Laboratório de Sistemas Digitais

Trabalho Prático nº 2

Modelação em VHDL, simulação e implementação de componentes combinatórios (multiplexadores, descodificadores e codificadores)

Objetivos

- Modelação em VHDL, simulação, implementação em FPGA e teste no *kit* DE2-115 de componentes combinatórios simples.

Sumário

Este trabalho prático está dividido em quatro partes. A primeira é dedicada a descodificadores, a segunda a multiplexadores, a terceira a um caso particular de descodificador (binário - 7 segmentos) e a quarta a codificadores de prioridade. Por conveniência, a primeira parte utiliza exclusivamente descrições em VHDL, enquanto as restantes combinam diagramas lógicos para as descrições *top-level* e VHDL para os módulos combinatórios utilizados.

Notas importantes:

- Neste guião, os passos do fluxo de projeto são descritos de forma sumária; consulte o guião anterior para esclarecer eventuais dúvidas.
- No final da aula, desligue o *kit* e arrume-o na respectiva caixa, juntamente com os cabos e o alimentador.
- Nos pontos de TPC efetue a simulação e compilação prévia do projeto no seu PC antes de se dirigir à sala do DETI onde se encontram os *kits*. Desta forma utilizará os *kits* disponíveis de uma forma mais eficiente.

Parte I

1. Abra a aplicação *"Altera Quartus Prime"* e crie um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. O nome do projeto e da entidade *top-level* deverão ser ambos "Dec2_4EnDemo".

2. Crie um novo ficheiro para código-fonte VHDL e introduza o código da Figura 1.

3. Grave o ficheiro com o nome "Dec2_4En.vhd".

4. Identifique o componente modelado e construa no seu *log book* a respetiva tabela de verdade.

5. Efetue a simulação do componente modelado, realizando para tal os seguintes passos:

- selecione o ficheiro "Dec2_4En.vhd" como o *top-level* do projeto, de forma a efetuar a simulação apenas deste módulo.
- execute a opção "*Analysis & Synthesis*" para que, entre outros aspetos, seja verificada a correção sintática e a estrutura do projeto.
- crie um ficheiro VWF de suporte à simulação.
- selecione os portos a usar na simulação e especifique os vetores de teste ao longo do tempo. Sugestão: para a entrada "**`inputs`**", utilize a opção "*Random Values*"; para

"**enable**", use o 'rato' (tal como na aula 1) para definir janelas temporais de (in)atividade deste sinal.

- grave o ficheiro com o nome "Dec2_4En.vwf", execute a simulação e analise os resultados.

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

entity Dec2_4En is
    port(enable  : in  std_logic;
         inputs  : in  std_logic_vector(1 downto 0);
         outputs : out std_logic_vector(3 downto 0));
end Dec2_4En;

architecture BehavEquations of Dec2_4En is
begin
    outputs(0) <= enable and (not inputs(1)) and (not inputs(0));
    outputs(1) <= enable and (not inputs(1)) and (    inputs(0));
    outputs(2) <= enable and (    inputs(1)) and (not inputs(0));
    outputs(3) <= enable and (    inputs(1)) and (    inputs(0));
end BehavEquations;
```

Figura 1 – Código VHDL da entidade **Dec2_4En** e arquitetura **BehavEquations**.

6. Crie um novo ficheiro VHDL, chamado "Dec2_4EnDemo.vhd", onde deverá instanciar o componente modelado **Dec2_4En** e associar os respetivos portos a pinos concretos da FPGA do *kit* de desenvolvimento DE2-115 que vai usar para o testar (entradas ligadas a interruptores e saídas ligadas a LEDs). O código base para este efeito, fornecido na Figura 2, foi escrito intencionalmente com alguns erros de sintaxe. Após corrigir esses erros, grave o ficheiro.

