Laboratório de Sistemas Digitais

Aula Teórico-Prática 3

Ano Letivo 2022/23

Modelação em VHDL de circuitos aritméticos e comparadores

Introdução à parametrização de componentes

Universidade de Aveiro

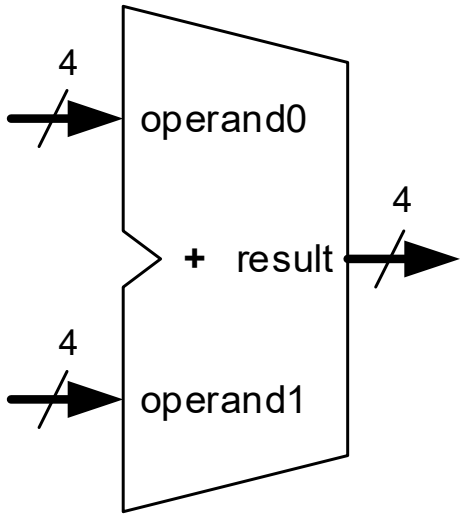
Conteúdo

- Circuitos aritméticos
  - Operadores aritméticos em VHDL
- Operações com quantidades *signed* e *unsigned*
- Modelação de comparadores em VHDL
- Introdução à parametrização de componentes em VHDL
  - Definição em VHDL
  - Instanciação em diagrama lógico e em VHDL

Exemplo de Circuito Aritmético – Somador Binário de 4 bits

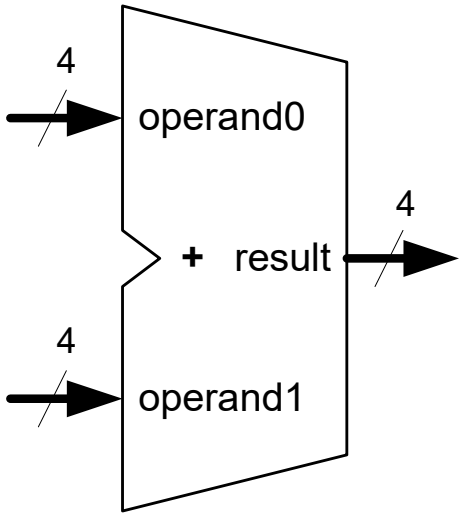
- *Entity*

```
entity Adder4 is
    port(operand0 : in  std_logic_vector(3 downto 0);
         operand1 : in  std_logic_vector(3 downto 0);
         result   : out std_logic_vector(3 downto 0));
end Adder4;
```



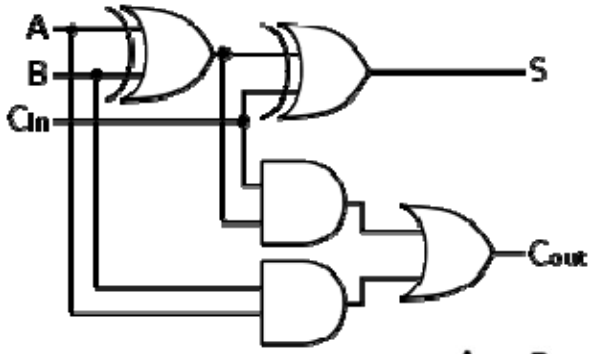
Somador Binário de 4 bits

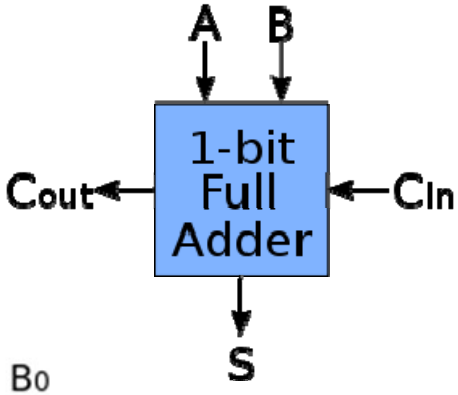
- *Architecture*
  - Diversas abordagens de modelação possíveis em VHDL
    - Instanciar e interligar as portas lógicas necessárias (estrutural)
    - Escrever as equações lógicas para cada saída
    - Escrever a expressão aritmética da saída

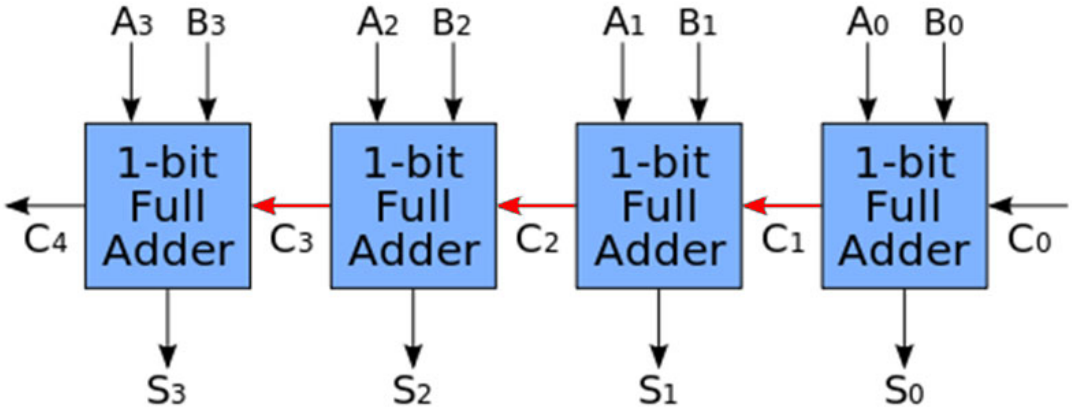


Vamos apresentar e analisar as diversas abordagens, com ênfase na última das três…

