

## Laboratório de Sistemas Digitais

### Trabalho Prático nº 4

#### Modelação em VHDL, simulação e implementação de circuitos sequenciais elementares

##### Objetivos

- Modelação em VHDL, simulação, implementação em FPGA e teste de circuitos sequenciais elementares (latches, flip-flops e registos), contadores e divisores de frequência.

##### Sumário

Este trabalho prático está dividido em cinco partes. Na primeira parte é abordada a implementação de *flip-flops* tipo D. Na segunda parte é implementado um registo com a dimensão de um byte. A terceira parte é dedicada à descrição comportamental e implementação de contadores. Na quarta parte é feita a descrição comportamental, a simulação e a implementação de um circuito de divisão de frequência. Finalmente, na última parte, utilizam-se alguns dos componentes modelados anteriormente na implementação de um sistema de contagem simples atualizado à frequência de 1Hz.

##### Parte I

1. Abra a aplicação "Quartus Prime" e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como "FlipFlopD\_Demo".
2. O código VHDL apresentado na Figura 1 implementa um *flip-flop* tipo D. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome "FlipFlopD.vhd".

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity FlipFlopD is
    port(clk : in  std_logic;
          d   : in  std_logic;
          q   : out std_logic);
end FlipFlopD;

architecture Behavioral of FlipFlopD is
begin
    process(clk)
    begin
        if (rising_edge(clk)) then
            q <= d;
        end if;
    end process;
end Behavioral;
```

Figura 1 – Código VHDL de um *flip-flop* tipo D.

3. Efetue a simulação do componente modelado, realizando para tal os seguintes passos:
  - selecione o ficheiro "FlipFlopD.vhd" como o *top-level* do projeto e execute a opção "Analysis & Synthesis".

- crie um ficheiro VWF de suporte à simulação, selecione os portos a usar e especifique os vetores de entrada.
- execute a simulação e analise os resultados.

4. Altere o ficheiro "FlipFlopD.vhd" de modo a modelar um *flip-flop* D com entradas "set" e "reset" síncronas, em que a entrada "reset" tenha a prioridade mais elevada.

5. Efetue a simulação do componente modelado, realizando os passos já apresentados no ponto 3.

6. Visualize o diagrama lógico do circuito resultante da síntese lógica, usando para tal a ferramenta disponível no menu "Tools->Netlist Viewers->RTL Viewer".

7. Crie um novo ficheiro VHDL, chamado "FlipFlopD\_Demo.vhd", onde deverá instanciar o *flip-flop* D modelado no ponto 4 e associar os respetivos portos a pinos concretos da FPGA do kit de desenvolvimento (entradas ligadas a interruptores e saída ligada a um LED do kit DE2-115). Para esse efeito é fornecido o código base da Figura 2 que apresenta, intencionalmente, erros de sintaxe. Edite o ficheiro e corrija esses erros.

```
library IEEE;
use IEEE.STD_LOGIC_1664.all;

entity FlipFlopD_Demo is
    port(SW      : std_logic_vector(2 downto 0);
         KEY     : std_logic_vector(0 downto 0);
         LEDR    : std_logic_vector(0 downto 0);