```
entity Dec2_4EnDemo is
    port(SW   : std_logic_vector(2 downto 0);
         LEDG : std_logic_vector(3 downto 0));

architecture Shell of Dec2_4EnDemo is
begin
    system_core : work entity.Dec2_4En(BehavEquations)
                  port map(enable  <= SW(2);
                           inputs  <= SW;
                           outputs => LEDG(3 downto 0);
end Dec2_4EnDemo;
```

Figura 2 – Código VHDL para instanciação do módulo **Dec2_4En** e ligação a pinos da FPGA.

7. Selecione o ficheiro "Dec2_4EnDemo.vhd" como o novo *top-level* do projeto.

8. Importe as definições de pinos da FPGA do *kit* DE2-115 (ficheiro "DE2_115.qsf").

9. Efetue a síntese e implementação do projeto através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA e teste o funcionamento do componente.

10. Visualize e interprete o esquema lógico do circuito resultante da síntese lógica, usando para tal a ferramenta disponível no menu "*Tools→Netlist Viewers→RTL Viewer*".

11. Visualize agora e interprete o esquema lógico do circuito resultante do mapeamento da *netlist* nas primitivas da FPGA, usando para tal a ferramenta disponível no menu "*Tools→Netlist Viewers→Technology Map Viewer (Post-Mapping)*". Efetue uma análise comparativa deste diagrama com o visualizado no ponto anterior.

12. Edite o ficheiro "Dec2_4En.vhd" acrescentando no final a arquitetura (implementação) alternativa ("**BehavAssign**") da Figura 3. Nesta arquitetura são usadas atribuições condicionais para modelar o componente em vez das atribuições convencionais (concorrentes e incondicionais) usadas anteriormente para escrever as equações das saídas.

```
architecture BehavAssign of Dec2_4En is
begin
    outputs <= "0000" when (enable = '0')  else
               "0001" when (inputs = "00") else
               "0010" when (inputs = "01") else
               "0100" when (inputs = "10") else
               "1000";
end BehavAssign;
```

Figura 3 – Código VHDL da arquitetura **BehavAssign** para o módulo **Dec2_4En**.

13. No ficheiro "Dec2_4EnDemo.vhd" altere a instanciação do componente por forma a que seja agora usada a arquitetura "**BehavAssign**" em vez da "**BehavEquations**".

14. Repita os pontos 9 a 11, realizando a síntese, implementação, programação da FPGA e teste para a implementação alternativa do componente (arquitetura "**BehavAssign**").

15. Feche a aplicação de programação da FPGA e seguidamente o projeto.

[TPC] Repita os pontos 9 a 11 para a arquitetura "**BehavProc**" apresentada na Figura 4 (baseada num processo em VHDL).

