Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Relatório do Experimento 6

Autor: Henrique Morcelles Salum

Matrícula: 232003008

ENE0040 - Laboratório de Sistemas Digitais - Turma 07

Sumário

1 Introdução **2**
1.1 Sobre o Experimento 2
1.1.1 Flip-Flop JK 2
1.1.2 Registrador de Deslocamento 2
1.2 Introdução Teórica 3
1.2.1 Flip-flops e Circuitos Sequenciais 3
1.2.2 Registradores 3

2 Códigos **5**
2.1 Flip-Flop JK 5
2.2 Registrador de Deslocamento 6
2.2.1 Descrição de *Hardware* 6
2.2.2 *Testbench* 8
2.2.3 *Top-Module* 8

3 Compilação **10**

4 Simulação **10**

5 Análise **11**

6 Conclusão **11**

1 Introdução

1.1 Sobre o Experimento

Este experimento é composto por duas tarefas; em ambas, deve-se escrever códigos em VHDL que implementem sistemas digitais e simulá-los no *software* ModelSim, da Intel.

1.1.1 Flip-Flop JK

Na primeira tarefa, devemos utilizar a estrutura "process" para implementar um flip-flop JK gatilhado pela borda de subida. A tabela-verdade desse sistema está apresentada a seguir:

Entradas					Saída
PR	**CLR**	**CLK**	**J**	**K**	**Q**
1	X	X	X	X	1
0	1	X	X	X	0
0	0	↓	0	0	$LastQ$
0	0	↓	0	1	0
0	0	↓	1	0	1
0	0	↓	1	1	$\overline{LastQ}$
0	0	outros	X	X	$LastQ$

Tabela 1: Tabela-verdade do flip-flop JK

1.1.2 Registrador de Deslocamento

A segunda tarefa consiste em implementar, novamente por meio da estrutura "process", um registrador de deslocamento bidirecional gatilhado pela borda de subida e dotado das entradas **RST** (*reset*) e **LOAD**. Esse sistema é descrito pela tabela-verdade a seguir.

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	$Q_2Q_1Q_00$
↓	0	0	XXXX	0	1	X	$Q_2Q_1Q_01$
↓	0	0	XXXX	1	X	0	$0Q_3Q_2Q_1$
↓	0	0	XXXX	1	X	1	$1Q_3Q_2Q_1$
outros	X	X	XXXX	X	X	X	$Q_3Q_2Q_1Q_0$

Tabela 2: Tabela-verdade do registrador de deslocamento

Para implementar esse registrador, além do código do flip-flop, que será apresentado doravante, foram utilizados multiplexadores 4x1. Isso não seria necessário de acordo com o roteiro, foi uma decisão minha (avalizada pelo professor) implementar o registrador em mais baixo nível (com uma arquitetura estrutural), o que exigiu o uso do multiplexador.

1.2 Introdução Teórica

1.2.1 Flip-flops e Circuitos Sequenciais

Os **flip-flops** são circuitos digitais fundamentais na eletrônica digital, responsáveis por armazenar informações binárias. Diferentemente das portas lógicas combinacionais, cuja saída depende unicamente das entradas atuais, os flip-flops fazem parte dos chamados circuitos sequenciais, nos quais a saída depende não apenas das entradas atuais, mas também do *estado anterior* do sistema. Essa característica os torna essenciais na implementação de memórias, registradores, contadores, máquinas de estados finitos e diversos sistemas digitais temporizados.

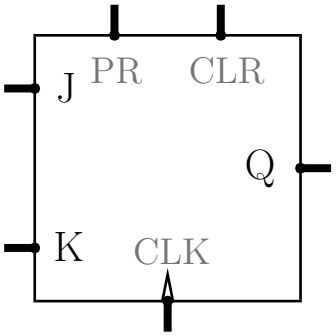

Figura 1: Flip-flop JK

Um flip-flop é capaz de armazenar um único bit de informação e muda de estado geralmente em função de um sinal de controle denominado **clock** (CLK). As transições ocorrem, nas bordas de subida (ou descida) desse sinal de clock, o que sincroniza a operação de múltiplos flip-flops em um sistema.

