Relatório 6 VHDL - Turma 05

Yan Tavares de Oliveira
202014323

Introdução

Este experimento consiste em 3 etapas. Na primeira (questão 1), iremos descrever em VHDL e simular no software ModelSim uma entidade que descreva o comportamento de um flip-flop JK.

Na segunda etapa (questão 2), iremos descrever em VHDL e simular no software ModelSim um registrador de deslocamento bidirecional de 4 bits, sendo descrito pela seguinte tabela verdade

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	$Q_3Q_2Q_1Q_0$

Imagem 1: Tabela verdade do circuito de deslocamento da questão 2

Teoria

Nas questões 1 e 2, devemos implementar um flip-flop JK, descrito pela imagem 1, que tem a função de memória em um circuito lógico.

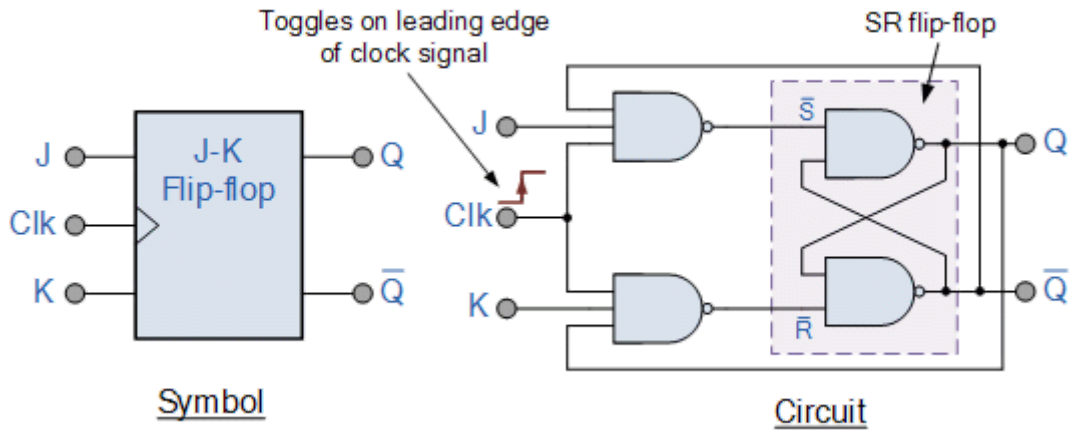

Imagem 2: Estrutura do flip-flop JK

Sua tabela verdade, por sua vez, é a seguinte

J	K	CLK	Q
0	0	↑	Q_0 (não muda)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (comuta)

Imagem 3: Tabela verdade do flip-flop JK

Podemos, portanto, notar que iremos implementar um flip-flop de subida de clock, ou seja, o valor irá ser checado a cada movimento de subida do clock.

Sabendo disso, na questão 2 usaremos estes conceitos para implementar um circuito síncrono e com memória que descreve um registrador de deslocamento bidirecional de 4 bits.

Códigos

Na questão 1, utilizamos as bibliotecas IEEE e IEEE.std_logic_1164.all para descrever o flip-flop JK

Q1.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  entity Q1 is
5     port( J,K: in  std_logic;
6           pr: in std_logic;
7           clk: in std_logic;
8           clr: in std_logic;
9           Q: out std_logic);
10 end Q1;
11 architecture main of Q1 is
12    signal JK: std_logic_vector(1 downto 0);
13    signal Qbuf  :  std_logic := '0';
14 begin
15   JK <= J & K;
16   process(pr, clk, clr)
17   begin
18     if pr = '1' then Qbuf <= '1';
19     elsif clr = '1' then Qbuf <= '0';
20     elsif rising_edge(clk) then
21       case JK is
22         when "00" =>    Qbuf <= Qbuf;
23         when "01" =>     Qbuf <= '0';
24         when "10" =>     Qbuf <= '1';
25         when "11" =>     Qbuf <= not(Qbuf);
26         when others =>     Qbuf <= Qbuf;
27       end case;
28     end if;
29   end process;
30   Q <= Qbuf;
31 end main;
32
```

Imagem 4. Código principal da questão 1

Foi também criado um código auxiliar de testbench para gerarmos os estímulos necessários para abranger todas as combinações possíveis de entradas.

tb_Q1.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_Q1 is
end tb_Q1;

architecture main of tb_Q1 is

component Q1 is
port(j,k,clr,pr,clk:in std_logic;
   q :out std_logic);
end component;

signal clka, clkb, clkc, clkd, clke: std_logic := '0';

begin

    tb_Q1 : Q1 port map( clk => clka, k => clkb, j => clkc, clr => clkd, pr => clke, q => open);
    clka <= not clka after 1 ns;
    clkb <= not clkb after 2 ns;
    clkc <= not clkc after 4 ns;
    clkd <= not clkd after 8 ns;
    clke <= not clke after 16 ns;
end main;

```

Imagem 5. Código auxiliar da questão 1

Na questão 2, utilizamos as mesmas bibliotecas para simular o registrador.