Modelação Estrutural do Somador







Abordagem da parte I do guião prático 3: modelar em VHDL um Full Adder de 1 bit (**`Entity FullAdder + Architecture`** contendo as equações lógicas); instanciar e interligar 4 **`FullAdder`** para construir um somador de 4 bits (**`Entity Adder4 + Architecture`**).

TPC: modelar o somador de 4 bits num único ficheiro (**`Entity Adder4 + Architecture`**) através das equações lógicas.

Modelação Comportamental do Somador (sem Carry In/Out)

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

entity Adder4 is
    port(operand0 : in  std_logic_vector(3 downto 0);
         operand1 : in  std_logic_vector(3 downto 0);
         result   : out std_logic_vector(3 downto 0));
end Adder4;

architecture Behavioral of Adder4 is
begin
    result <= std_logic_vector(unsigned(operand0) +
                               unsigned(operand1));
end Behavioral;
```

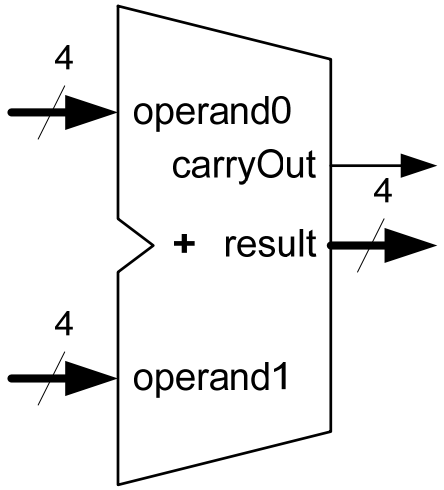
Recomendação: usar sempre **`std_logic(_vector)`** nos portos da entidade

Para que serve a biblioteca **`IEEE.NUMERIC_STD`**?

Conversão entre tipos: **`std_logic_vector(…) e unsigned(…)`**

Adição da Saída Carry Out

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

entity Adder4 is
    port(operand0 : in  std_logic_vector(3 downto 0);
         operand1 : in  std_logic_vector(3 downto 0);
         result   : out std_logic_vector(3 downto 0);
         carryOut : out std_logic);
end Adder4;

architecture Behavioral of Adder4 is

    signal s_operand0, s_operand1, s_result : unsigned(4 downto 0);

begin
    s_operand0 <= '0' & unsigned(operand0);
    s_operand1 <= '0' & unsigned(operand1);
    s_result   <= s_operand0 + s_operand1;
    result     <= std_logic_vector(s_result(3 downto 0));
    carryOut   <= std_logic(s_result(4)); -- std_logic(...) opcional
end Behavioral;
```



Operador "&" – concatenação / justaposição

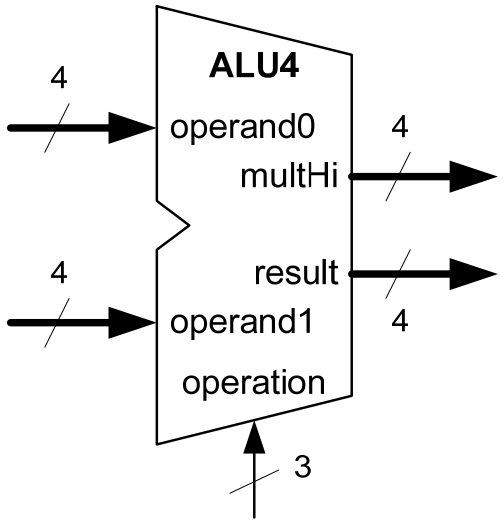
Definição formal dos tipos **unsigned** e **signed** de VHDL em **IEEE.NUMERIC_STD**:

```
type unsigned is array (natural range <> ) of std_logic;
type signed is array (natural range <> ) of std_logic;
```

TPC: O que seria necessário fazer para incluir também um "carryIn"?