O comportamento do flip-flop JK, objeto desse experimento, está descrito na Tabela 1. A operação da penúltima linha dessa tabela — de *toggle* ou comutação — torna o flip-flop JK particularmente útil na construção de contadores binários, nos quais o estado se alterna a cada pulso de clock. Além disso, o comportamento bem definido para todos os pares de entradas diferencia o flip-flop JK do flip-flop RS, que apresenta uma condição inválida quando ambas as entradas são 1.

1.2.2 Registradores

Na eletrônica digital, um registrador é um circuito sequencial síncrono composto por um conjunto de flip-flops, geralmente do tipo D ou JK, que armazena um grupo de bits simultaneamente. Cada flip-flop representa uma posição de armazenamento de um bit, e, juntos, formam uma unidade capaz de armazenar palavras binárias de múltiplos bits. Os registradores são componentes essenciais em microprocessadores, microcontroladores e em diversas arquiteturas digitais, pois atuam como memória de curto prazo e mecanismos de transferência de dados.

Um tipo especial de registrador é o registrador de deslocamento (*shift register*). Ele permite que os dados armazenados sejam deslocados, bit a bit, para a esquerda ou para a direita, em cada pulso de clock. Essa funcionalidade é útil para diversas aplicações. Tipicamente, processadores são dotados desses dispositivos; na programação em assembly, eles são diretamente manipulados pelo programador e, em linguagens de alto nível, a sua manipulação é abstraída por meio de variáveis.

