EXPERIMENTO 6.

-Bernardo Rizzone (232013194)

Introdução: Este relatório tem como objetivos a implementação de circuitos sequenciais (um flip-flop e um registrador de deslocamento bidirecional) ambos em VHDL, e a simulação no ModelSim usando a estrutura “process” da linguagem VHDL.

1) O objetivo do primeiro exercício é a implementação de um flip-flop JK gatilhado pela borda de subida usando a linguagem VHDL. O flip-flop JK é um dispositivo de armazenamento binário que pode ser usado em uma variedade de aplicações de circuitos digitais, incluindo contadores e registradores.

   ***Implementação e simulação**: A entidade flipflopJK é definida com cinco entradas (pr,clr,clk,j,k) e uma saída (Q). As entradas pr e clr são usadas paras as funções preset e clear, respectivamente. A entrada clk é o clock do sistema, e J e K são as entradas de controle do flip-flop. A arquitetura, contem a implementação do comportamento do flip-flop. Um sinal auxiliar JK é usado para concatenar as entradas J e K, e um sinal Q_buffer é utilizado para armazenar o valor intermediário de O. O processo principal é sensível às entras pr, clr e clk. Dentro do processo, as condições para as operações de preset e clear são verificadas primeiro. Se pr for ‘1’, Q_buffer é definido como ‘1’. Se clr for ‘1’, Q_buffer é definido como ‘0’. Se nenhuma dessas condições for verdadeira, o processo verifica a borda de subida do clock (clk). Dependendo do valor do sinal JK, Q_buffer é atualizado de acordo com a tabela verdade do flip-flop.

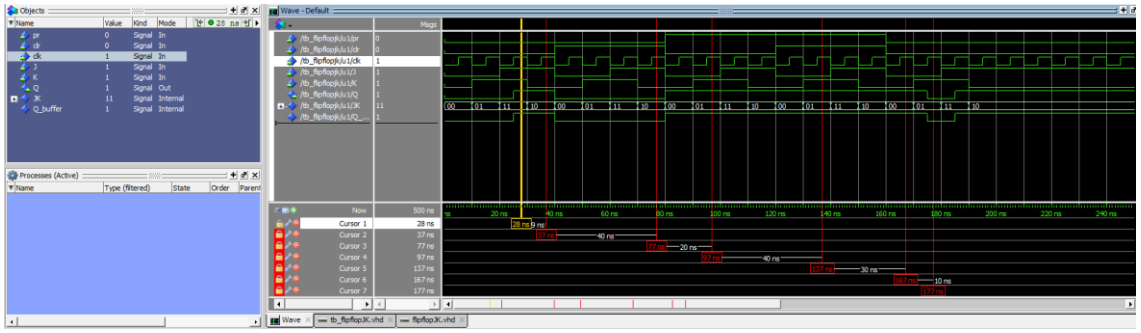
CÓDGIO:

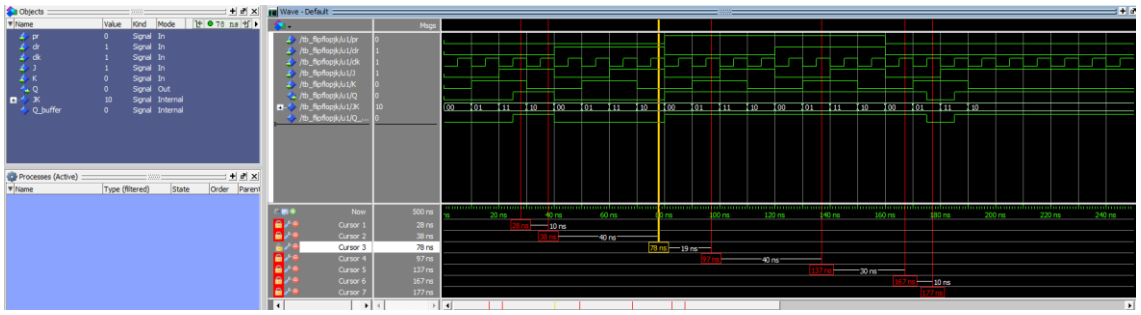
```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;



entity flipflopJK is
    port (
        pr, clr, clk, J, K :in std_logic;
        Q : out std_logic
    );
end flipflopJK;
architecture flipflopJK_arch of flipflopJK is



signal JK :std_logic_vector(1 downto 0);
signal Q_buffer :std_logic := '0';
begin
    JK <= J & K;



    process(pr, clr, clk)
    begin
        if pr = '1' then
            Q_buffer <= '1';
        elsif clr = '1' then
            Q_buffer <= '0';
        elsif rising_edge(clk) then
            if JK = "01" then Q_buffer <= '0'; end if;
            if JK = "10" then Q_buffer <= '1'; end if;
            if JK = "11" then Q_buffer <= not(Q_buffer); end if;
        end if;
    end process;
    Q <= Q_buffer;

end flipflopJK_arch;

