, tanto a saída $Q$ como a variável `Qbuf` serão tratadas como o mesmo termo.

Dentro do escopo de `process`, aqui definido para acionar na mudança do clock, CLR ou PR (linha 5), implementa-se a Tabela 1. Nas linhas 8 e 9 garante-se que quando PR é 1, $Q$ também seja 1. Em cascateamento, se a primeira condição falhar (ou seja, PR é 0), então confere-se se CLR é 1. Se sim, então a saída é definida como 0.

Para o processamento de dados, se tanto PR e CLR forem 0, e o clock estiver em uma subida (linha 12), confere-se o valor de JK, aqui nas linhas 13 a 19. As quatro possíveis combinações de entradas são definidas das linhas 14 a 17, com a linha 18 agindo como uma rede de proteção caso ocorra alguma falha.

Assim, o flip flop está completamente implementado. Seu código final está na Listagem 3.

**Listagem 3:** Código final do flipflop JK.

```
1       architecture JK_ARCH of JK is
2           signal JK : std_logic_vector(1 downto 0);
3       begin
4           JK <= J & K;
5           process(clk, CLR, PR)
6               variable Qbuf : std_logic;
7           begin
8               if PR = '1' then
9                   Qbuf := '1';
10              elsif CLR = '1' then
11                  Qbuf := '0';
```

```
12              elsif rising_edge(clk) then
13                  case JK is
14                      when "00" => Qbuf := Qbuf;
15                      when "01" => Qbuf := '0';
16                      when "10" => Qbuf := '1';
17                      when "11" => Qbuf := not(Qbuf);
18                      when others => Qbuf := Qbuf;
19                  end case;
20              end if;
21              Q <= Qbuf;
22          end process;
23      end JK_ARCH;
```

### 2.1.3 Simulação

Para a simulação, usa-se de um testbench e um top module para controlar os testes. O código do testbench utilizado com o flip flop JK segue na Listagem 7, enquanto o top module está descrito na Listagem 8.

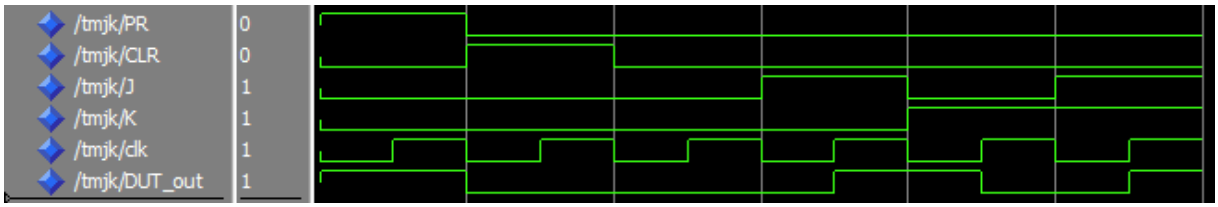
O circuito é colocado em uma série de testes sequenciais. Inicialmente, no teste 1, verifica-se se a entrada PR funciona, e no teste 2 se a entrada CLR funciona. Para o teste 3 checa-se se a combinação 00 para JK é válida, no teste 4 a combinação 01, e assim por diante, até o teste 6.

Verifica-se, pelas Figuras 1 e 2 que não ocorreu nenhum erro durante a simulação, visto a ausência de notificações no console. A saída `DUT_out` representa a saída $Q$ do circuito.

**Figura 1:** Console após a simulação do flip flop JK.

```
VSIM 2> run -all
# ** Note: Iniciando teste...
#    Time: 0 ps  Iteration: 0  Instance: /tmjk/INT2
# ** Note: Teste finalizado.
#    Time: 3 us  Iteration: 0  Instance: /tmjk/INT2
```

**Figura 2:** Figura de onda gerada na simulação do flip flop JK.




## 2.2 Exercício 2

### 2.2.1 Modelagem

O shifter é um circuito capaz de adicionar 0 ou 1, na esquerda ou direita, de um vetor de bits pré-carregado na memória. Este circuito será projetado para um vetor de 4 bits. De acordo com as especificações requisitadas, serão necessárias três entradas de controle, três entradas de comandos, uma entrada de clock e uma saída. Essa máquina será completamente síncrona, ou seja, todas suas operações serão realizadas apenas na subida do sinal do clock (CLK).