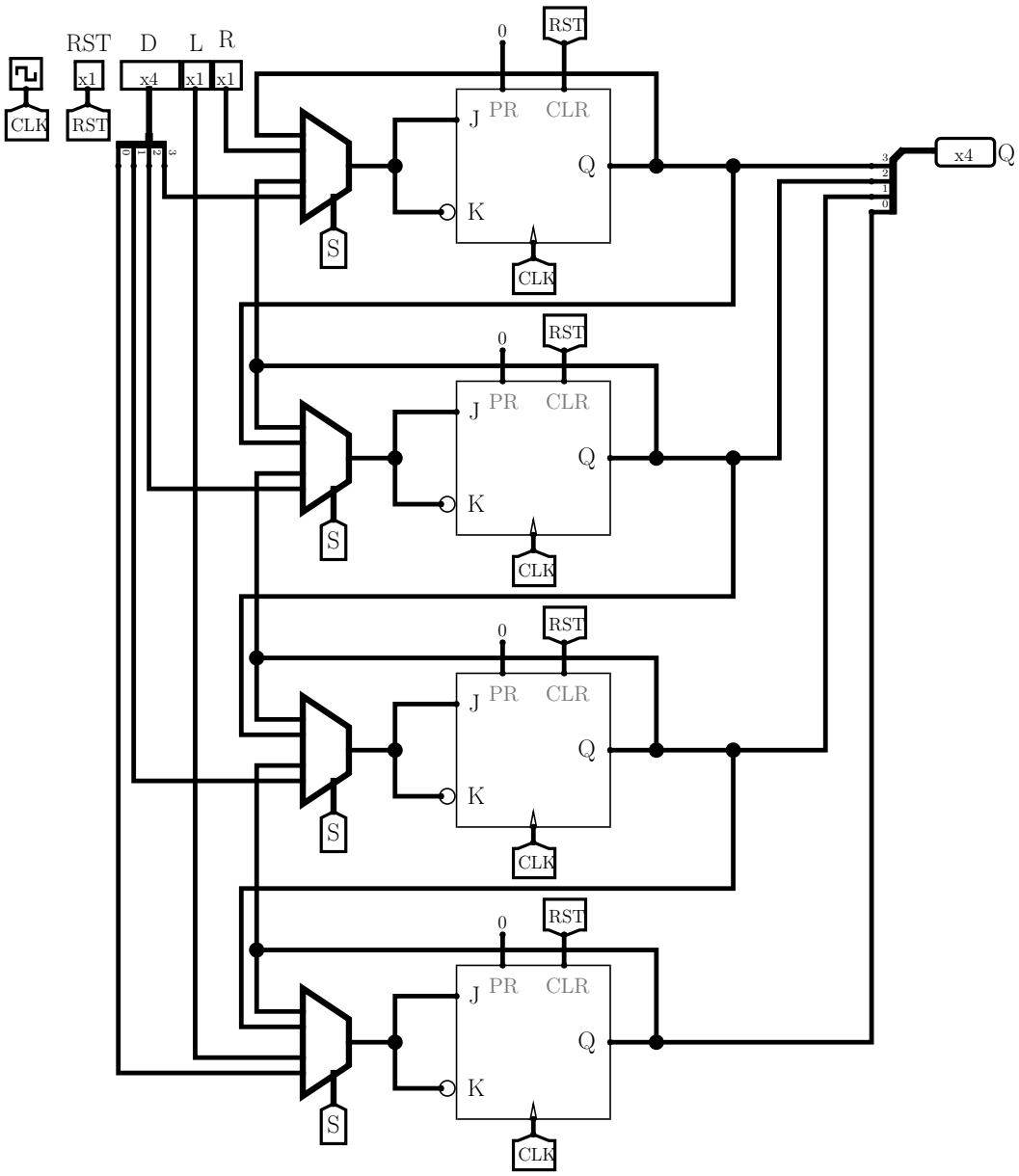

Figura 2: Representação interna do registrador de deslocamento de 4 bits

2 Códigos

2.1 Flip-Flop JK

O código utilizado para implementar o flip-flop seguiu uma arquitetura de alto nível (*behavioral*). O código é bastante simples: as entradas assíncronas foram implementadas por um *if-elsif-else* e as síncronas por uma expressão avaliada na última cláusula desse *if*.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity FlipFlopJK is
5      port (
6          preset:     in std_logic;
7          clear:      in std_logic;
8          clock:      in std_logic;
9          J:          in std_logic;
10         K:          in std_logic;
11         Q:          out std_logic
12     );
13 end entity FlipFlopJK;
14 architecture behavioral of FlipFlopJK is
15     signal Q_in: std_logic;
16 begin
17     process(preset, clear, clock)
18     begin
19         if preset = '1' then
20             Q_in <= '1';
21         elsif clear = '1' then
22             Q_in <= '0';
23         elsif rising_edge(clock) then
24             Q_in <= (not J and not K and Q_in) or (J and not K) or (J and K and not Q_in);
25         end if;
26     end process;
27     Q <= Q_in;
28 end architecture behavioral;
```

Código 1: Descrição de Hardware do flip-flop JK

Não foi necessário *testbench* ou *top-module*. Ao invés disso, o registrador serviu para testar o correto funcionamento do flip-flop.

2.2 Registrador de Deslocamento

O código referente ao registrador, ao contrário do referente ao flip-flop, é de baixo nível (*structural*). Utilizamos multiplexadores para tomar as decisões referentes às entradas de "controle" (que decidem qual será a operação realizada pelo registrador) e instanciamos os flip-flops para armazenar os dados.

2.2.1 Descrição de Hardware

```
entity Registrador4Bits is
    port (
        clock:    in std_logic;
        reset:    in std_logic;
        load:   in std_logic;
        data:      in std_logic_vector(3 downto 0);
        direction:    in std_logic;
        left:      in std_logic;
        right:      in std_logic;
        Q:      out std_logic_vector(3 downto 0)
    );
end entity Registrador4Bits;