```

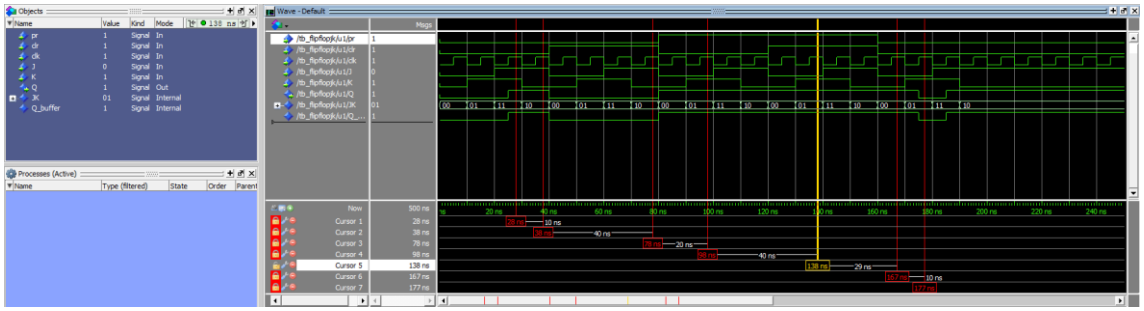
TESTBENCH:

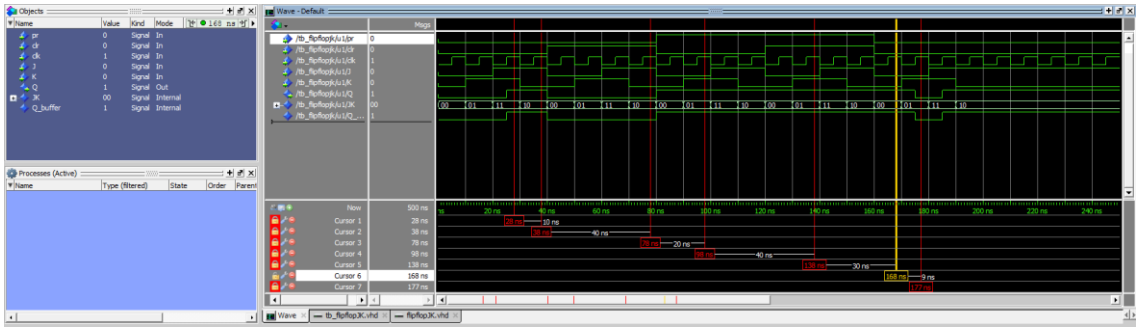
```vhdl
entity tb_flipflopJK is end;

library ieee;
use ieee.std_logic_1164.ALL;
use std.textio.all;
use ieee.numeric_std.all;


architecture tb_flipflopJK_arch of tb_flipflopJK is

 component flipflopJK is
    port (
        pr, clr, clk, J, K :in std_logic;
        Q : out std_logic
    );
 end component;
  signal s  : std_logic_vector(1 downto 0);
  signal clk : std_logic := '0';
  signal jk :std_logic_vector(1 downto 0);

begin
    u1: flipflopJK PORT MAP (pr => s(1), clr => s(0), J => jk(1), K => jk(0), clk => clk, Q => open);


 clk <= not clk after 5 ns;

 estimulo: process
    begin
        for i in 0 to 4 loop
            s <= std_logic_vector(to_unsigned(i, 2));
            jk <= "00";
            wait for 10 ns;
            jk <= "01";
            wait for 10 ns;
            jk <= "11";
            wait for 10 ns;
            jk <= "10";
            wait for 10 ns;
        end loop;
        wait;


 end process;

END;
```