```
architecture BehavProc of Dec2_4En is
begin
    process(enable, inputs)
    begin
        if (enable = '0') then
            outputs <= "0000";
        else
            if (inputs = "00") then
                outputs <= "0001";
            elsif (inputs = "01") then
                outputs <= "0010";
            elsif (inputs = "10") then
                outputs <= "0100";
            else
                outputs <= "1000";
            end if;
        end if;
    end process;
end BehavProc;
```

Figura 4 – Código VHDL da arquitetura **BehavProc** para o módulo **Dec2_4En**.

Parte II

1. Escreva no seu *log book* código VHDL para definir um multiplexador 2:1 com interface de acordo com a Figura 5 e arquitetura baseada num processo (*process*).

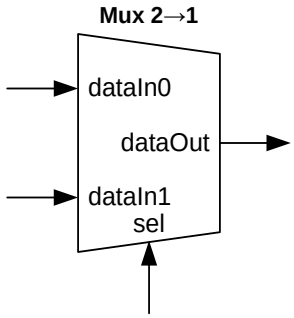

Figura 5 – Interface do multiplexador 2→1.

2. Crie no *"Altera Quartus Prime"* um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. Designe-o "Mux2_1Demo", bem como a sua entidade *top-level*.

3. Grave o código-fonte VHDL que preparou no ponto 1 num novo ficheiro com o nome "Mux2_1.vhd".

4. Efetue a simulação do multiplexador, realizando todos os passos necessários e especificando uma adequada sequência de vetores de entrada para validar o seu comportamento.

5. Crie um símbolo para poder usar o módulo "Mux2_1.vhd" em esquemas lógicos.

6. Crie um novo ficheiro para um esquema lógico, chamado "Mux2_1Demo.bdf". Instancie nele o multiplexador e associe os seus portos a pinos concretos da FPGA do *kit* DE2-115 que vai usar para o testar (sugere-se que ligue as entradas de dados a interruptores, a entrada de seleção a um botão e a saída a um LEDs, como mostra a Figura 6).

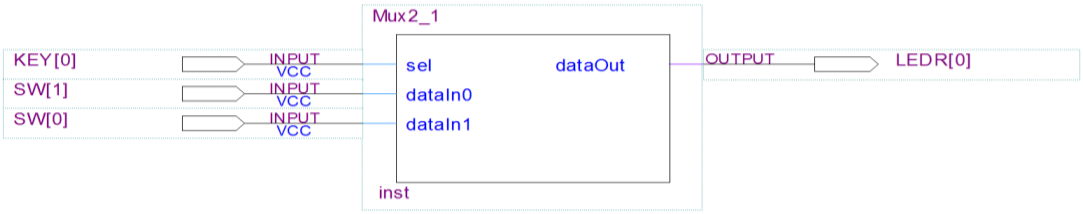

Figura 6 – Instanciação do módulo "Mux2_1" e ligação a pinos da FPGA.

7. Importe as definições de pinos da FPGA do *kit* DE2-115 (ficheiro "DE2_115.qsf").

8. Efetue a síntese e implementação do projeto através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA do *kit* e teste o multiplexador.

9. Feche a aplicação de programação da FPGA e seguidamente o projeto.

10. Repita a parte II para um multiplexador 4:1; note que a entrada de seleção **`sel`** passa a ser um vetor (**`std_logic_vector`**) de 2 *bits*.

[TPC] Repita a parte II, mas usando agora uma abordagem de modelação do multiplexador 4:1 baseada em atribuições condicionais.

Parte III

Nesta parte do trabalho prático, pretende-se implementar e testar um módulo muito útil para visualização de dados em sistemas digitais. Trata-se do descodificador binário→7 segmentos, cuja função é converter palavras binárias de entrada de 4 *bits*, representando valores entre 0 e 9 (ou entre 0 e 15), em padrões de saída de 7 *bits* destinados a controlar os LEDs de um *display* de 7 segmentos. No *kit* DE2-115, criaremos um sistema de teste muito simples, com os 4 *bits* de entrada controlados por interruptores e os 7 *bits* de saída ligados ao *display* HEX0. Considere o esquema de ligações entre este *display* e a FPGA, mostrado na Figura 7. Os restantes *displays* do *kit* (HEX1 - HEX7) possuem esquemas de ligação semelhantes – vide manual do utilizador disponível no site de LSDig.

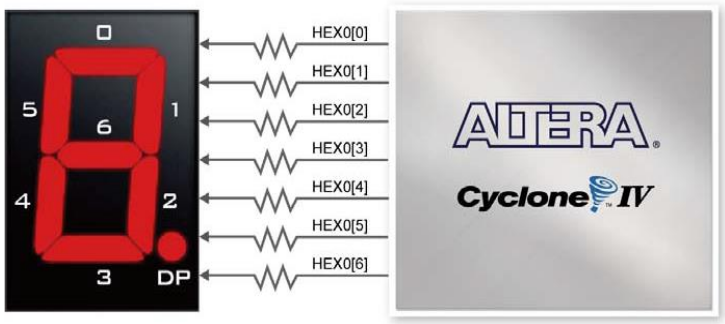

Figura 7 – Ligações entre a FPGA e o *display* HEX0 do *kit* DE2-115.

A Figura 8 apresenta uma possível implementação em VHDL (baseada em atribuições condicionais) de um descodificador binário para 7 segmentos.

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

entity Bin7SegDecoder is
    port(binInput : in  std_logic_vector(3 downto 0);
         decOut_n : out std_logic_vector(6 downto 0));
end Bin7SegDecoder;

architecture Behavioral of Bin7SegDecoder is
begin
    decOut_n <= "1111001" when (binInput = "0001") else  --1
                -- determine o valor das saídas para o dígito 2
                "0110000" when (binInput = "0011") else  --3
                "0011001" when (binInput = "0100") else  --4
                "0010010" when (binInput = "0101") else  --5
                "0000010" when (binInput = "0110") else  --6
                "1111000" when (binInput = "0111") else  --7
                "0000000" when (binInput = "1000") else  --8
                "0010000" when (binInput = "1001") else  --9
                "0001000" when (binInput = "1010") else  --A
                "0000011" when (binInput = "1011") else  --B
                "1000110" when (binInput = "1100") else  --C
                -- determine o valor das saídas para o dígito D
                "0000110" when (binInput = "1110") else  --E
                "0001110" when (binInput = "1111") else  --F
                "1000000";                               --0
end Behavioral;
```

Figura 8 – Código VHDL do descodificador binário→7 segmentos.