Outros Operadores Aritméticos e Lógicos (ALU de 4 bits)

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

entity ALU4 is
    port(operation: in  std_logic_vector(2 downto 0);
         operand0 : in  std_logic_vector(3 downto 0);
         operand1 : in  std_logic_vector(3 downto 0);
         result   : out std_logic_vector(3 downto 0);
         multHi   : out std_logic_vector(3 downto 0));
end ALU4;
```



Operation	
000	+
001	-
010	*
011	/
100	rem
101	and
110	or
111	xor

A saída **`result`** disponibiliza o resultado da operação realizada.

No caso das multiplicações, os 4 bits mais significativos do resultado são disponibilizados em **`multHi`**.

Arquitetura da ALU de 4 bits

```
architecture Behavioral of ALU4 is
  signal s_multRes : std_logic_vector(7 downto 0);
begin
  s_multRes <= std_logic_vector(unsigned(operand0) *
                                unsigned(operand1));
  process(operation, operand0, operand1, s_multRes)
  begin
    case operation is
        when "000" =>
            result <= std_logic_vector(unsigned(operand0) + unsigned(operand1));
        when "001" =>
            result <= std_logic_vector(unsigned(operand0) - unsigned(operand1));
        when "010" =>
            result <= s_multRes(3 downto 0) ;
        when "011" =>
            result <= std_logic_vector(unsigned(operand0) / unsigned(operand1));
        when "100" =>
            result <= std_logic_vector(unsigned(operand0) rem unsigned(operand1));
        when "101" =>
            result <= operand0 and operand1;
        when "110" =>
            result <= operand0 or operand1;
        when others =>
            result <= operand0 xor operand1;
    end case;
  end process;
  multHi <= s_multRes(7 downto 4) when (operation = "010") else (others => '0');
end Behavioral;
```

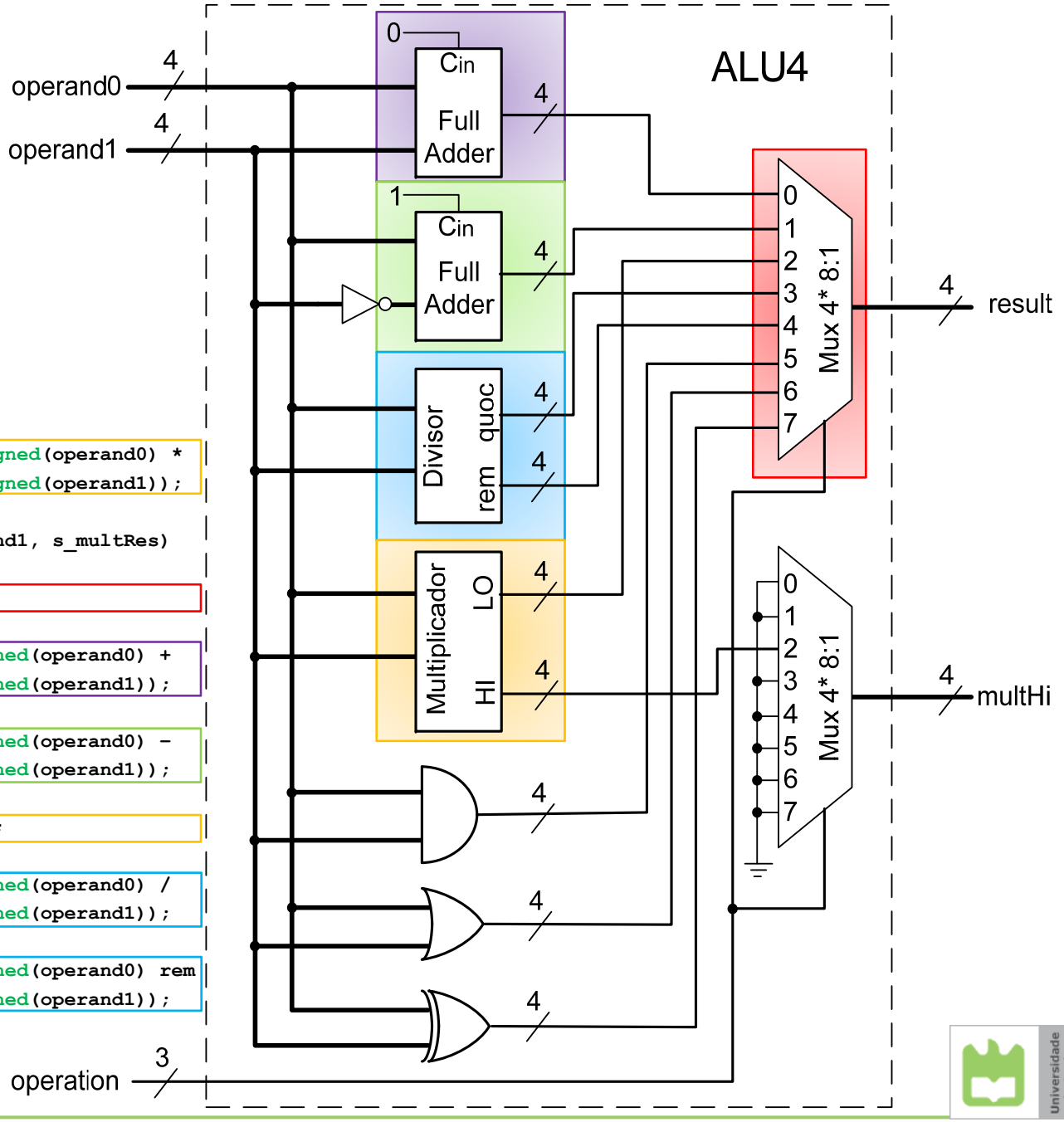
Utilização de sinais numa arquitetura para comunicação entre vários "blocos funcionais" (processos e atribuições concorrentes).

Assumindo:
Operand0 = "0100" e **Operand1 = "1110"**
Determine manualmente em decimal o resultado de cada operação. Simule e compare os resultados.

Nesta implementação perde-se o *carry/borrow out* do bit 3 da "+" e "-". Proponha uma solução.

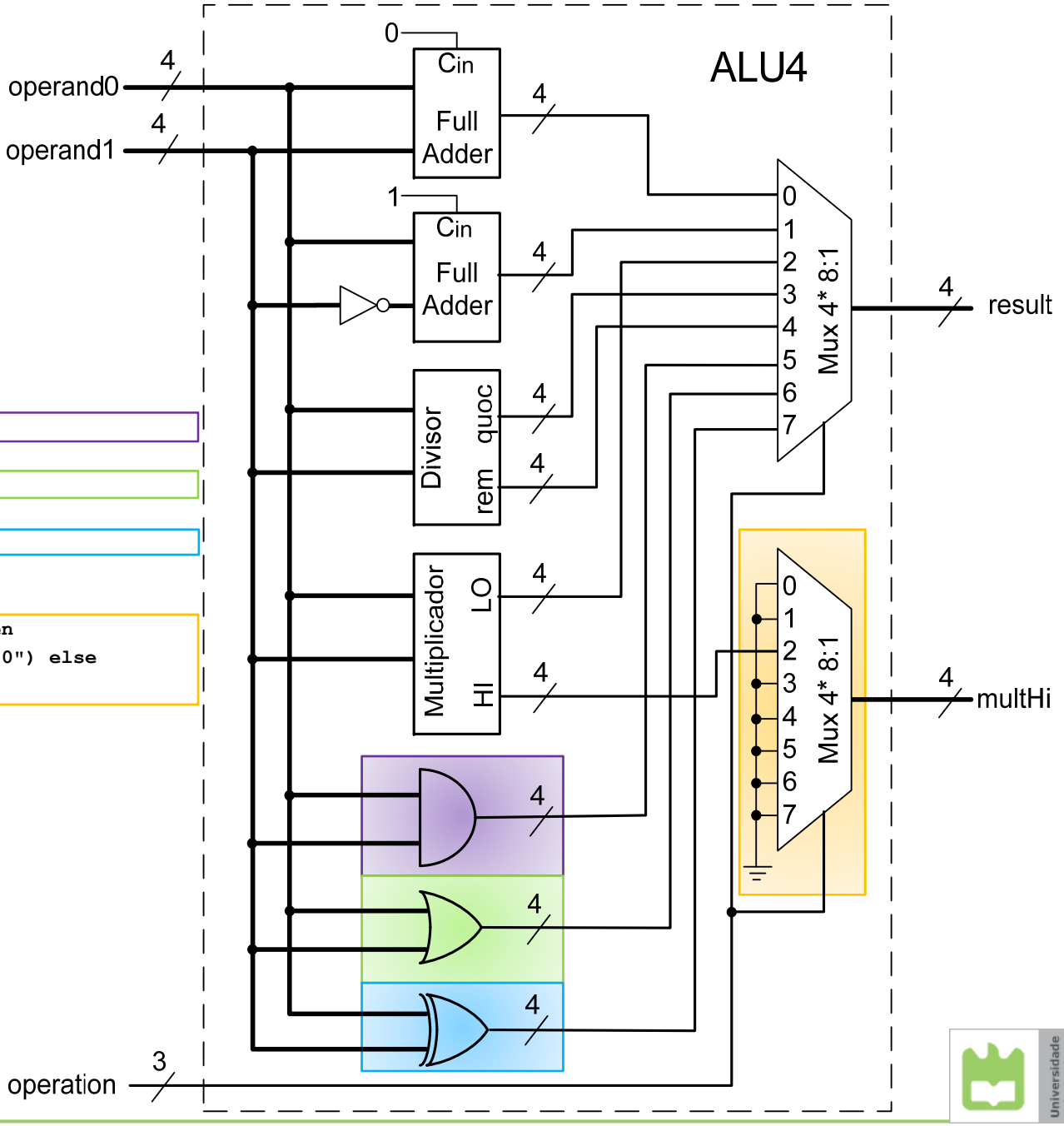
Implementação da ALU Gerada pelas Ferramentas de Compilação (síntese)

