Laboratório de Sistemas Digitais

Trabalho Prático nº 3

Modelação em VHDL e implementação de circuitos aritméticos

Objetivos

- Modelação em VHDL, simulação, implementação em FPGA e teste de circuitos aritméticos.

Sumário

Este trabalho prático está dividido em cinco partes. Na primeira parte é abordada a implementação estrutural de somadores/subtratores. A segunda parte é dedicada à descrição comportamental e implementação de operações aritméticas e lógicas em quantidades sem sinal. Na terceira parte são utilizados *displays* de 7 segmentos do *kit* DE2-115 para visualização do resultado de operações aritméticas. A quarta parte é dedicada à representação da informação em BCD (decimal codificado em binário). Finalmente, a quinta parte é semelhante à segunda parte, mas incide sobre a realização de operações aritméticas em quantidades com sinal.

Parte I

1. Escreva no seu *log book* as equações lógicas das saídas "s" e "cout" de um somador completo de 1 bit, cuja interface é apresentada na Figura 1 (sugestão: consulte os slides da aula TP 1).

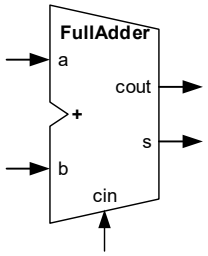

Figura 1 – Interface do somador completo de 1 bit (módulo "FullAdder").

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

3. Descreva em VHDL o somador completo de 1 bit e guarde num ficheiro com o nome "FullAdder.vhd". A entidade **`FullAdder`** e respetiva arquitetura **`Behavioral`** pode ser especificada de acordo com o esqueleto da Figura 2.

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

entity FullAdder is
    port(a, b, cin : in  std_logic;
         s, cout   : out std_logic);
end FullAdder;

architecture Behavioral of FullAdder is
begin
    -- Especifique aqui as equações lógicas para as saídas "s" e "cout"
end Behavioral;
```

Figura 2 – Esqueleto do código VHDL da entidade **`FullAdder`** e respetiva arquitetura **`Behavioral`**.

4. Desenhe no seu *log book* um somador *ripple carry* de 4 bits, cuja interface externa é apresentada na Figura 3, construído a partir de somadores completos de 1 bit. Quantos somadores de 1 bit serão necessários?

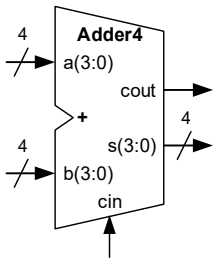

Figura 3 – Interface do somador de 4 bits (módulo "Adder4").

5. Identifique o número de sinais internos de *Carry* (C) que serão necessários para interligar os somadores de 1 bit entre si.

6. Descreva em VHDL estrutural um somador *ripple carry* de 4 bits construído com somadores completos de 1 bit conforme a estrutura que elaborou no ponto 4. Grave o ficheiro com o nome "Adder4.vhd". A respetiva especificação em VHDL deverá seguir a estrutura da Figura 4, baseada na instanciação de 4 somadores completos de um bit.

```
-- Inclua as bibliotecas e os pacotes necessários

entity Adder4 is
    port(a, b : in  std_logic_vector(3 downto 0);
         cin  : in  std_logic;
         s    : out std_logic_vector(3 downto 0);
         cout : out std_logic);
end Adder4;

architecture Structural of Adder4 is
    -- Declare um sinal interno (carryOut) do tipo std_logic_vector (de
    -- C bits) que interligará os bits de carry dos somadores entre si
begin
    bit0: entity work.FullAdder(Behavioral)
        port map(a    => a(0),
                 b    => b(0),
                 cin  => cin,
                 s    => s(0),
                 cout => carryOut(0));