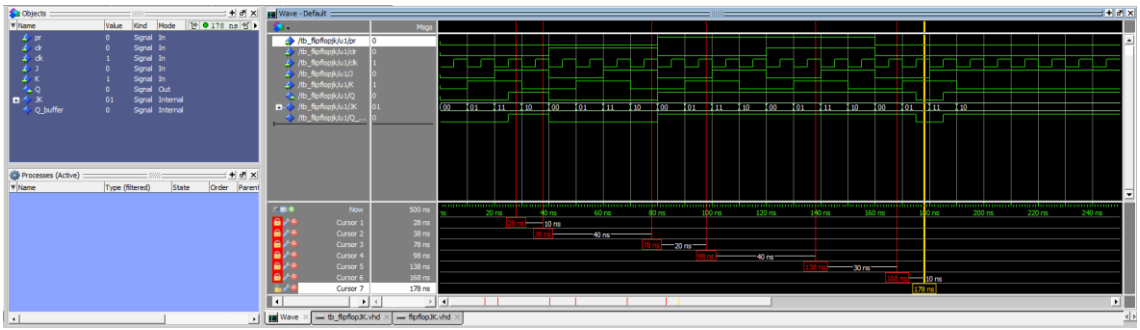
SIMULAÇÃO:











A SIMULAÇÃO: A instancia u1 do componente mapeia os sinais auxiliares aos pinos correspondentes do fliflop. O pino Q é deixado em aberto porque não estamos interessados em sua saída para este teste. O processo de estímulo é responsável por gerar os sinais de teste. O sinal de clock (clk) é alterado a cada 5 ns para simular a borda de subida. Dentro do process, um loop é utilizado para aplicar as diferentes combinações de valores aos sinais S(pr e clr) e jk (entradas J e K) Cada combinação é mantida por 10 ns antes de ser passado para próxima.

Inicializa os sinais s e jk com “00”.
Alterna o sinal de clock a cada 5 ns para simular a borda de subida.
Aplica sequencialmente as combinações de pr e clr através do sinal s.
Testa todas as combinações possíveis das entradas J e K através do sinal jk.
Repete o processo para um número definido de ciclos.

2) O objetivo deste exercício e a implementação de um registrador de deslocamento bidirecional de 4 bits em VHDL, conforme o solicitado. O registrador é capaz de deslocar bits para esquerda ou direita, dependendo do sinal de controle, e pode ser carregado com um novo valor ou resetado para o zero.

***Implementação e simulação**: A entidade reg__bidirecional é definida com entradas para o clock (clk), reset (rst), carregamento (load), direção do deslocamento (dir), e sinais de entrada para o deslocamento (L, R) e dados (D). A saída é um vetor de 4 bits (Q). A arquitetura reg_bidirecional_arch contém a lógica de implementação do registrador. Um sinal auxiliar Q_buffer é utilizado para manter o estado interno do registrador. O processo é sensível ao sinal de clock (clk) e é executado na borda de subida do clock. Dentro do processo, verifica-se primeiro se o reset (rst) está ativo, caso em que o Q_buffer é resetado para “0000”. Se o sinal de carregamento (load) estiver ativo, o Q_buffer é carregado com o valor de D. Caso contrário, o deslocamento é realizado com base no sinal de direção (dir): se dir for ‘0’, os bits são deslocados para a esquerda e o bit menos significativo (Q_buffer(0)) recebe o valor de L; se dir for ‘1’, os bits são deslocados para a direita e o bit mais significativo (Q_buffer(3)) recebe o valor de R.