As entradas de controle servem dois diferentes propósitos. Quando RST (reset) for 1, remove-se toda a informação guardada, ou seja, a saída $Q$ é 0000. Já as entradas LOAD e $D$ (data) realizam o carregamento de um vetor no circuito: quando LOAD é 1, a saída $Q$ é $D$.

Finalmente, o funcionamento do shifter é dirigido por três variáveis. Quando DIR (direction) é 0, usa-se o valor de $L$ (left) para realizar um shift para esquerda. Já quando DIR é 1, usa-se o valor de $R$ (right) para realizar um shift para a direita. A tabela verdade resultante do sistema está representada na Tabela 2.

**Tabela 2:** Tabela verdade do shifter

| **Entradas** | | | | | | | **Saída** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| CLK | RST | LOAD | D | DIR | L | R | Q | | | |
| ↑ | 1 | x | x | x | x | x | 0 | 0 | 0 | 0 |
| ↑ | 0 | 1 | $D_3D_2D_1D_0$ | x | x | x | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| ↑ | 0 | 0 | x | 0 | 0 | x | $Q_2$ | $Q_1$ | $Q_0$ | 0 |
| ↑ | 0 | 0 | x | 0 | 1 | x | $Q_2$ | $Q_1$ | $Q_0$ | 1 |
| ↑ | 0 | 0 | x | 1 | x | 0 | 0 | $Q_3$ | $Q_2$ | $Q_1$ |
| ↑ | 0 | 0 | x | 1 | x | 1 | 1 | $Q_3$ | $Q_2$ | $Q_1$ |
| Outros | | | | | | | Mantém | | | |

### 2.2.2 Implementação

Para a implementação do shifter serão necessárias sete entradas e uma saída, como discutido anteriormente. Seis dessas entradas serão valores de um bit, enquanto uma entrada será um vetor de 4 bits. A saída será um vetor de 4 bits, também. O circuito é declarado na Listagem 4.

**Listagem 4:** Requisitos para o shifter.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity SHIFTER is
5      port (
6          clk, rst, load, DIR, L, R: in std_logic;
7          D: in std_logic_vector(3 downto 0);
8          Q: out std_logic_vector(3 downto 0)
9      );
10 end SHIFTER;
```

As linhas 1 e 2 na Listagem 4 implementam os requisitos para a simulação de circuitos em VHDL. Da linha 4 até a linha 10 declara-se o shifter, com as entradas de um bit declaradas na linha 6, a entrada vetor de 4 bits na linha 7 e a saída na linha 8. Em seguida, constrói-se a arquitetura do circuito, aqui na Listagem 5.

**Listagem 5:** Arquitetura do shifter.

```
1  architecture SHIFTER_ARCH of SHIFTER is
2  begin
3      process(clk)
4          variable Qbuf : std_logic_vector(3 downto 0);
5      begin
6          if rising_edge(clk) then
7              if rst = '1' then
8                  Qbuf := "0000";
9              elsif load = '1' then
10                 Qbuf := D;
11             elsif DIR = '0' then
12                 Qbuf := Qbuf(2 downto 0) & L;
13             else
14                 Qbuf := R & Qbuf(3 downto 1);
15             end if;
16         end if;
17         Q <= Qbuf;
18     end process;
19 end SHIFTER_ARCH;
```

Novamente, prova-se necessário o uso de uma variável para processar o novo valor da saída antes de atribuí-lo. Declarada na linha 8, `Qbuf` será usada para esse propósito. A saída $Q$ recebe o valor da variável no final das operações, na linha 17. Por conveniência, $Q$ e `Qbuf` serão tratadas como o mesmo termo.

Para esse método de processamento, será necessário apenas checar a variável clock (na linha 3) e se tal se encontra numa subida (na linha 6). Começa-se, então, a implementação da Tabela 2.

Se a entrada reset for 1 (linha 7), então define-se a saída como o vetor 0000 (linha 8). Senão, confere-se se a entrada load é 1 (linha 9) e, se for, coloca-se o vetor carregado em $D$ como a saída. Caso ambas estejam desligadas, confere-se o valor de DIR (linha 11). Se for 0, concatena-se o valor de $L$ à *direita* da saída, a fim de empurrar o vetor à esquerda. Caso contrário, concatena-se $R$ à esquerda (seguindo a mesma lógica). Usa-se do operador `&` para concatenar os bits.

Assim, a arquitetura está completamente implementada, com o código final presente na Listagem 6.

**Listagem 6:** Código final do shifter.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity SHIFTER is
5      port (
6          clk, rst, load, DIR, L, R: in std_logic;
7          D: in std_logic_vector(3 downto 0);
8          Q: out std_logic_vector(3 downto 0)
9      );
10 end SHIFTER;
11
12 architecture SHIFTER_ARCH of SHIFTER is
13 begin
14     process(clk)
15         variable Qbuf : std_logic_vector(3 downto 0);
16     begin
17         if rising_edge(clk) then
18             if rst = '1' then
19                 Qbuf := "0000";
20             elsif load = '1' then
21                 Qbuf := D;
22             elsif DIR = '0' then
23                 Qbuf := Qbuf(2 downto 0) & L;
24             else
25                 Qbuf := R & Qbuf(3 downto 1);
26             end if;
27         end if;
28         Q <= Qbuf;
29     end process;
30 end SHIFTER_ARCH;
```

