Laboratório de Sistemas Digitais

Trabalho Prático nº 3

Circuitos aritméticos e parametrização de componentes

Objetivos

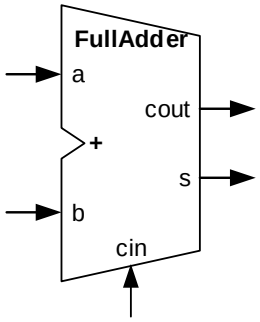
- Modelação em VHDL, simulação, implementação em FPGA e teste de circuitos aritméticos.
- Introdução à parametrização de componentes.

Sumário

Este trabalho prático está dividido em sete partes. Na primeira parte é abordada a implementação de somadores/subtratores. A segunda parte introduz a parametrização de componentes. A terceira parte é dedicada à descrição comportamental e implementação de operações aritméticas e lógicas com base em modelos VHDL inicialmente fixos e posteriormente parametrizáveis. Na quarta parte é solicitada a utilização dos *displays* de 7 segmentos do kit DE2-115 para visualização do resultado de operações aritméticas. A quinta parte é dedicada à representação da informação em BCD (decimal codificado em binário). A sexta e sétima partes contêm exercícios adicionais para trabalho de casa.

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 seguinte.



2. Abra a aplicação "Altera Quartus II" e crie um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. O nome do projeto deverá ser "AdderDemo".

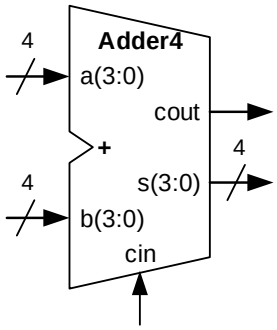
3. Descreva em VHDL o somador completo de 1 bit e guarde num ficheiro com o nome "FullAdder.vhd". A entidade **`FullAdder`** e respetiva arquitetura **`Behavioral`** pode ser especificada de acordo com o seguinte esqueleto:

```
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;
```

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



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 poderá seguir a seguinte estrutura, 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;
```

7. Efetue a simulação do componente modelado ("Adder4"), realizando para tal 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;

- 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, qual o resultado que se obtém para as seguintes entradas: *"cin"* = '0', *"a"* = "1111" e *"b"* = "1111"?).

8. Crie um novo ficheiro VHDL, chamado "AdderDemo.vhd", para instanciar o somador de 4 bits 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"* 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).

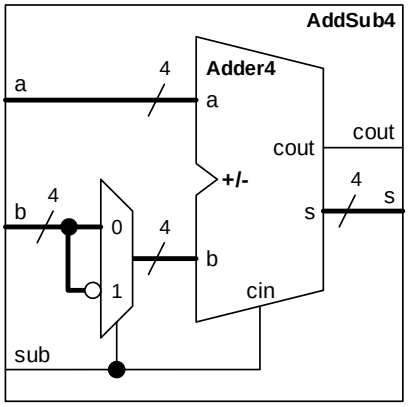
9. Selecione o ficheiro "AdderDemo.vhd" como o novo *top level* do projeto.

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

11. Efetue a síntese e implementação do projeto através do comando "*Compile Design*".

12. 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 de acordo com a figura seguinte. A entidade deverá chamar-se **`AddSub4`** e a arquitetura **`Structural`**.



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"*).

14. Modifique o ficheiro "AdderDemo.vhd" de forma a que seja instanciado o módulo "AddSub4" em vez do "Adder4" e que a entrada de controlo "sub" seja associada ao botão KEY(0) do kit DE2-115.

15. Efetue a síntese e implementação do projeto modificado, através do comando "*Compile Design*".

16. No final do processo de compilação, programe a FPGA e teste o funcionamento do somador/subtrator de 4 bits.

17. Adicione a seguinte nova arquitetura comportamental (**Behavioral**) ao módulo "Adder4".

```
architecture Behavioral of Adder4 is

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

begin
  s_a  <= '0' & unsigned(a);
  s_b  <= '0' & unsigned(b);
  s_s  <= s_a + s_b;
  s    <= std_logic_vector(s_s(3 downto 0));
  cout <= s_s(4);
end Behavioral;
```

18. Modifique a instanciação do módulo "Adder4" no ficheiro "AddSub4.vhd" de forma a que seja usada esta 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.

20. Adicione a seguinte nova arquitetura comportamental (**Behavioral**) ao módulo "AddSub4".