1. Analise o código da Figura 8 e identifique o nível lógico ('0' ou '1') que o descodificador aplica ao *display* para ativar um segmento.

2. Note que a descodificação é apresentada para todos os dígitos hexadecimais, exceto para os dígitos "2" e "D". Determine o valor das saídas para esses dígitos e efetue as modificações necessárias nas linhas de código comentadas.

3. Crie no *"Altera Quartus Prime"* um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. O nome do projeto e da entidade *top-level* deverão ser ambos "DisplayDemo".

4. Crie um novo ficheiro para código-fonte VHDL, introduza o código apresentado na Figura 8 e grave-o com o nome "Bin7SegDecoder.vhd".

5. Crie um símbolo para o módulo "Bin7SegDecoder.vhd", de forma a poder ser usado num diagrama lógico.

6. Num novo ficheiro chamado "DisplayDemo.bdf", crie um esquema lógico para instanciar o descodificador binário→7 segmentos e associar os respetivos portos a pinos concretos da FPGA do *kit* de desenvolvimento. Como mostra a Figura 9, ligue as entradas a interruptores e as saídas aos sinais de controlo dos segmentos do *display* HEX0.

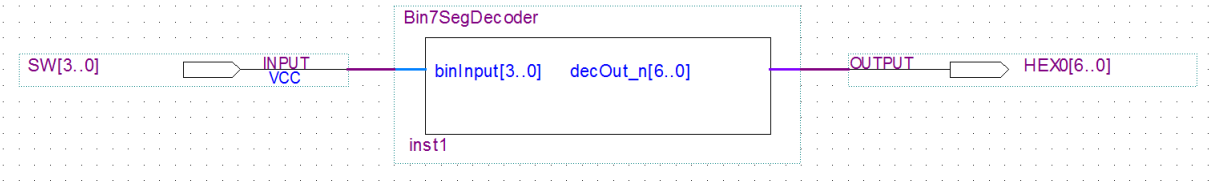

Figura 9 – Instanciação do módulo "Bin7SegDecoder" e ligação a pinos da FPGA.

7. Importe as definições de pinos da FPGA do *kit* de desenvolvimento (ficheiro "DE2_115.qsf").

8. Efetue a síntese e implementação do projeto através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA e teste o funcionamento do sistema.

9. Altere o projecto para que as entradas e as saídas do descodificador binário→7 segmentos sejam também visualizadas em LEDs do *kit* (observe a sugestão da Figura 10: quer os portos de entrada, onde ligam os interruptores, quer as saídas do descodificador passam também a ligar a portos de saída associados aos LEDs).

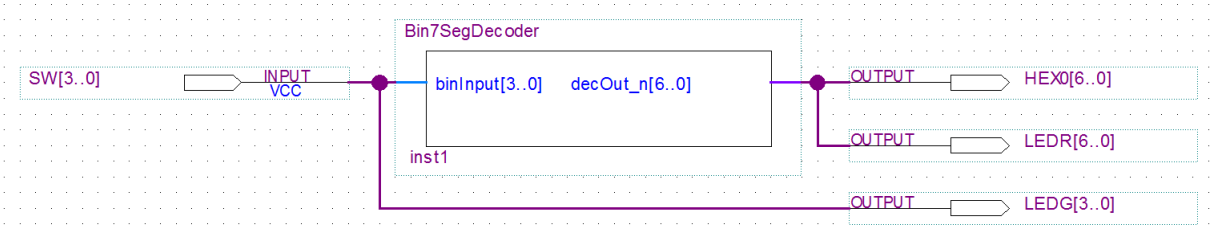

Figura 10 – Alteração do circuito da Figura 9 para visualização em LEDs das entradas e saídas do descodificador.

10. Importe novamente o ficheiro "DE2_115.qsf" e repita a síntese e implementação do projeto através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA e teste o funcionamento do sistema.