architecture Shell of FlipFlopD_Demo is
begin
    ff_d : wokr.FlipFlopD(Behavioral)
        port map(clk    => KEY(0 downto 0);
                 d      => SW(0);
                 set    => SW(1);
                 reset  => SW(2);
                 q      => LEDR(0 downto 0) ,
end FlipFlopD_Demo;
```

Figura 2 – Módulo *top-level* em VHDL para ligação dos portos do *flip-flop* a pinos da FPGA do kit DE2-115.

**Nota: O ponto 8 deve ser feito apenas na Aula Nº5/6. Passe para a parte II.**

8. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do componente (não se esqueça de selecionar o ficheiro "FlipFlopD\_Demo.vhd" como o novo *top-level* do projeto e de importar as definições de pinos da FPGA do kit DE2-115, ficheiro "master.qsf"). Quando se prime o botão KEY(0) é gerado um flanco ascendente ou descendente no sinal de relógio "clk" do *flip-flop*?

[TPC1] Altere o ficheiro "FlipFlopD.vhd" de modo a modelar um *flip-flop* D com entradas "set" e "reset" assíncronas, em que a entrada "reset" tenha a prioridade mais elevada. Simule o funcionamento desta versão do *flip-flop* D.

[TPC2] Crie um novo projeto que lhe permita modelar e simular o funcionamento de uma *latch* de 1 bit.

### Parte II

1. Crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como "Register\_Demo".

2. Descreva em VHDL um registo de 8 bits, com a interface representada na Figura 3. Guarde o seu código num ficheiro com o nome "Register8.vhd". Sugestão: utilize como base o código fonte do *flip-flop* D fornecido na parte 1, adaptando a entrada e a saída para vetores de 8 bits.

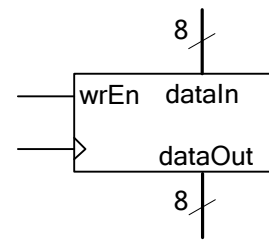


Figura 3 – Interface do módulo "Register8.vhd".

3. Efetue a simulação do componente modelado, realizando os passos habituais.

4. Após a simulação crie um novo símbolo para o seu registo de 8 bits. Crie em seguida um novo ficheiro, chamado "Register\_Demo.bdf" de modo a instanciar graficamente o seu registo de 8 bits. Ligue as entradas "dataIn" a 8 interruptores (SW[7..0]), a entrada "wrEn" a SW[8], a entrada "clk" ao botão KEY[0] e as saídas "dataOut" a 8 LEDs (e.g. verdes).

**Nota: O pontos 5 deve ser feito apenas na Aula Nº5/6. Passe para a parte III.**

5. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do componente.

### Parte III

1. Crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como "Counter\_Demo".

2. O código VHDL apresentado na Figura 5 implementa um contador binário natural em modo *count down*, cuja interface é apresentada na Figura 4. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome "CounterDown4.vhd".

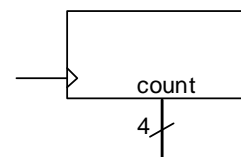


Figura 4 – Interface do módulo "CounterDown4.vhd".

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity CounterDown4 is
    port (clk    : in  std_logic;
          count  : out std_logic_vector(3 downto 0));
end CounterDown4;

architecture Behavioral of CounterDown4 is
    signal s_count : unsigned(3 downto 0);
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
            s_count <= s_count - 1;
        end if;
    end process;
    count <= std_logic_vector(s_count);
end Behavioral;

```

Figura 5 – Código VHDL de um contador binário natural em modo *count down*.

3. Efetue a simulação do componente modelado. Para especificar o vetor associado à entrada "clk" pode usar a opção de inicialização "Overwrite Clock" com um período de 20 ns.

4. Copie o ficheiro "CounterDown4.vhd" para o ficheiro "CounterUpDown4.vhd" e altere-o de modo a modelar um contador *up/down* de 4 bits, de acordo com a interface da Figura 6, em que a entrada "reset" é síncrona.

5. Efetue a simulação do novo componente modelado.

6. Crie um símbolo para o contador "CounterUpDown4".

7. Crie um novo ficheiro, chamado "Counter\_Demo.bdf", onde deverá instanciar o contador *up/down* modelado no ponto 4 e associar os respetivos portos a pinos concretos da FPGA do kit de desenvolvimento ("clk" ligado a KEY[0], restantes entradas ligadas a interruptores e saídas ligadas a LEDs).

**Nota: Os pontos 8, 10 e 14 devem ser feitos apenas na Aula Nº5/6. Passe para a parte IV.**

8. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do componente.

9. Pretende-se que o valor de saída do contador seja simultaneamente mostrado nos LEDs e no *display* mais significativo da placa. Para isso, instancie no ficheiro "Counter\_Demo.bdf" o decodificador de binário para sete segmentos que implementou na aula 2 ("Bin7SegDecoder") e faça as demais alterações que permitam cumprir esse objetivo.

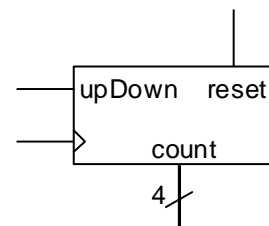


Figura 6 – Interface do módulo "CounterUpDown4.vhd".

**10.** Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do projeto.

**11.** Copie o ficheiro "CounterUpDown4.vhd" para o ficheiro "CounterLoadUpDown4.vhd". Altere-o de modo a modelar um contador *up/down* de 4 bits com entradas síncronas de "reset" e de "load" e ainda uma entrada de "enable" de acordo com o bloco lógico da figura. A entrada "reset" deve ter a prioridade mais elevada, seguida das entradas "enable", "load" e "upDown".

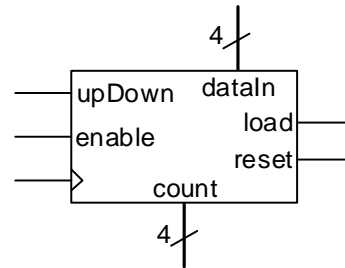


Figura 7 – Interface do módulo "CounterLoadUpDown4".