```
architecture Behavioral of ALU4 is
 ...
 s_multRes <= std_logic_vector(unsigned(operand0) *
                               unsigned(operand1));

 process(operation, operand0, operand1, s_multRes)
 begin
  case operation is
   when "000" =>
    result <= std_logic_vector(unsigned(operand0) +
                               unsigned(operand1));
   when "001" =>
    result <= std_logic_vector(unsigned(operand0) -
                               unsigned(operand1));
   when "010" =>
    result <= s_multRes(3 downto 0) ;
   when "011" =>
    result <= std_logic_vector(unsigned(operand0) /
                               unsigned(operand1));
   when "100" =>
    result <= std_logic_vector(unsigned(operand0) rem
                               unsigned(operand1));
 ...
```



Implementação da ALU Gerada pelas Ferramentas de Compilação (síntese)

```
  when "101" =>
    result <= operand0 and operand1;
  when "110" =>
    result <= operand0 or operand1;
  when others =>
    result <= operand0 xor operand1;
  end case;
 end process;
 multHi <= s_multRes(7 downto 4) when
                    (operation = "010") else
           (others => '0');
end Behavioral;
```



Operações com Quantidades Signed

```
architecture Behavioral of ALU4 is
  signal s_multRes : std_logic_vector(7 downto 0);
begin
  s_multRes <= std_logic_vector(signed(operand0) *
                                signed(operand1));
  process(operation, operand0, operand1, s_multRes)
  begin
    case operation is
        when "000" =>
            result <= std_logic_vector(signed(operand0) + signed(operand1));
        when "001" =>
            result <= std_logic_vector(signed(operand0) - signed(operand1));
        when "010" =>
            result <= s_multRes(3 downto 0);
        when "011" =>
            result <= std_logic_vector(signed(operand0) / signed(operand1));
        when "100" =>
            result <= std_logic_vector(signed(operand0) rem signed(operand1));
        when "101" =>
            result <= operand0 and operand1;
        when "110" =>
            result <= operand0 or operand1;
        when others =>
            result <= operand0 xor operand1;
    end case;
  end process;

  multHi    <= s_multRes(7 downto 4) when (operation = "010") else (others => '0');
end Behavioral;
```

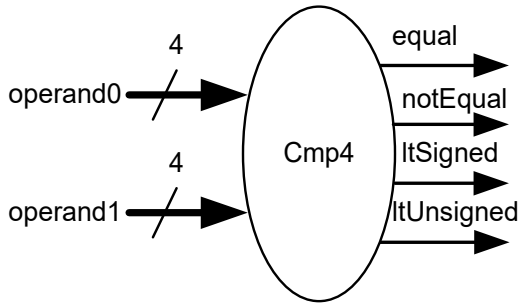
Adições e subtracões de quantidades com ou sem sinal representadas em complemento para 2 são realizadas da mesma forma. **O mesmo não acontece com as multiplicações e divisões!**

Assumindo: `Operand0 = "0100"` e `Operand1 = "1110"`
Determine manualmente em decimal o resultado de cada operação. Simule e compare os resultados.

Comparadores em VHDL

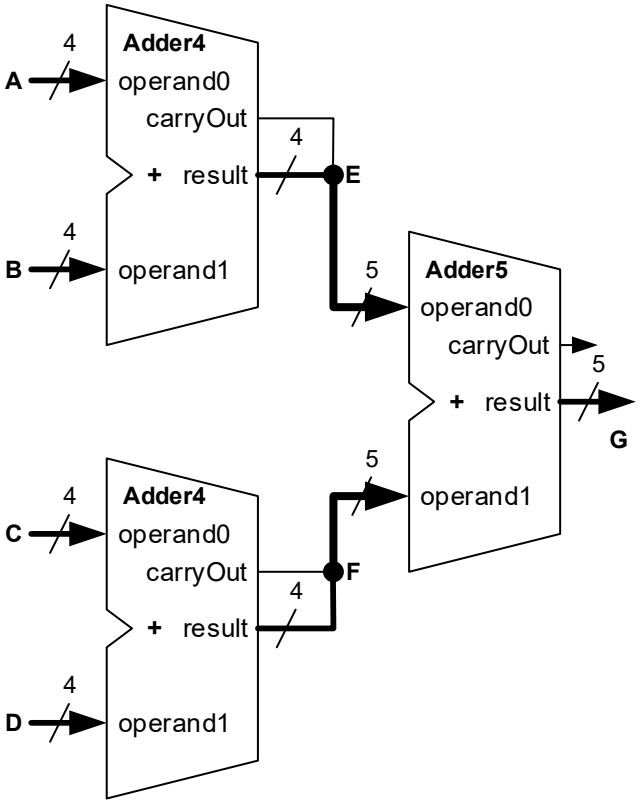
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Cmp4 is
    port(operand0   : in  std_logic_vector(3 downto 0);
         operand1   : in  std_logic_vector(3 downto 0);
         equal      : out std_logic;
         notEqual   : out std_logic;
         ltSigned   : out std_logic;
         ltUnsigned : out std_logic);
end Cmp4;

architecture Behavioral of Cmp4 is
begin
    equal      <= '1' when (operand0 = operand1) else
                  '0';
    notEqual   <= '1' when (operand0 /= operand1) else
                  '0';
    ltSigned   <= '1' when (signed(operand0) < signed(operand1)) else
                  '0';
    ltUnsigned <= '1' when (unsigned(operand0) < unsigned(operand1)) else
                  '0';
end Behavioral;
```

Comparador de 4 bits
=, ≠, < (com e sem sinal)



Assumindo:
Operand0 = "0100"
Operand1 = "1110"
Determine o resultado de cada comparação.

