Laboratório de Sistemas Digitais

Trabalho Prático nº 4

Circuitos sequenciais elementares: flip-flops, registos, contadores e divisores de frequência

Objetivos

- Modelação em VHDL, simulação, implementação em FPGA e teste de circuitos sequenciais elementares.
- Parametrização de componentes.

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 dimensão parametrizável. 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 sequencial simples.

Parte I

1. Abra a aplicação "Altera Quartus II" e crie um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como "**FlipFlopD_Demo**".

2. O código VHDL que se apresenta de seguida 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;
```

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 FlipFlop 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 esquemático 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 saídas ligadas a LEDs do kit DE2-115). Para esse efeito é fornecido o seguinte código base que **apresenta, intencionalmente, erros de sintaxe**. Edite o ficheiro e corrija esses erros.

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

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 "**DE2_115.qsf**"). Quando se prime o botão é gerado um flanco ascendente ou descendente no sinal de relógio?

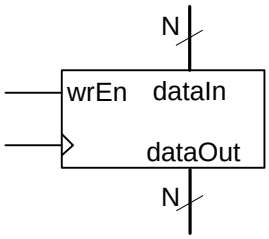
TPC:

9. Altere o ficheiro "**FlipFlopD.vhd**" de modo a modelar um FlipFlop 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.
10. Crie um novo projeto que lhe permita modelar e simular o funcionamento de uma *latch* de 4 bits.

Parte II

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

2. Descreva em VHDL um registo de "**N**" bits, parametrizável, com a interface representada na figura ao lado. A constante de parametrização poderá ter o valor por omissão 4. Guarde o seu código num ficheiro com o nome "**RegisterN.vhd**". Sugestão: utilize como base o código fonte do Flip Flop D fornecido na parte 1.



3. Efetue a simulação do componente modelado, realizando os passos já apresentados anteriormente.

4. Crie um novo ficheiro VHDL, chamado "**Register_Demo.vhd**" de modo a que instancie o registo parametrizável com uma dimensão de 8 (i.e. deve atribuir, aquando da instanciação e usando a construção **generic map**, o valor 8 à constante genérica "**N**" - ver slides da aula teórico-prática 3). Ligue as entradas "**dataIn**" a 8 interruptores, **SW(7:0)**, a entrada "**wrEn**" a **SW(8)**, a entrada "**clk**" ao pulsador **KEY(0)** e as saídas "**dataOut**" a 8 LEDs (e.g verdes).

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

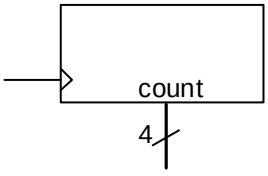
TPC:

6. Crie no ficheiro " **Register_Demo.vhd**" mais uma instância do registo parametrizável com uma dimensão de 6 (i.e. deve atribuir, aquando da instanciação e usando a construção **generic map**, o valor 6 à constante genérica "**N**"). Ligue as entradas "**dataIn**" a 6 interruptores, **SW(14:9)**, a entrada "**wrEn**" a **SW(15)**, a entrada "**clk**" ao pulsador **KEY(1)** e as saídas "**dataOut**" a 6 LEDs (e.g. vermelhos).

Parte III

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

2. O código VHDL que se apresenta de seguida implementa um contador binário natural em modo *count down*. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome "**CounterDown4.vhd**".



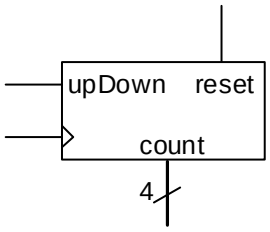
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

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;
```

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 50 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 o bloco lógico da figura, em que a entrada "**reset**" é síncrona.



5. Efetue a simulação do novo componente modelado.

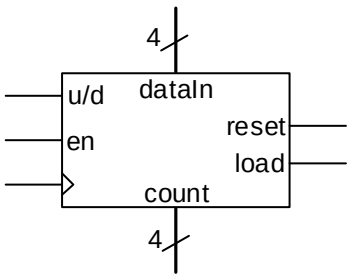
6. Crie um novo ficheiro VHDL, chamado "**Counter_Demo.vhd**", onde deverá instanciar o contador *up/down* modelado no ponto 4 e associar os respetivos portos a pinos concretos da FPGA do kit de desenvolvimento ("**clk**" ligado a **KEY(3)**, restantes entradas ligadas a interruptores e saídas ligadas a LEDs).

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

8. 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.vhd**" o descodificador de binário para sete segmentos que implementou na aula 3 ("**Bin7SegDecoder**") e faça as demais alterações que permitam cumprir esse objetivo.

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

10. Copie o ficheiro "**`CounterUpDown4.vhd`**" para o ficheiro "**`CounterLoadUpDown4.vhd`**" e 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.



11. Efetue a simulação do novo componente modelado.
12. Instancie, no *top level*, o novo contador *up / down* modelado no ponto 10 e associe os respetivos portos a pinos concretos da FPGA do kit de desenvolvimento (e.g. "**`clk`**" ligado a **`KEY(3)`**, restantes entradas ligadas a interruptores e saídas ligadas a LEDs e ao *display* de 7 segmentos como descrito em 8).
13. Efetue a síntese e implementação do projeto, programe a FPGA e teste o funcionamento do componente.

TPC:

14. Copie o ficheiro "**`CounterUpDown4.vhd`**" para "**`CounterUpDownN.vhd`**" e altere-o de modo a permitir a parametrização da dimensão do contador.
15. Efetue a simulação do novo 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 entrada por 2, 4, 8, 16, ..., i.e. divisões por potências inteiras de 2. A divisão por outras constantes requer uma especialização do circuito de contagem.

Nesta parte do guião pretende-se implementar um divisor de frequência por uma constante "**`K`**" igual ou superior a 2. O funcionamento do divisor de frequência em que a saída apresenta um *duty-cycle* de 50%[1] (no caso em que "**`K`**" é par) pode ser descrito do seguinte modo:

- o contador 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.

[1] 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 igual ao tempo a que está ao nível lógico '0').

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

2. O código VHDL que se apresenta mais abaixo 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 que se apresentou anteriormente.

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 50 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 por omissão da constante "**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 (50E6); "**clkOut**" ligado a um LED. Instancie o divisor de frequência atribuindo a "**K**" o valor 25000000 (25E6).

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

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

entity FreqDivider is
   generic(K : positive := 2);
   port(clkIn  : in  std_logic;
        clkOut : out std_logic);
end FreqDivider;

architecture Behavioral of FreqDivider is
   signal s_counter : natural;
   constant halfWay : natural := K/2-1;
begin
   process(clkIn)
   begin
      if( rising_edge(clkIn) ) then
      -- Complete o código
      end if;
   end process;
end Behavioral;
```

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 5 Hz, obtido por divisão de frequência do relógio principal da FPGA (50 MHz).

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