**12.** Efetue a simulação do novo componente modelado.

**13.** Crie um novo símbolo para o seu novo contador. Adicione a instanciação deste contador ao ficheiro *top-level* "Counter\_Demo.bdf" e associe os respetivos portos a pinos concretos da FPGA do *kit* de desenvolvimento (e.g. "c1k" ligado a KEY[1], restantes entradas ligadas a interruptores e saídas ligadas a LEDs e ao *display* de 7 segmentos como descrito em 9).

**14.** Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do componente.

#### Parte IV

A geração de um sinal de frequência mais baixa a partir de um de frequência mais alta é um processo de grande utilidade em sistemas digitais, designado por divisão de frequência. Por exemplo, um divisor de frequência por 3 produz na saída um sinal com um período igual a 3 vezes o período do sinal de entrada (ou seja, gera na saída um ciclo completo por cada 3 ciclos do sinal de entrada). Um contador simples gera nas suas saídas sinais que correspondem a divisões do sinal de relógio de entrada por 2, 4, 8, 16, ..., i.e. divisões por potências inteiras de 2. A divisão por outros valores requer uma especialização do circuito de contagem.

Nesta parte do guião pretende-se implementar um divisor de frequência por um valor "*k*" igual ou superior a 2, sendo que "*k*" deve ser o valor colocado num porto de entrada de 32 bits. O funcionamento do divisor de frequência em que a saída apresenta um *duty-cycle* de 50%<sup>1</sup> (no caso em que "*k*" é par) pode ser descrito do seguinte modo:

- o contador de módulo "*k*" é incrementado a cada transição ativa de relógio (por exemplo a transição de '0' para '1');
- quando o valor da contagem atingir o valor " $k/2 - 1$ " (i.e. metade da contagem), a saída é colocada ao nível lógico '1';
- quando o valor da contagem atingir o valor de " $k-1$ " (i.e. o valor final da contagem), a saída é colocada ao nível lógico '0' e é feito o *reset* do contador.

<sup>1</sup> O *duty-cycle* representa a percentagem de tempo a que um sinal está ao nível lógico '1' durante um período. Por exemplo, num sinal com 50% de *duty-cycle*, o tempo durante o qual o sinal está ao nível lógico '1' é metade do período e é portanto igual ao tempo a que está ao nível lógico '0'.

1. Crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como "FreqDivider\_Demo".
2. O código VHDL apresentado na Figura 8 mostra o esboço da implementação de um divisor de frequência. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome "FreqDivider.vhd". Complete o código fornecido, tendo em consideração a descrição funcional do divisor de frequência apresentada acima.
3. Efetue a simulação do componente modelado (especifique o vetor associado à entrada "clkIn" usando a opção de inicialização "Overwrite Clock" com um período de 20 ns). Se os resultados da simulação não corresponderem aos esperados, faça as alterações ao código que achar convenientes e repita a simulação. Para testar com diferentes constantes de divisão altere o valor atribuído à entrada "k".
4. Crie um novo ficheiro VHDL, chamado "FreqDivider\_Demo.vhd", onde deverá instanciar o divisor de frequência e associar os respetivos portos a pinos concretos da FPGA do *kit* de desenvolvimento: "clkIn" ligado a CLOCK\_50 que é a linha de relógio principal da FPGA e que apresenta um sinal com uma frequência de 50 MHz ( $50 \times 10^6$ ); "clkOut" ligado a um LEDR. Instancie o divisor de frequência atribuindo à entrada "k" o valor **x"017D7840"** ( $25 \times 10^6$ ).

**Nota: O ponto 5 deve ser feito apenas na Aula Nº5/6. Passe para a parte V.**

5. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do componente.

```
entity FreqDivider is
    port(clkIn  : in  std_logic;
          k      : in  std_logic_vector(31 downto 0);
          clkOut : out std_logic);
end FreqDivider;

architecture Behavioral of FreqDivider is
    signal s_counter : unsigned(31 downto 0);
    signal s_halfWay : unsigned(31 downto 0);
begin
    s_halfWay <= (unsigned(k) / 2);

    process(clkIn)
    begin
        if (rising_edge(clkIn)) then

            -- Complete o código

        end if;
    end process;
end Behavioral;
```

Figura 8 – Código VHDL de um divisor de frequência de um sinal de relógio.

### Parte V

1. Altere o ficheiro "FreqDivider\_Demo.vhd" de modo a instanciar o contador *up/down* de 4 bits que implementou na parte III, com a saída a ser visualizada num *display* de 7 segmentos.

O sinal de relógio do contador deverá ter a frequência de 1 Hz, obtido por divisão de frequência do relógio principal da FPGA (50 MHz).

**Nota: O ponto 2 deve ser feito apenas na Aula Nº5/6.**

**2.** Efetue a síntese e implementação do projeto, programe a FPGA e teste o seu funcionamento.

**PDF criado em 29/03/2021 às 11:01:12**