### 2.2.3 Simulação

Para a simulação, usa-se de um testbench e um top module para controlar os testes. O código do testbench utilizado com o flip flop JK segue na Listagem 9, enquanto o top module está descrito na Listagem 10.

O circuito é colocado numa série de testes sequenciais. Primeiramente, coloca-se a entrada reset como 1, e verifica-se se a saída está com o vetor 0000. Para o teste 2, introduz-se o vetor 0110 na saída por meio das entradas load e $D$. Segue-se, então, uma sequência de testes para verificar a funcionalidade das entradas DIR, L e R. Empurra-se um 0 e um 1 à esquerda do vetor para os testes 3 e 4. E, finalmente, um 0 e um 1 à direita do vetor para os testes 5 e 6, finalizando-se os testes.

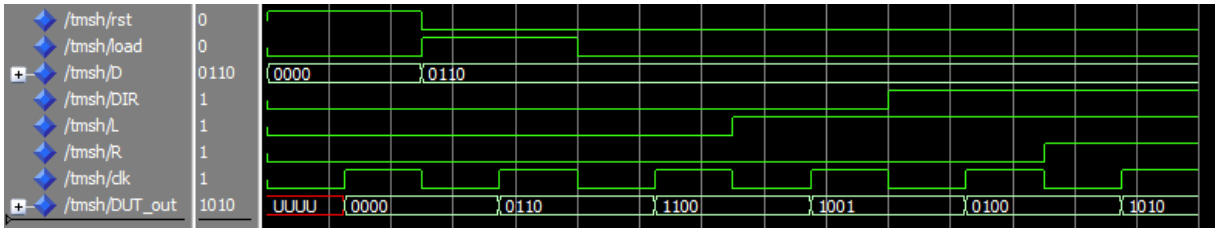
Como visto nas Figuras 3 e 4, não ocorreram erros durante a simulação. O console

não apresenta nenhum aviso além dos de início e término, e o gráfico de onda segue as espectativas do sistema. A saída `DUT_out` representa a saída $Q$ do circuito.

**Figura 3:** Console após a simulação do shifter.

```
VSIM 5> run -all
# ** Note: Iniciando teste...
#    Time: 0 ps  Iteration: 0  Instance: /tmsh/INT2
# ** Note: Teste finalizado.
#    Time: 3 us  Iteration: 0  Instance: /tmsh/INT2
```

**Figura 4:** Figura de onda gerada na simulação do shifter.




## 2.3 Extra

### 2.3.1 Testbenches e Topmodules

**Listagem 7:** Testbench do flipflop JK.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity TBJK is
5      port (
6          DUT_out: in std_logic;
7          J, K, clk, CLR, PR: out std_logic
8      );
9  end TBJK;
10
11 architecture TBJK_ARCH of TBJK is
12 begin
13     process
14     begin
15         report "Iniciando teste..." severity NOTE;
16
17         J <= '0';
18         K <= '0';
19         CLR <= '0';
20         PR <= '1';
21         clk <= '0';
22         wait for 250 ns;
23         clk <= '1';
24         wait for 250 ns;
25         assert (DUT_out = '1') report "Falha no Teste 1." severity ERROR;
26
27         PR <= '0';
28         CLR <= '1';
29         clk <= '0';
30         wait for 250 ns;
31         clk <= '1';
32         wait for 250 ns;
33         assert (DUT_out = '0') report "Falha no Teste 2." severity ERROR;
34
35         CLR <= '0';
36         J <= '0';
37         K <= '0';
```