    -- complete para os restantes bits (1 a 3)
end Structural;
```

Figura 4 – Esqueleto do código VHDL da entidade **`Adder4`** e arquitetura **`Structural`**.

7. Efetue a simulação do componente modelado ("Adder4"), realizando os seguintes passos:

- selecione o ficheiro "Adder4.vhd" como o *top-level* do projeto;
- execute a opção "*Analysis & Synthesis*" para que, entre outros aspetos, sejam analisadas a correção sintática e a estrutura do projeto;
- crie um ficheiro VWF de suporte à simulação e selecione os sinais/portos a usar na simulação e especifique os vetores de entrada a aplicar;
- grave o ficheiro com o nome "Adder4.vwf", execute a simulação e analise os resultados para vários valores das entradas (por exemplo, que resultados se obtêm para os seguintes conjuntos de entradas: "cin"='0', "a"="1111" e "b"="1111"; "cin"='1',

"a"="1111" e "b"="0000"; "cin"='0', "a"="1010" e "b"="1100"). Altere a base de representação usada na visualização dos operandos e resultado entre binário, decimal e hexadecimal.

8. Crie um símbolo para poder instanciar o módulo "Adder4.vhd" em diagramas lógicos.

9. Crie um novo ficheiro para um diagrama lógico, chamado "AdderDemo.bdf". Instancie nele o somador de 4 bits e associe os seus 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" e "b" a oito interruptores (entrada "a" a SW[7..4] e entrada "b" a SW[3..0]) e as saídas "cout" e "s" a cinco LEDs (saída "cout" a LEDR[4] e saída "s" a LEDR[3..0]). Ligue a entrada "cin" do somador "Adder4" ao nível lógico '0' (zero).

10. Selecione o ficheiro "AdderDemo.bdf" como o novo *top-level* do projeto.

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

12. 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 somador de 4 bits.

13. Crie um novo ficheiro VHDL, chamado "AddSub4.vhd", para implementação de um somador/subtrator de 4 bits, onde deverá instanciar o somador "Adder4" e a lógica adicional necessária de acordo com a Figura 5. A entidade deverá chamar-se **`AddSub4`** e a arquitetura **`Structural`**.

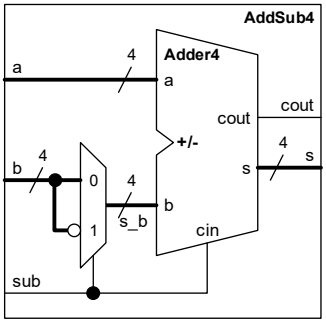

Figura 5 – Interface do somador/subtrator de 4 bits (módulo "AddSub4").

A entrada "sub" permite selecionar a operação do circuito. Quando "sub" = '0' o circuito realiza a soma aritmética ("a" + "b") e quando "sub" = '1' a subtração ("a" - "b"). O multiplexador usado para complementar o sinal "b" no caso da operação subtração, pode ser construído como um módulo autónomo, instanciado e ligado no "AddSub4" de acordo com a Figura 5. Alternativamente, uma forma mais expedita de adicionar o multiplexador consiste no acrescento da seguinte atribuição condicional no corpo da arquitetura do módulo "AddSub4", a qual descreve o mesmo multiplexador de forma comportamental:

```
s_b <= b when (sub = '0') else not b;
```

em que **`s_b`** deve ser um sinal de 4 bits do tipo **`std_logic_vector`** declarado na arquitetura do módulo "AddSub4".

14. Crie um símbolo para poder instanciar o módulo "AddSub4.vhd" em diagramas lógicos.

15. Adicione ao ficheiro "AdderDemo.bdf" a instanciação do módulo "AddSub4". Ligue as respetivas entradas "a" e "b" a oito interruptores do *kit* DE2-115 (entrada "a" a SW[17..14], e a entrada "b" a SW[13..10]), a entrada de controlo "sub" a KEY[0] e as saídas "cout" e "s" a cinco LEDs (saída "cout" a LEDR[14] e saída "s" a LEDR[13..10]).
Nota: após a instanciação do módulo "AddSub4" no ficheiro "AdderDemo.bdf" e ligação dos seus portos aos dispositivos do *kit* da forma indicada, passam neste caso a ser usados bits não contíguos dos periféricos LEDR e SW. Para garantir que a ferramenta de implementação (*fitter* do *"Quartus Prime"*) usa os pinos corretos, é importante ligar os bits não usados da forma ilustrada na Figura 6, explicitando as entradas não usadas (sem as ligar), assim como as saídas (ligando-as a um nível lógico fixo, e.g. '0'). Além disso, volte a importar o ficheiro "master.qsf" com as definições de pinos da FPGA do *kit*.

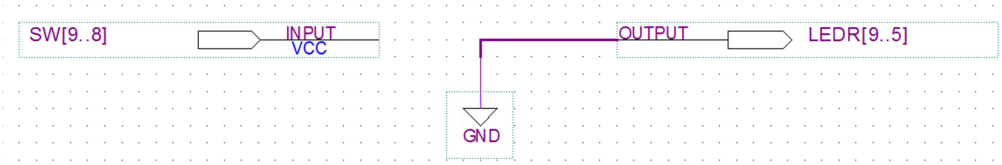

Figura 6 – Explicitação das entradas não usadas e ligação a um nível lógico fixo das saídas (no caso da utilização de conjuntos de bits não contíguos dos periféricos do *kit* DE2-115). O símbolo "GND" encontra-se na biblioteca de componentes do *"Quartus Prime"*, acessível através do *"Symbol Tool"*.

16. Efetue a síntese e implementação do projeto modificado, através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA e teste o funcionamento do somador/subtrator de 4 bits.

17. Adicione a arquitetura comportamental (**`Behavioral`**) da Figura 7 ao módulo "AddSub4" e coloque na forma de comentário o excerto de código relativo à arquitetura **`Structural`** do mesmo módulo (**Nota:** este e os próximos pontos pretendem ilustrar as capacidades da linguagem VHDL para a modelação comportamental das operações, remetendo para as ferramentas de síntese a geração dos circuitos que as implementam).