Q2.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity Q2 is
    port( dir, L, R: in  std_logic;
          reset: in std_logic;
          clk: in std_logic;
          load: in std_logic;
       D:  in std_logic_vector(3 downto 0);
       Q: out std_logic_vector(3 downto 0));
end Q2;
architecture main of Q2 is
  signal Qbuf : std_logic_vector(3 downto 0);
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then Qbuf <= "0000";
      elsif load = '1' then Qbuf <= D;
      elsif dir = '0' then Qbuf <= Qbuf(2) & Qbuf(1) & Qbuf(0) & L;
      elsif dir = '1' then Qbuf <= L & Qbuf(3) & Qbuf(2) & Qbuf(1);
      end if;
   else Qbuf <= Qbuf;
   end if;
  end process;
  Q <= Qbuf;
end main;

```

Imagem 6. Código principal da questão 2

Foi também criado um código auxiliar de testbench para gerarmos os estímulos necessários para abranger todas as combinações possíveis de entradas.

tb_Q2.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_Q2 is end;
architecture main of tb_Q2 is
component Q2 is
port( dir, L, R: in  std_logic;
          reset: in std_logic;
          clk: in std_logic;
          load: in std_logic;
      D:  in std_logic_vector(3 downto 0);
      Q: out std_logic_vector(3 downto 0));
end component;
signal clka, clkb, clkc, clkd, clke, clkf: std_logic := '0';
signal clkg : std_logic_vector(3 downto 0) := "0000";
begin
   tb_Q2 : Q2 port map( clk => clka, R => clkb, L => clkc, dir => clkd, D => clkg, load => clke,
    clka <= not clka after 1 ns;
    clkb <= not clkb after 2 ns;
    clkc <= not clkc after 4 ns;
    clkd <= not clkd after 8 ns;
    clke <= not clke after 16 ns;
    clkf <= not clkf after 32 ns;
    clkg <= not clkg after 64 ns;

end main;

```

Imagem 7. Código auxiliar da questão 2

Compilação

Abaixo estão as mensagens de compilação do projeto, com nenhum erro sendo apresentado em nenhum dos casos

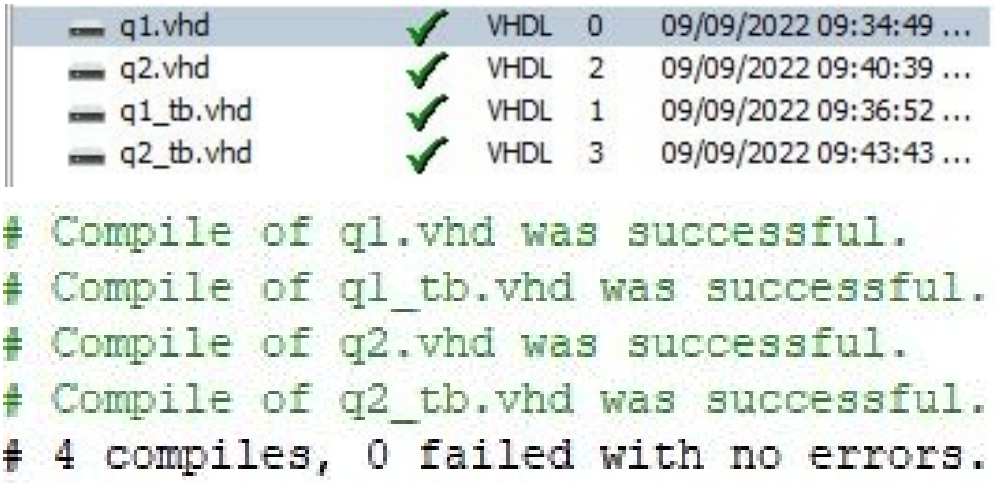

Imagem 8. Mensagem de compilação

Simulação

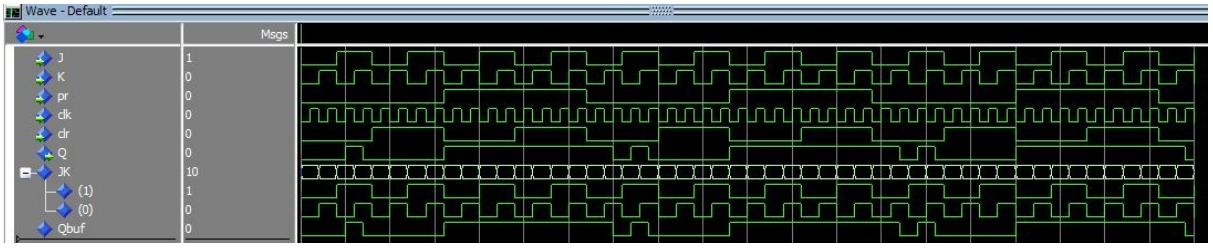

Figura 9. Simulação em forma de onda binária da questão 1.

Figura 10. Simulação em forma de onda binária da questão 2.

Análise

Para determinar a ordem dos bits utilizados na simulação e para abranger todas as combinações possíveis, foram utilizados clocks para alterar o valor bit a bit. Pudemos perceber um resultado condizente com as tabelas verdades em ambos os casos.

Conclusão

Como não houve divergência alguma nos dois casos, podemos então afirmar que implementamos um flip-flop do tipo JK e um registrador bidirecional de 4 bits funcional por meio de uma simulação no software ModelSim.