```
        clk <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = '0') report "Falha no Teste 3." severity ERROR;

        J <= '1';
        K <= '0';
        clk <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = '1') report "Falha no Teste 4." severity ERROR;

        J <= '0';
        K <= '1';
        clk <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = '0') report "Falha no Teste 5." severity ERROR;

        J <= '1';
        K <= '1';
        clk <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = '1') report "Falha no Teste 6." severity ERROR;

        report "Teste finalizado." severity NOTE;
        wait;
    end process;
end TBJK_ARCH;
```

**Listagem 8:** Top Module do flipflop JK.

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

entity TMJK is
end TMJK;

architecture TMJK_ARCH of TMJK is
    component JK is
        port (
            J, K, clk, CLR, PR: in std_logic;
            Q: out std_logic
        );
    end component;

    component TBJK is
        port (
            DUT_out: in std_logic;
            J, K, clk, CLR, PR: out std_logic
        );
    end component;

    signal J, K, clk, CLR, PR : std_logic;
    signal DUT_out : std_logic;

begin
    INT1 : JK port map(J, K, clk, CLR, PR, DUT_out);
    INT2 : TBJK port map(DUT_out, J, K, clk, CLR, PR);
end TMJK_ARCH;
```

**Listagem 9:** Testbench do shifter.

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

entity TBSH is
    port (
```

```
        DUT_out: in std_logic_vector(3 downto 0);
        clk, rst, load, DIR, L, R: out std_logic;
        D: out std_logic_vector(3 downto 0)
    );
end TBSH;

architecture TBSH_ARCH of TBSH is
begin
    process
    begin
        report "Iniciando teste..." severity NOTE;

        clk <= '0';
        rst <= '1';
        load <= '0';
        D <= "0000";
        DIR <= '0';
        L <= '0';
        R <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = "0000") report "Falha no Teste 1." severity ERROR;

        clk <= '0';
        rst <= '0';
        load <= '1';
        D <= "0110";
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = "0110") report "Falha no Teste 2." severity ERROR;

        clk <= '0';
        load <= '0';
        DIR <= '0';
        L <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = "1100") report "Falha no Teste 3." severity ERROR;

        clk <= '0';
        DIR <= '0';
        L <= '1';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = "1001") report "Falha no Teste 4." severity ERROR;

        clk <= '0';
        DIR <= '1';
        R <= '0';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = "0100") report "Falha no Teste 5." severity ERROR;

        clk <= '0';
        DIR <= '1';
        R <= '1';
        wait for 250 ns;
        clk <= '1';
        wait for 250 ns;
        assert (DUT_out = "1010") report "Falha no Teste 6." severity ERROR;

        report "Teste finalizado." severity NOTE;
        wait;
    end process;
end TBSH_ARCH;
```

**Listagem 10:** Top Module do shifter.

```
library IEEE;
```

```vhdl
use IEEE.std_logic_1164.all;

entity TMSH is
end TMSH;

architecture TMSH_ARCH of TMSH is
    component SHIFTER is
        port (
            clk, rst, load, DIR, L, R: in std_logic;
            D: in std_logic_vector(3 downto 0);
            Q: out std_logic_vector(3 downto 0)
        );
    end component;

    component TBSH is
        port (
            DUT_out: in std_logic_vector(3 downto 0);
            clk, rst, load, DIR, L, R: out std_logic;
            D: out std_logic_vector(3 downto 0)
        );
    end component;

    signal clk, rst, load, DIR, L, R : std_logic;
    signal DUT_out, D : std_logic_vector(3 downto 0);

begin
    INT1 : SHIFTER port map(clk, rst, load, DIR, L, R, D, DUT_out);
    INT2 : TBSH port map(DUT_out, clk, rst, load, DIR, L, R, D);
end TMSH_ARCH;
```

### 2.3.2 Compilação

Como segue na Figura 5, nenhum dos códigos obteve erros de compilação.

**Figura 5:** Compilação dos códigos.

```
# Compile of e1.vhd was successful.
# Compile of e2.vhd was successful.
# Compile of tb_jk.vhd was successful.
# Compile of tb_sh.vhd was successful.
# Compile of tm_jk.vhd was successful.
# Compile of tm_sh.vhd was successful.
# 6 compiles, 0 failed with no errors.
```