```
architecture Behavioral of AddSub4 is

  signal s_a, s_b, s_s : unsigned(4 downto 0);

begin
  s_a  <= '0' & unsigned(a);    -- '0's para capturar o cout do
  s_b  <= '0' & unsigned(b);    --  do bit mais significativo
  s_s  <= (s_a + s_b) when (sub = '0') else
          (s_a - s_b);
  s    <= std_logic_vector(s_s(3 downto 0));
  cout <= s_s(4);
end Behavioral;
```

Figura 7 – Arquitetura **`Behavioral`** do módulo **`AddSub4`**.

18. Atualize o símbolo e a instanciação do módulo "AddSub4" no diagrama lógico "AdderDemo.bdf" de forma a ser usada a implementação baseada na arquitetura **`Behavioral`** em vez da **`Structural`**.

19. Efetue a síntese e implementação do projeto modificado, através do comando "*Compile Design*". Programe a FPGA e teste o funcionamento do somador/subtrator de 4 bits baseado na síntese da descrição comportamental da Figura 7.

Parte II

Pretende-se desenvolver uma unidade aritmética e lógica (*Arithmetic and Logic Unit* – ALU) que processe operandos inteiros "a" e "b" de 4 bits sem sinal. A operação a executar é definida com uma entrada *"op"* de acordo com a Tabela 1.

op	Operação
000	adição
001	subtração
010	multiplicação sem sinal
011	quociente de divisão sem sinal
100	resto de divisão sem sinal
101	AND (bit-a-bit)
110	OR (bit-a-bit)
111	XOR (bit-a-bit)

Tabela 1 – Codificação binária das operações suportadas pela ALU.

1. Calcule no seu *log book* o resultado produzido pela ALU para cada uma das operações suportadas e assumindo os seguintes operandos: a = "1001" e b = "0011".

Para simplificar apresente os cálculos das operações aritméticas em decimal e das operações lógicas em binário.

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

3. Crie um novo ficheiro VHDL, chamado "ALU4.vhd" para introduzir o código da ALU fornecido na Figura 8, mas que possui intencionalmente erros de sintaxe. Edite e corrija os erros de sintaxe e no final grave o ficheiro.

Note que o código fornecido usa a atribuição seletiva de sinais (construção concorrente **`with ... select`** do VHDL), sendo portanto uma codificação alternativa à apresentada nos slides das aulas teórico-práticas. Analise e compare ambas as codificações.

4. Crie um símbolo para poder usar o módulo "ALU4.vhd" em diagramas lógicos.

5. Crie um novo ficheiro para um diagrama lógico, chamado "ALUDemo.bdf", para instanciar a ALU e associar os respetivos portos a pinos concretos da FPGA do *kit* de desenvolvimento DE2-115 que vai usar para a testar (sugere-se que ligue as entradas "a" e "b" a interruptores SW[7..4] e SW[3..0], respetivamente, a entrada "op" a interruptores SW[10..8] e as saídas "m" e "r" a LEDR[7..4] e LEDR[3..0], respetivamente).

6. Efetue a síntese e implementação do projeto através do comando "*Compile Design*". No final do processo de compilação, programe a FPGA e teste a ALU no *kit* de desenvolvimento.

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

entity ALU4
    port(a, b : input  std_logic_vetcor(3 downto 0);
         op   : input  std_logic_vetcor(2 downto 0);
         r, m : output std_logic_vetcor(3 downto 0));
end ALU4;

architecture Behavioral of ALU4 is

    signal s_a, s_b, s_r : unsigned(3 downto 0);
    signal s_m           : unsigned(7 downto 0);

benig
    s_a <= unsigned(a);
    s_b <= unsigned(b);

    s_m <= s_a * s_b;

    with op select
        s_r <= s_a  +  s_b     when "000",
               s_a  -  s_b     when "001",
               s_m(3 downto 0) when "010",
               s_a  /  s_b     when "011",
               s_a rem s_b     when "100",
               s_a and s_b     when "101",
               s_a or  s_b     when "110",
               s_a xor s_b     when "111";

    r <= std_logic_vector(s_r);

    m <= std_logic_vector(s_m(7 downto 4)) when (op = "010") else
         (others => '0');
end Beahvioral;
```

Figura 8 – Código fonte VHDL do módulo **ALU4**.