architecture structural of Registrador4Bits is
    component FlipFlopJK is
        port (
            preset:     in std_logic;
            clear:      in std_logic;
            clock:      in std_logic;
            J:          in std_logic;
            K:          in std_logic;
            Q:          out std_logic
        );
    end component FlipFlopJK;
    component Mux4x1 is
        port (
            D: in std_logic_vector(3 downto 0);
            S: in std_logic_vector(1 downto 0);
            Y: out std_logic
        );
    end component Mux4x1;
    signal OP: std_logic_vector(1 downto 0) := (others => '0');
    signal FF_output: std_logic_vector(3 downto 0);
    signal mux_input: std_logic_vector(15 downto 0);
    signal mux_output: std_logic_vector(3 downto 0);
    signal J_signals, K_signals: std_logic_vector(3 downto 0);
begin
    OP_ctrl: process(load, direction)
    begin
        if load = '1' then OP <= "11"; -- Load
        elsif direction = '1' then OP <= "01"; -- Shift Right
        elsif direction = '0' then OP <= "10"; -- Shift Left
        else OP <= "00"; -- Hold
        end if;
    end process;
    mux_input <=
        data(3) & FF_output(2) & right & FF_output(3) -- Mux4
```

```
48          & data(2) & FF_output(1) & FF_output(3) & FF_output(2)-- Mux3
49          & data(1) & FF_output(0) & FF_output(2) & FF_output(1)-- Mux2
50          & data(0) & left & FF_output(1) & FF_output(0); -- Mux1
51      Mux1: Mux4x1 port map (D => mux_input(3 downto 0), S => OP, Y => mux_output(0));
52      Mux2: Mux4x1 port map (D => mux_input(7 downto 4), S => OP, Y => mux_output(1));
53      Mux3: Mux4x1 port map (D => mux_input(11 downto 8), S => OP, Y => mux_output(2));
54      Mux4: Mux4x1 port map (D => mux_input(15 downto 12), S => OP, Y => mux_output(3));
55      J_signals <= mux_output;
56      K_signals <= not mux_output;
57
58      FF_JK_1: component FlipFlopJK
59          port map (
60              preset => '0',
61              clear => reset,
62              clock => clock,
63              J => J_signals(0),
64              K => K_signals(0),
65              Q => FF_output(0)
66          );
67
68      FF_JK_2: component FlipFlopJK
69          port map (
70              preset => '0',
71              clear => reset,
72              clock => clock,
73              J => J_signals(1),
74              K => K_signals(1),
75              Q => FF_output(1)
76          );
77
78      FF_JK_3: component FlipFlopJK
79          port map (
80              preset => '0',
81              clear => reset,
82              clock => clock,
83              J => J_signals(2),
84              K => K_signals(2),
85              Q => FF_output(2)
86          );
87
88      FF_JK_4: component FlipFlopJK
89          port map (
90              preset => '0',
91              clear => reset,
92              clock => clock,
93              J => J_signals(3),
94              K => K_signals(3),
95              Q => FF_output(3)
96          );
97
98      Q <= FF_output;
99  end architecture structural;
```

Código 2: Descrição de Hardware do registrador de deslocamento de 4 bits

2.2.2 Testbench

```
entity tb_Registrador4Bits is
    port (
        clock:      in std_logic;
        reset:      out std_logic := '0';
        load:       out std_logic := '1';
        data:       out std_logic_vector(3 downto 0) := "1011";
        direction:  out std_logic := '0';
        left:       out std_logic := '-';
        right:      out std_logic := '-'
    );
end entity tb_Registrador4Bits;

architecture test of tb_Registrador4Bits is
begin
    load_and_reset: process
    begin
        wait for 4 ns;
        reset <= '1';
        wait for 20 ns;
        reset <= '0';
        wait for 25 ns;
        load <= '0';
        left <= '0';
        wait for  30 ns;
        direction <= '1';
        right <= '0';
        wait;
    end process;
end architecture;

```

Código 3: Testbench do registrador

2.2.3 Top-Module

A presença do *process* referente ao *clock* nesse código pode parecer uma mistura das responsabilidades, mas a decisão tomada foi utilizar o *testbench* como um simulacro do resto do circuito, gerando variações nas entradas "normais", e sincronizar esse simulacro com o dispositivo testado diretamente no *top module*. Basicamente, tratamos o *clock* como algo independente do circuito.