```
architecture Behavioral of AddSub4 is

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

begin
  s_a  <= '0' & unsigned(a);
  s_b  <= '0' & unsigned(b);
  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;
```

21. Modifique a instanciação do módulo "AddSub4" no ficheiro "AdderDemo.vhd" de forma a que seja usada esta arquitetura (**Behavioral**) em vez da **Structural**.

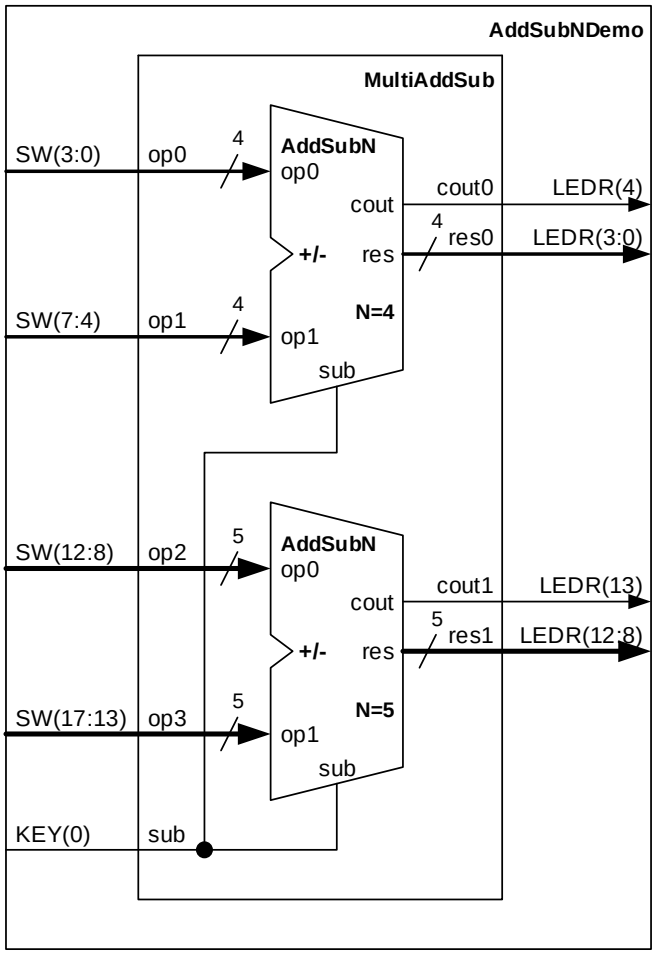
22. Efetue a síntese e implementação do projeto modificado, através do comando "*Compile Design*". Programe a FPGA e teste o funcionamento do somador/subtrator de 4 bits.

Estes últimos pontos pretendem ilustrar as capacidades do VHDL para a modelação comportamental das operações, remetendo paras as ferramentas de síntese a geração dos circuitos que as implementam.

Parte II

1. Crie um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. O nome do projeto deverá ser "AddSubNDemo".

2. Descreva em VHDL um somador/subtrator parametrizável de N bits de acordo com a interface do módulo "AddSubN" da figura seguinte. Baseie a sua descrição na arquitetura **`Behavioral`** do módulo "AddSub4" da parte I deste guião. Grave o ficheiro com o nome "AddSubN.vhd".



3. Crie um novo ficheiro VHDL com o nome "MultiAddSub.vhd" onde vai implementar o módulo "MultiAddSub", em que a arquitetura é construída com base em duas instanciações do módulo "AddSubN", uma com N=4 e outra com N=5. Este módulo permite realizar duas adições/subtrações sobre operandos de 4 e 5 bits, em que o sinal de controlo "sub" é comum.

4. Finalmente crie um novo ficheiro VHDL com o nome "AddSubNDemo" para instanciar o módulo "MultiAddSub" e efetuar a sua ligação a dispositivos do kit de acordo com a figura anterior.

5. Selecione o ficheiro "AddSubNDemo.vhd" como o *top level* do projeto.

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

7. Efetue a síntese e implementação do projeto através do comando "*Compile Design*".

8. No final do processo de compilação, programe a FPGA e teste o funcionamento do circuito.

Parte III

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

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)

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 "Altera Quartus II" um novo projeto para a FPGA Altera Cyclone IV EP4CE115F29C7. O nome do projeto deverá ser "ALUDemo".

3. Crie um novo ficheiro VHDL, chamado "ALU4.vhd". Por conveniência é fornecido o seguinte código base para este efeito, 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.

```
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;
```

4. Crie um novo ficheiro VHDL, "ALUDemo.vhd", 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(17:15) e as saídas *"m"* e *"r"* a LEDR(7:4) e LEDR(3:0), respetivamente).

5. Efetue a síntese e implementação do projeto através do comando "*Compile Design*" e teste a ALU no kit DE2-115.

6. Crie um novo ficheiro chamado "ALUN.vhd", baseado no "ALU4.vhd", em que dimensão dos operandos da ALU é parametrizável, de modo a que seja possível processar operandos de qualquer tamanho, definido com uma constante genérica *"N"* fixada aquando da instanciação da ALU. Para tal será necessário declarar a constante genérica *"N"* com a construção *generic*, inicializá-la com um valor por omissão (8 no código seguinte) e expressar as dimensões dos portos *"a"*, *"b"*, *"r"* e *"m"*, e dos sinais *"s_a"*, *"s_b"* e *"s_m"* em termos desta constante. O excerto de código seguinte ilustra algumas das modificações que devem ser realizadas.