Exemplo de Motivação dos Componentes Parametrizáveis

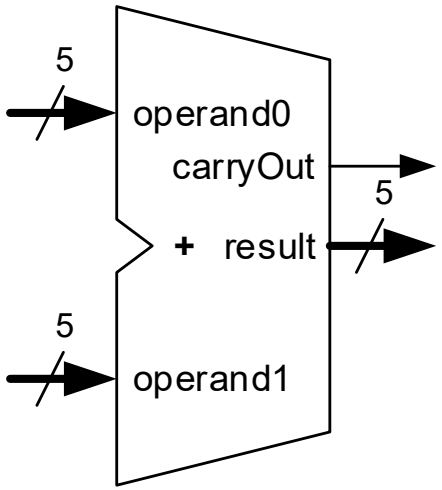


E = A + B (4 + 4 bits = 5 bits)
F = C + D (4 + 4 bits = 5 bits)
G = E + F (5 + 5 bits = 6 bits)

- Passos de modelação com componentes convencionais (**sem** utilizar componentes parametrizáveis):
  - Modelar somador de 4 bits (Adder4.vhd)
  - Modelar somador de 5 bits (Adder5.vhd)
  - Instanciar 2 somadores de 4 bits, 1 somador de 5 bits e interligá-los (e.g. TripleAdder.vhd)
- E se no mesmo ou noutros projetos fossem utilizados somadores com outras dimensões?
  - Teríamos de possuir um módulo para cada dimensão do somador?

Somador de 5 bits com Carry Out

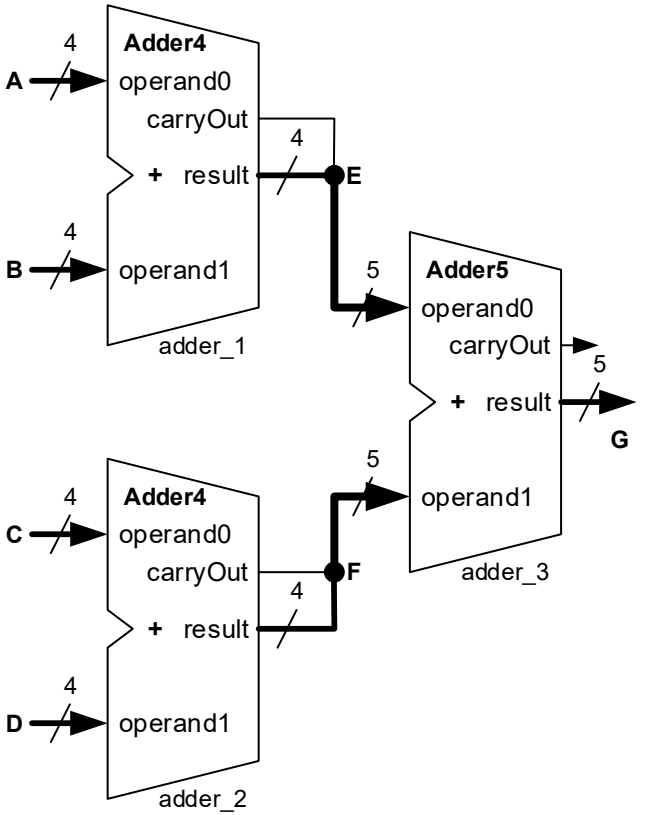
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Adder5 is
    port(operand0 : in  std_logic_vector(4 downto 0);
         operand1 : in  std_logic_vector(4 downto 0);
         result   : out std_logic_vector(4 downto 0);
         carryOut : out std_logic);
end Adder5;

architecture Behavioral of Adder5 is

    signal s_operand0, s_operand1, s_result : unsigned(5 downto 0);

begin
    s_operand0 <= '0' & unsigned(operand0);
    s_operand1 <= '0' & unsigned(operand1);
    s_result   <= s_operand0 + s_operand1;
    result     <= std_logic_vector(s_result(4 downto 0));
    carryOut   <= std_logic(s_result(5));
end Behavioral;
```



Modelo semelhante ao **Adder4** diferindo apenas na dimensão dos vetores!

Instanciação e Ligação dos Somadores Convencionais

```
...
 adder_1: entity WORK.Adder4(Behavioral)
          port map(operand0 => A,
                   operand1 => B,
                   result   => E(3 downto 0),
                   carryOut => E(4));

 adder_2: entity WORK.Adder4(Behavioral)
          port map(operand0 => C,
                   operand1 => D,
                   result   => F(3 downto 0),
                   carryOut => F(4));