```
entity TopModule is
end entity;

architecture structural of TopModule is
    component Registrador4Bits is
        port (
            clock:    in std_logic;
            reset:    in std_logic;
            load:   in std_logic;
            data:      in std_logic_vector(3 downto 0);
            direction:    in std_logic;
            left:      in std_logic;
```

```
            right:      in std_logic;
            Q:      out std_logic_vector(3 downto 0)
        );
    end component;

    component tb_Registrador4Bits is
        port (
            clock:       in std_logic;
            reset:       out std_logic;
            load:        out std_logic;
            data:        out std_logic_vector(3 downto 0);
            direction:   out std_logic;
            left:        out std_logic;
            right:       out std_logic
        );
    end component;

    signal clock_top:     std_logic := '0';
    signal reset_top:     std_logic;
    signal load_top:      std_logic;
    signal data_top:      std_logic_vector(3 downto 0);
    signal direction_top: std_logic;
    signal left_top:      std_logic;
    signal right_top:     std_logic;
    constant clock_period: time := 20 ns;
begin
    DUT: Registrador4Bits
        port map (
            clock => clock_top,
            reset => reset_top,
            load => load_top,
            data => data_top,
            direction => direction_top,
            left => left_top,
            right => right_top,
            Q => open
        );
    TB: tb_Registrador4Bits
        port map (
            clock => clock_top,
            reset => reset_top,
            load => load_top,
            data => data_top,
            direction => direction_top,
            left => left_top,
            right => right_top
        );