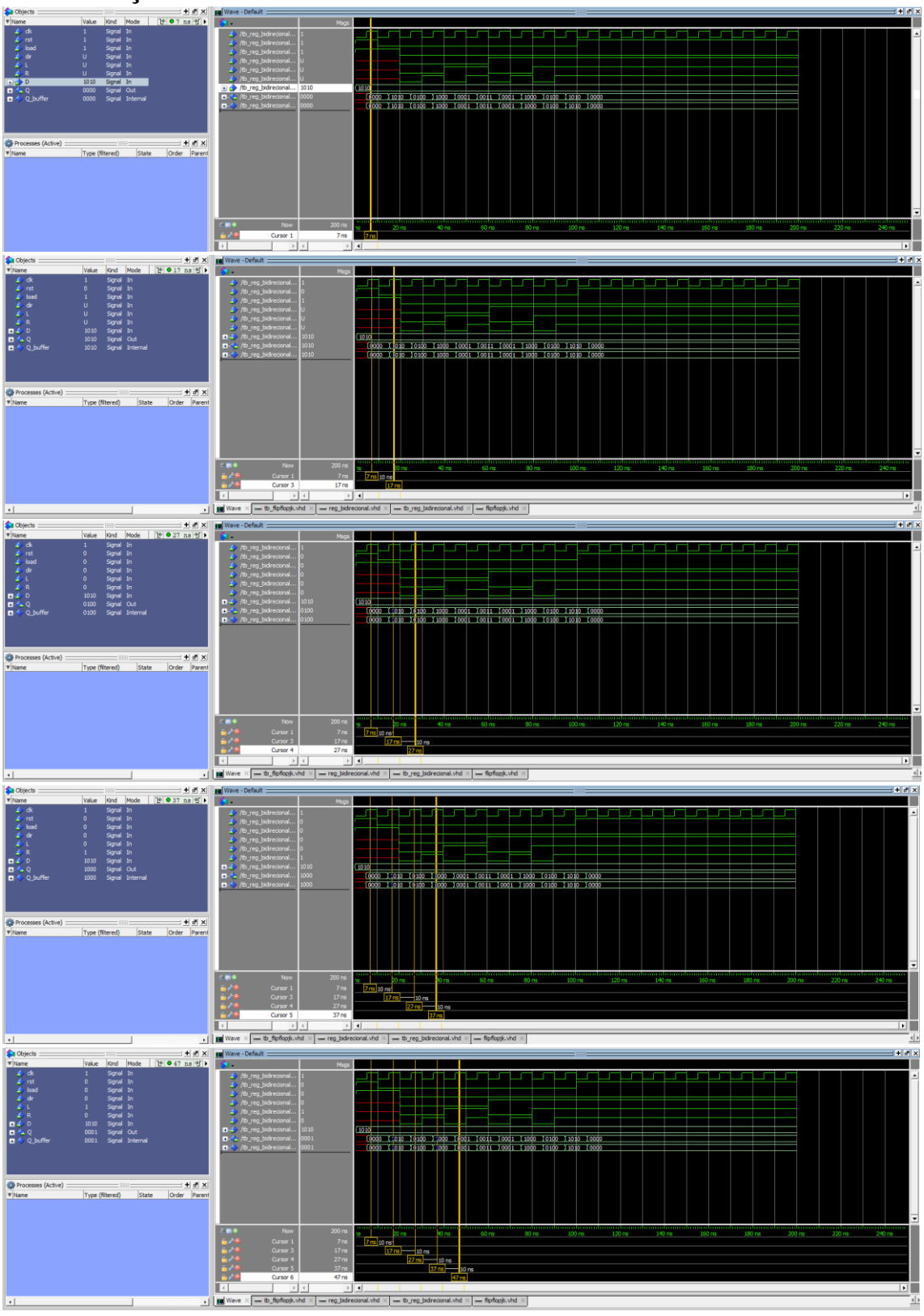
CÓDIGO:

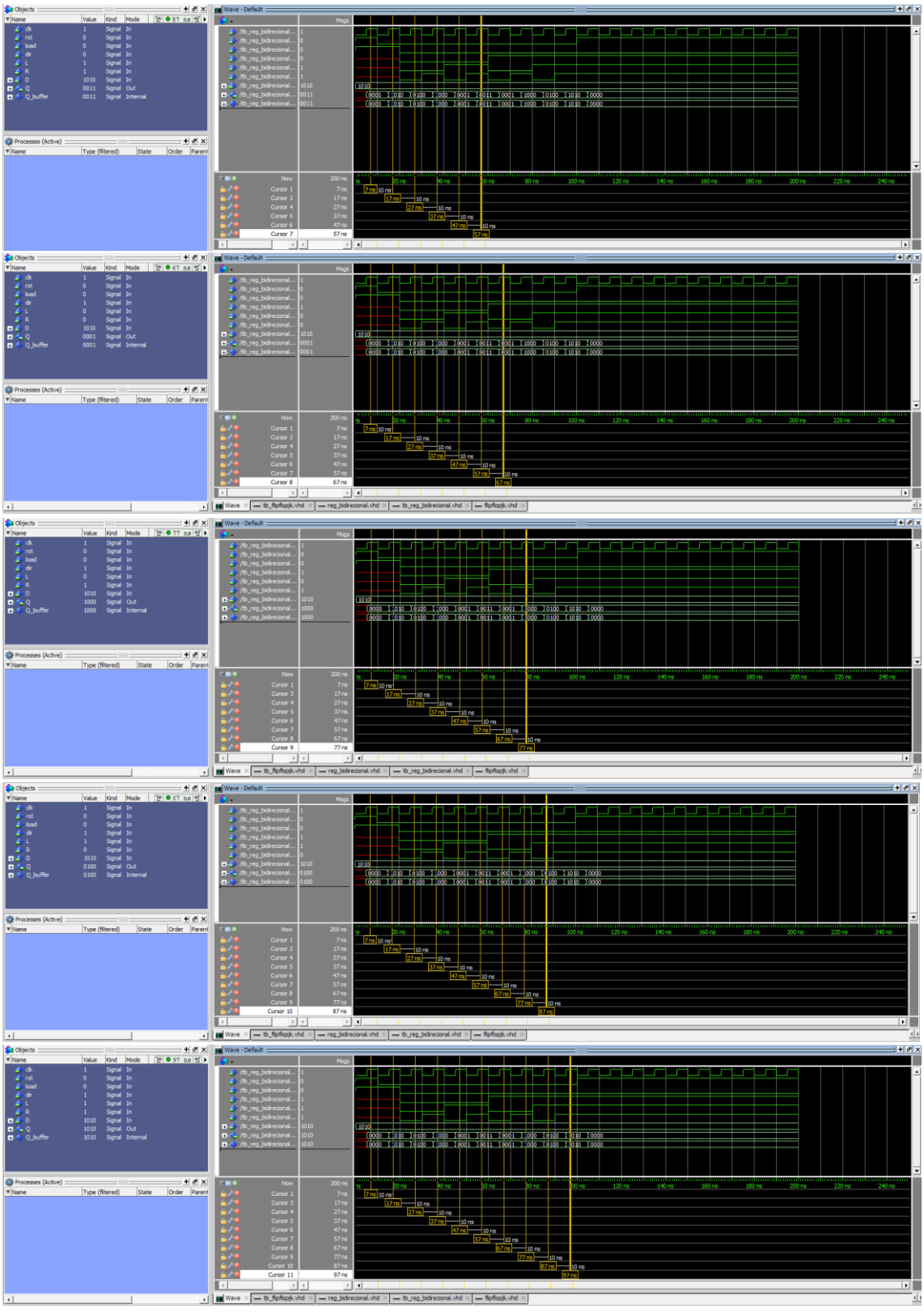
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;



entity reg_bidirecional 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 reg_bidirecional;

architecture reg_bidirecional_arch of reg_bidirecional is


signal Q_buffer :std_logic_vector(3 downto 0);

begin



    process(clk)
    begin
        if (rising_edge(clk)) then
            if (rst = '1') then Q_buffer <= "0000";
            elsif (load = '1') then Q_buffer <= D;
            elsif (dir = '0') then Q_buffer <= std_logic_vector(unsigned(Q_buffer) sll 1); Q_buffer(0) <= L;
            elsif (dir = '1') then Q_buffer <= std_logic_vector(unsigned(Q_buffer) srl 1); Q_buffer(3) <= R;
            end if;
        end if;
    end process;
    Q <= Q_buffer;