```
...
entity ALUN
    generic(N : natural := 8);
    port(a, b : in std_logic_vector(N - 1 downto 0);
    ...
end ALUN;

architecture Behavioral of ALUN is

    signal s_a, s_b : unsigned(N - 1 downto 0);
    signal s_m      : unsigned((2 * N) - 1 downto 0);

    ...
```

7. Modifique o ficheiro VHDL, "ALUDemo.vhd" de modo a que instancie a unidade aritmética e lógica "ALUN" configurada para processar operandos de 4 bits. i.e. deve atribuir aquando da instanciação o valor 4 à constante genérica *"N"* com a construção *generic map* (ver slides da aula teórico-prática). Associe os respetivos portos a pinos concretos da FPGA do kit de desenvolvimento que vai usar (ligue as entradas *"a"* e *"b"* a interruptores SW(7:4) e SW(3:0), respetivamente, a entrada *"op"* a interruptores SW(17:15) e as saídas *"m"* e *"r"* a LEDR(7:4) e LEDR(3:0), respetivamente).

8. Efetue a síntese, implementação e teste da unidade aritmética e lógica.

9. Realize as alterações do projeto necessárias para que a ALU seja instanciada para operandos de 6 bits e sejam usados os interruptores e os LEDs do kit DE2-115 necessários para a testar.

10. Efetue a síntese, implementação e teste da nova instanciação da ALU.

Parte IV

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 da placa DE2-115 e a FPGA e o código fornecido no guião prático 2.

1. Para implementar a visualização nos displays do resultado *"r"* calculado pela ALU ("ALUN" instanciada com *"N"*=4), elabore no seu *log book* um diagrama de blocos do sistema a implementar na FPGA, incluindo as entradas, saídas, descodificador para 7 segmentos e ALU.

2. Com base no diagrama de blocos elaborado no ponto anterior e no projeto da parte III deste trabalho prático edite os ficheiros VHDL necessários para construir o sistema. O ficheiro "ALUDemo.vhd" deverá conter:

- instanciação da ALU configurada com *"N"* = 4;
- instanciação do descodificador "Bin7SegDecoder";

- ligação com os 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*".

4. No final do processo de compilação, programe a FPGA e teste o funcionamento da ALU e do *display* de 7 segmentos.

5. Crie um novo ficheiro VHDL, chamado "ALUDisplay.vhd" para instanciar a ALU (configurada com *"N"* = 4) e o descodificador "Bin7SegDecoder", encapsulando desta forma toda a funcionalidade do sistema num único módulo e deixando para o *top level* apenas a ligação dos portos a pinos concretos da FPGA.

Nota: Deixar para *o top level* apenas os aspetos relacionados com a FPGA e kit alvo (e.g. ligação das entradas e saídas do sistema a pinos concretos) é uma boa prática de modelação, ao nível da portabilidade e reutilização do código e, por isso, esta abordagem será adotada em LSDig.

6. Altere o ficheiro "ALUDemo.vhd" de modo a instanciar o módulo "ALUDisplay" e a ligá-lo a 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)).

7. Repita a síntese e implementação do projeto através do comando "*Compile Design*" e no final do processo de compilação, programe a FPGA e teste o funcionamento da ALU e do *display* de 7 segmentos que deve ser semelhante ao que observou em 4.

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

9. Efetue a síntese e implementação e teste no kit do projeto.

Parte V

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(N-1:0)"* de binário para BCD. Quantos dígitos BCD serão necessários (para o caso de *"N"* = 4)?

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. 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.

4. Modifique o ficheiro "ALUDemo.vhd" 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).

5. Efetue a síntese, implementação e teste do projeto.

Trabalho para Casa (pós-aula)

Parte VI

1. Substitua o comparador de 4 bits da parte III do trabalho prático 1 por uma instanciação do comparador parametrizável (*"EqCmpN"*) fornecido nos slides da aula teórica 3 (com *"size"* = 4).

2. Sintetize, implemente e teste o projeto no kit DE-115.

3. Repita os pontos anteriores para *"size"* = 8.

Parte VII

Crie uma cópia completa do projeto da parte IV (replicando toda a estrutura de diretórios do projeto) e altere a réplica de forma a que realize operações aritméticas sobre quantidades com sinal. Compile o projeto e teste-o no kit.

PDF criado em 26/02/2015 às 07:22:29