 adder_3: entity WORK.Adder5(Behavioral)
          port map(operand0 => E,
                   operand1 => F,
                   result   => G,
                   carryOut => open);
...
```



A, B, C, D – sinais ou portos de entrada (4 bits)
E, F – sinais (5 bits)
G – sinal ou porto de saída (5 bits)

Componentes Parametrizáveis (em VHDL)

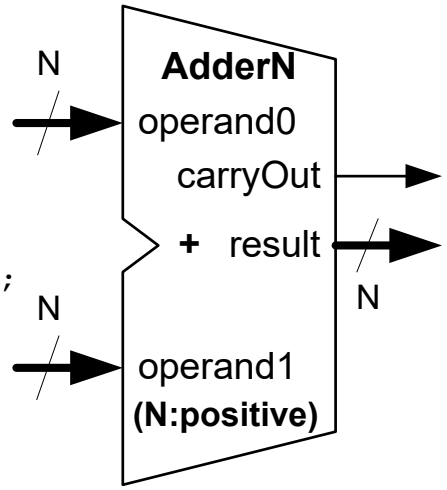
- O que são?
  - Componentes em que algumas das suas caraterísticas podem ser especificadas (especializadas) aquando da sua instanciação (e.g. número de bits de um somador, multiplexador, etc.)
- Para que servem?
  - Evitar bibliotecas "enormes" com todas as especializações e variantes possíveis de todos os componentes standard (e.g. somadores de 1, 2, 3, …, 30, 31, 32, … bits)
- Como fazer?
  - Criação do componente (em VHDL) – descrever o comportamento e estrutura genérica do componente baseada em **generic constants** (ou parâmetros) definidos na interface do componente (entity)
  - Utilização do componente
    - Em VHDL - instanciar o componente, ligando os portos e atribuindo valores concretos às generic constants
    - Em diagramas lógicos – definir um símbolo, instanciar o componente e definir valores concretos para os parâmetros

Somador de "N" bits com Carry Out

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

entity AdderN is
    generic(N      : positive := 4);
    port(operand0 : in  std_logic_vector(N - 1 downto 0);
         operand1 : in  std_logic_vector(N - 1 downto 0);
         result   : out std_logic_vector(N - 1 downto 0);
         carryOut : out std_logic);
end AdderN;

architecture Behavioral of AdderN is

    signal s_operand0, s_operand1, s_result : unsigned(N downto 0);

begin
    s_operand0 <= '0' & unsigned(operand0);
    s_operand1 <= '0' & unsigned(operand1);
    s_result   <= s_operand0 + s_operand1;
    result     <= std_logic_vector(s_result(N - 1 downto 0));
    carryOut   <= std_logic(s_result(N));
end Behavioral;
```

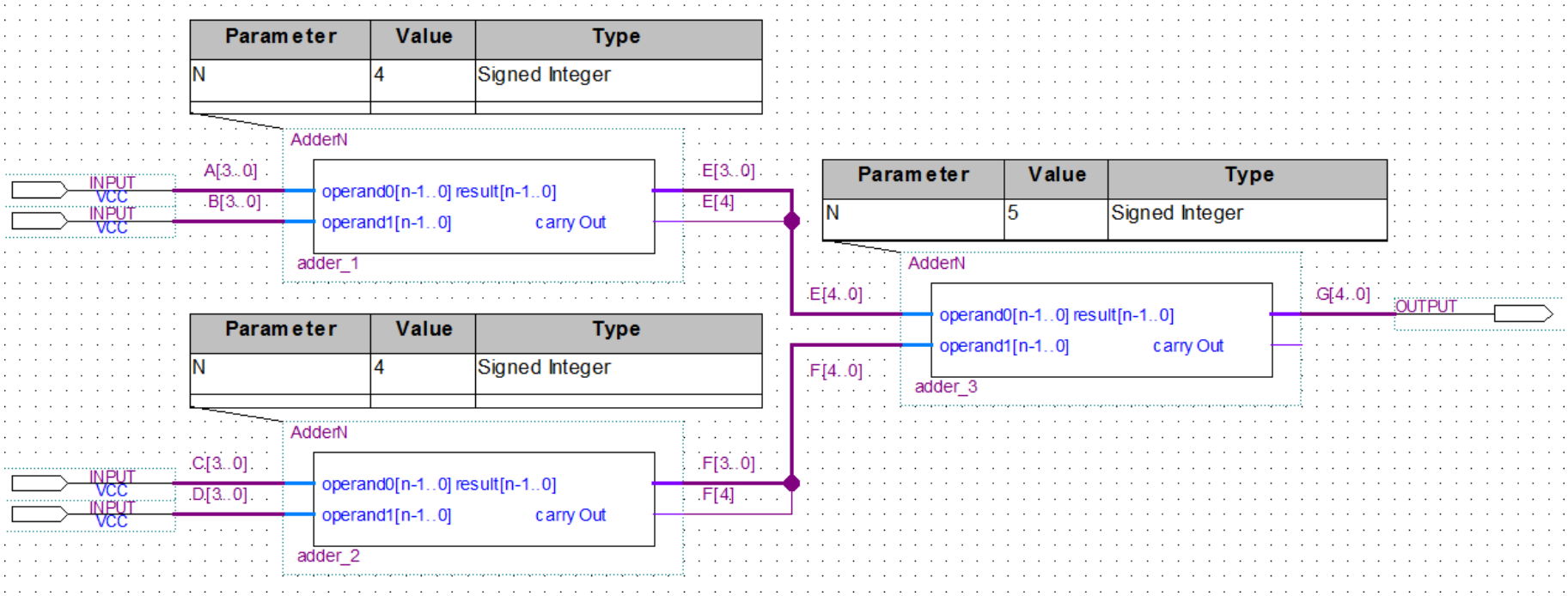
positive = inteiro positivo

Valor por omissão do *generic* - útil apenas para facilitar a simulação



Implementação em função de "N", com "N" **definido em *compile time***

Instanciação e Ligação dos Somadores Parametrizáveis (em diagrama lógico)