11. Adicione ao descodificador uma entrada de *enable* que, quando ativa, habilite o seu funcionamento normal e, quando inativa, iniba (desative) todas as saídas, apagando todos os segmentos, independentemente das restantes entradas do descodificador.

12. Volte a criar o símbolo do módulo "Bin7SegDecoder.vhd" de forma a atualizá-lo com a nova entrada (*enable*).

13. Edite o módulo *top-level* do projeto ("DisplayDemo.bdf") para que a entrada *enable* do descodificador seja controlada por um interruptor do *kit* (e.g. KEY(0) – vide Figura 11). Como a interface do módulo "Bin7SegDecoder.vhd" foi alterada, deverá voltar a instanciá-lo.

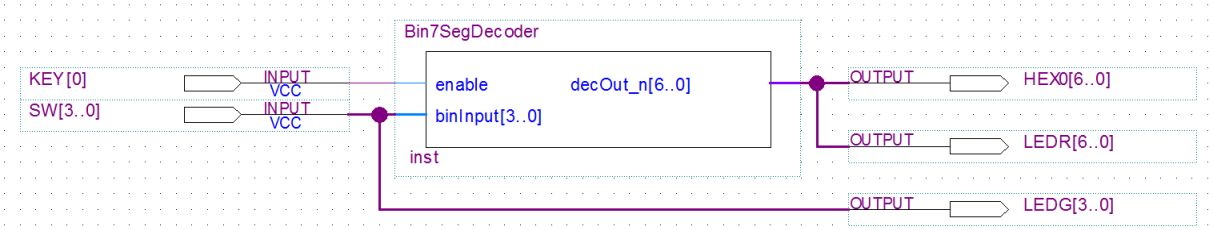

Figura 11 – Instanciação do módulo "Bin7SegDecoder" com *enable* e ligação a pinos da FPGA.

14. Volte a importar o ficheiro "DE2_115.qsf", a sintetizar e a implementar o projeto através do comando "*Compile Design*". No final programe a FPGA e teste o funcionamento do sistema.

15. Feche a aplicação de programação da FPGA e seguidamente o projeto.

[TPC] Repita o exercício a partir de 6 usando VHDL em vez do diagrama lógico para definir o módulo *top-level* ("DisplayDemo.vhd").

Parte IV

1. Construa no seu *log book* a tabela de verdade de um codificador de prioridade 4→2 com indicação de saída válida. A interface deste componente é apresentada na Figura 12.

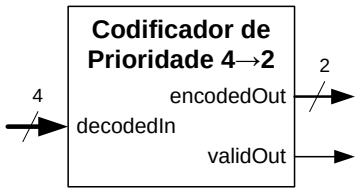

Figura 12 – Interface de codificador de prioridade.

2. Escreva no seu *log book* código VHDL para este codificador de prioridade, com arquitectura baseada num processo (*process*).

3. Crie no *"Altera Quartus Prime"* um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. O nome do projeto e da entidade *top-level* poderão ser ambos "PEnc4_2Demo".

4. Grave o código-fonte VHDL do ponto 2 num novo ficheiro designado "PEnc4_2.vhd".

5. Efetue a simulação do codificador, realizando todos os passos necessários e especificando uma sequência adequada de vetores de entrada para validar o seu comportamento.

6. Crie um símbolo para o módulo "PEnc4_2.vhd", de forma a poder ser usado num diagrama lógico.

7. Crie um novo ficheiro para um esquema lógico, chamado "PEnc4_2Demo.bdf", que irá servir para instanciar o codificador e associar os respetivos portos a pinos concretos da FPGA do *kit* de desenvolvimento DE2-115 que vai usar para o testar (sugere-se que ligue as entradas a interruptores e as saídas a LEDs).

8. Importe as definições de pinos da FPGA do *kit* DE2-115 (ficheiro "DE2_115.qsf").

9. Efetue a síntese e implementação do projeto através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA e teste o funcionamento do codificador.

10. Feche a aplicação de programação da FPGA e seguidamente o projeto.

[TPC] Repita a parte IV para um codificador de prioridade 16→4 com indicação de saída válida.

PDF criado em 20/02/2016 às 01:23:12