end reg_bidirecional_arch;

```

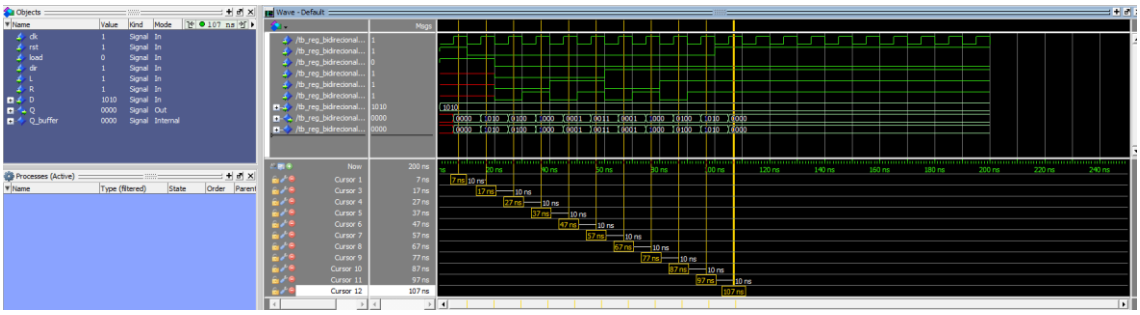
TESTBENCH:

```
entity tb_reg_bidirecional is end;

library ieee;
use ieee.std_logic_1164.ALL;
use std.textio.all;
use ieee.numeric_std.all;



architecture tb_reg_bidirecional_arch of tb_reg_bidirecional is


 component reg_bidirecional 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;

  signal clk : std_logic := '0';
  signal rst, load, dir : std_logic;
  signal LR : std_logic_vector(1 downto 0);
  signal d : std_logic_vector(3 downto 0);



begin
    u0: reg_bidirecional port map (clk, rst, load, dir, LR(1), LR(0), D, Q => open);

 clk <= not clk after 5 ns;

 estimulo: process
    begin
        D <= "1010"; rst <= '1'; load <= '1';
        wait for 10 ns;
        rst <= '0';
        wait for 10 ns;

        dir <= '0'; load <= '0';
        for i in 0 to 3 loop
            LR <= std_logic_vector(to_unsigned(i, 2));
            wait for 10 ns;
        end loop;

        dir <= '1';
        for i in 0 to 3 loop
            LR <= std_logic_vector(to_unsigned(i, 2));
            wait for 10 ns;
        end loop;

        rst <= '1';
        wait;

 end process;

 END;
```

SIMULAÇÃO:







SIMULAÇÃO: A instância u0 do componente reg_bidirecional mapeia os sinais auxiliares aos pinos correspondentes do registrador de deslocamento. O pino Q é deixado aberto (open) porque não estamos interessados em sua saída para este teste. O processo de estímulo é responsável por gerar os sinais de teste. O sinal de clock (clk) é alternado a cada 5 nanossegundos para simular a borda de subida. Dentro do processo estimulo, diferentes combinações de sinais são aplicadas para testar o comportamento do registrador.
Inicialmente, o registrador é carregado com o valor “1010”, o reset (rst) e o load (load) são ativados.
Após 10 ns, o reset é desativado, permitindo que o registrador opere normalmente.
O sinal de direção (dir) é definido para ‘0’ para testar o deslocamento para a esquerda, e o sinal de load é desativado.
Um loop é executado para aplicar diferentes valores ao vetor LR, que representa os sinais L e R.
O sinal de direção (dir) é alterado para ‘1’ para testar o deslocamento para a direita, e o mesmo loop é repetido.
Finalmente, o reset é ativado novamente para reiniciar o processo.

CONCLUSÃO: foram implementados com sucesso um flip-flop JK e um registrador de deslocamento bidirecional de 4 bits. Ambos os componentes são fundamentais para a eletrônica digital e foram validados através de simulações no ModelSim, demonstrando a eficácia do VHDL na modelagem de circuitos lógicos complexos.