Parte III

Seria muito mais simples e amigável testar alguns dos circuitos aritméticos deste trabalho prático (somador e/ou ALU) se os resultados fossem também mostrados num ou mais *displays* de 7 segmentos na forma de dígitos hexadecimais (além dos LEDs que mostram o resultado em binário). Para implementar esta funcionalidade considere a explicação sobre o esquema de ligações entre o *display* de 7 segmentos HEX0 do *kit* DE2-115 e a FPGA, assim como o código fornecido no guião prático 2.

1. Para implementar a visualização nos *displays* do resultado "r" calculado pela ALU da parte II, elabore no seu *log book* um diagrama de blocos do sistema a implementar na FPGA, incluindo as entradas, saídas, descodificador para 7 segmentos e a ALU.
2. Com base no diagrama de blocos elaborado no ponto anterior e no projeto da parte II deste trabalho prático, acrescente ao projeto e edite os ficheiros necessários para construir o sistema. O ficheiro "ALUDemo.bdf" (*top-level*) deverá conter:

- instanciação dos módulos "ALU4" e "Bin7SegDecoder";
- ligação aos pinos concretos da FPGA (o resultado "r" deve ser mostrado em LEDR[3..0] e no *display* de 7 segmentos HEX0; o resultado "m" é mostrado apenas em LEDR[7..4]).

3. 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 da ALU e do *display* de 7 segmentos.

4. Altere o projeto de forma a que o valor da saída "m" da ALU seja também visualizada no *display* HEX1 do *kit* DE2-115.

5. Repita a síntese e implementação do projeto através do comando "*Compile Design*", no final programe a FPGA e teste o funcionamento da ALU e dos dois *displays* de 7 segmentos.

Parte IV

1. Crie uma cópia completa do projeto da parte anterior e modifique-o de modo a que o resultado "r" da ALU seja mostrado em *displays* de 7 segmentos em decimal em vez de hexadecimal. Para implementar esta funcionalidade é necessário converter o resultado "r[3..0]" de binário para BCD. Quantos dígitos BCD serão necessários?

2. Descreva em VHDL um módulo, chamado "Bin2BCD", que realize a conversão para BCD de uma quantidade binária de 4 bits do tipo *unsigned*. Faça a conversão recorrendo a um processo e a operações aritméticas e de comparação de VHDL que conhece.

3. Verifique por simulação o comportamento do módulo "Bin2BCD" e uma vez validado, crie um símbolo para o poder usar num diagrama lógico.

4. Esboce no seu *log book* o diagrama de blocos e de ligações do sistema incluindo a ALU, o conversor binário-BCD, descodificadores para 7 segmentos e portos de entrada e saída.

5. Modifique o ficheiro "ALUDemo.bdf" de forma a incluir o conversor binário-BCD e o número necessário de componentes "Bin7SegDecoder". Ligue as saídas dos componentes "Bin7SegDecoder"a *displays* (HEX0, HEX1, HEX2, ... – use tantos quantos os necessários).

6. Efetue a síntese, implementação e teste o projeto no *kit*.

[TPC] Modifique o projeto de modo a que o resultado "m" da ALU seja também mostrado em *displays* de 7 segmentos em decimal em vez de hexadecimal. Efetue a síntese, implementação e teste o projeto no *kit*.

Parte V

1. Crie uma cópia completa do projeto da parte II (replicando toda a estrutura de diretórios do projeto) e altere a réplica de forma a realizar operações aritméticas sobre quantidades com sinal. Compile o projeto e teste-o no *kit* DE2-115.

[TPC] Modifique o projeto de modo a que o resultado "r" e "m" da ALU seja também mostrado em *displays* de 7 segmentos em decimal em vez de hexadecimal. Efetue a síntese, implementação e teste o projeto no *kit*.

PDF criado em 18/03/2022 às 18:24:10