    clock_process: process
    begin
        wait for 0.1 ns;
        while true loop
            clock_top <= not clock_top;
            wait for clock_period / 2;
        end loop;
    end process;
end architecture;
```

Código 4: Testbench do registrador

3 Compilação

Após escrever os códigos, é necessário compilá-los pelo ModelSim para que se possa simular os sistemas digitais discutidos. Caso a compilação tenha sucesso, sabemos que não houve erros nos códigos apresentados, mas ainda não podemos afirmar que a lógica para implementar os circuitos está correta; isso será analisado nas próximas seções.

A seguir, está a mensagem de compilação dos códigos apresentados acima. Sem nenhum erro, como pode ser visto no terminal no canto inferior da figura.

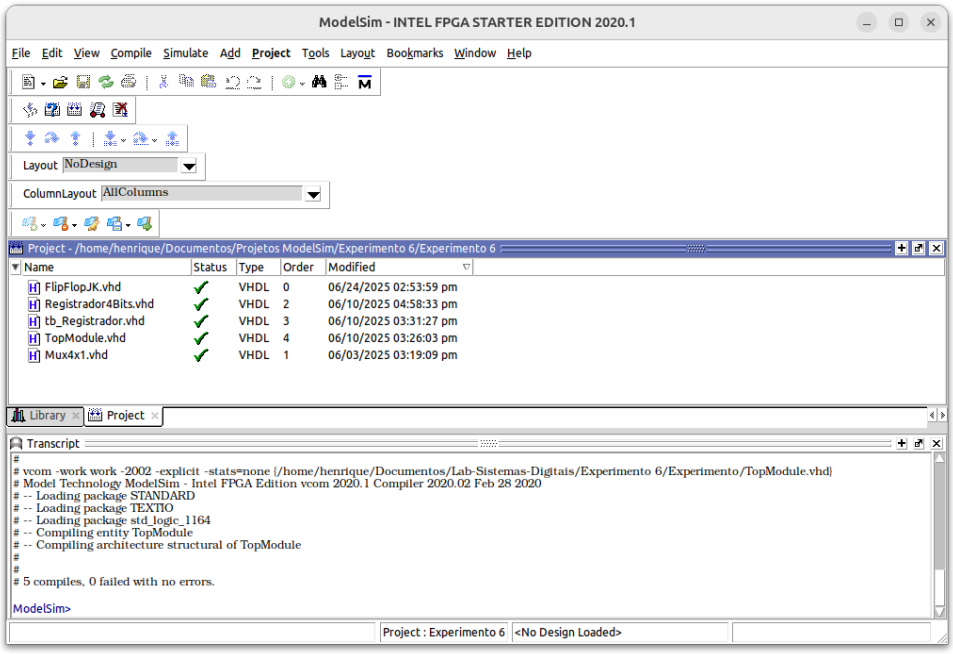

Figura 3: Compilação de todos os códigos apresentados

4 Simulação

O gráfico de forma de onda gerado pelo ModelSim ao simular o *top-module* está exibido abaixo. Foram marcados com cursores os momentos de interesse para a análise.

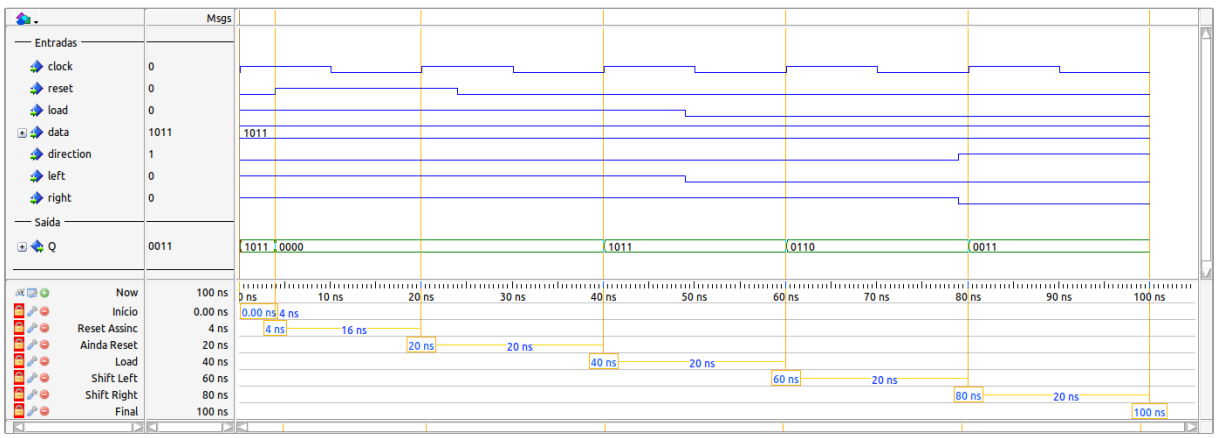

Figura 4: Simulação em forma de onda binária do registrador

5 Análise

Na onda apresentada na Figura 4, estão marcados os momentos cruciais do teste. Esses instantes são:

1. 4 ns - *reset* assíncrono;
2. 20 ns - tentativa de *load* (não ocorre pois o *reset* segue em 1);
3. 40 ns - *load*;
4. 60 ns - *shift left* (entra o 0 na direita);
5. 80 ns - *shift right* (entra o 0 na esquerda).

Analisando a saída Q, percebe-se que todas as instruções funcionam de acordo com o esperado. Podemos concluir, portanto, que a implementação apresentada está correta.

6 Conclusão

Neste experimento, tivemos o primeiro contato com flip-flops e, portanto, com circuitos sequenciais. Esses dispositivos são cruciais no projeto de máquinas de estados, além de serem base para o desenvolvimento de dispositivos mais complexos, como registradores, contadores, microprocessadores, etc.

Além disso, instanciando os flip-flops JK, desenvolvemos um registrador de deslocamento de 4 bits, dispositivo que é usado em máquinas de estados mais complexas e é essencial para a arquitetura de processadores digitais.

Em suma, chegamos a uma etapa essencial do estudo de sistemas digitais: começamos a entender o funcionamento em baixo nível dos dispositivos que se tornaram base para o desenvolvimento do mundo digital. É com base nisso que foram desenvolvidos os processadores digitais e, posteriormente, as linguagens de programação, sistemas operacionais, etc.