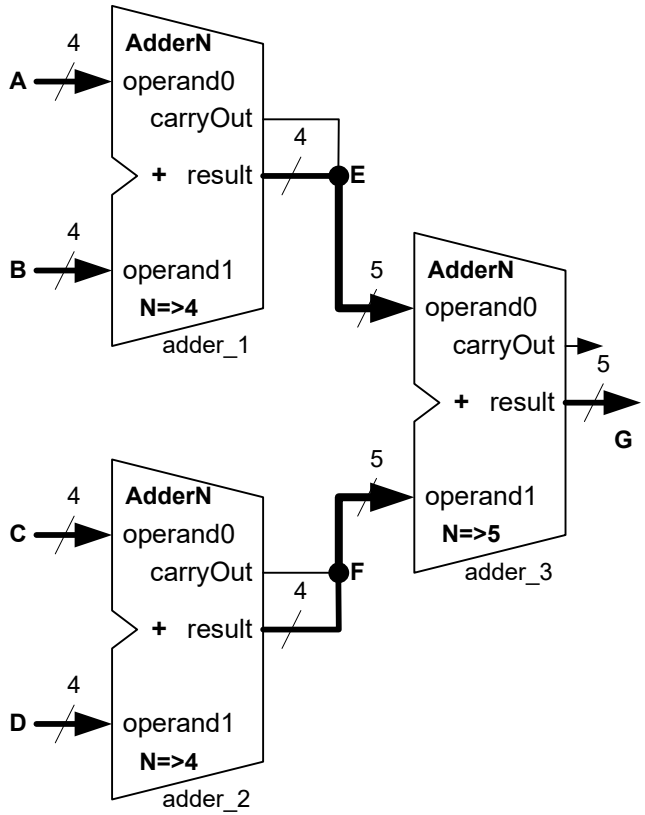
A, B, C, D – sinais ou portos de entrada (4 bits)
E, F – sinais (5 bits)
G – sinal ou porto de saída (5 bits)

Instanciação e Ligação dos Somadores Parametrizáveis (em VHDL)

```
...
adder_1: entity WORK.AdderN(Behavioral)
         generic map(N      => 4)
         port map(operand0 => A,
                  operand1 => B,
                  result   => E(3 downto 0),
                  carryOut => E(4));

adder_2: entity WORK.AdderN(Behavioral)
         generic map(N      => 4)
         port map(operand0 => C,
                  operand1 => D,
                  result   => F(3 downto 0),
                  carryOut => F(4));

adder_3: entity WORK.AdderN(Behavioral)
         generic map(N      => 5)
         port map(operand0 => E,
                  operand1 => F,
                  result   => G,
                  carryOut => open);
...
```

Atribuição "obrigatória" de um valor concreto ao *generic "N"*

Portos do somador

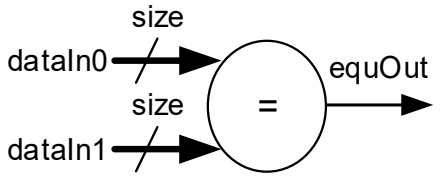


A, B, C, D – sinais ou portos de entrada (4 bits)
E, F – sinais (5 bits)
G – sinal ou porto de saída (5 bits)

Comparador Parametrizável de N bits

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

entity EqCmpN is
  generic(size : positive := 8);
  port(dataIn0 : in  std_logic_vector((size - 1) downto 0);
       dataIn1 : in  std_logic_vector((size - 1) downto 0);
       equOut  : out std_logic);
end EqCmpN;

architecture Behavioral of EqCmpN is
begin
  equOut <= '1' when (dataIn0 = dataIn1) else
            '0';
end Behavioral;
```

Valor por omissão do *generic*

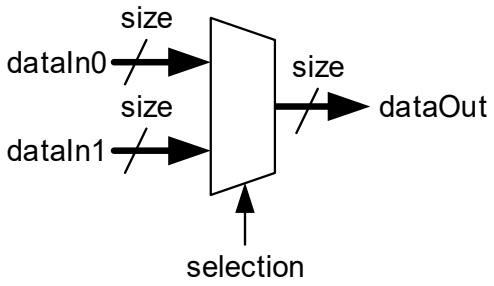


TPC: Substitua o comparador fixo de 4 bits do trabalho prático 1, por uma instanciação deste comparador parametrizável (com `size = 4`). Implemente e teste no kit DE2-115.

Mux 2→1 Parametrizável de N bits

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

entity Mux2N is
  generic(size     : positive := 8);
  port(selection   : in  std_logic;
       dataIn0     : in  std_logic_vector((size - 1) downto 0);
       dataIn1     : in  std_logic_vector((size - 1) downto 0);
       dataOut     : out std_logic_vector((size - 1) downto 0));
end Mux2N;

architecture Behavioral of Mux2N is
begin
  dataOut <= dataIn1 when (selection = '1') else
             dataIn0;
end Behavioral;
```

Valor por omissão do *generic*



Comentários Finais

- No final desta aula e do trabalho prático 3 de LSD, deverá ser capaz de:
  - Descrever componentes com operações aritméticas simples (**+**, **–**, *****, **/**, **rem**)
  - Usar adequadamente as bibliotecas e os tipos **std_logic(_vector)**, **signed** e **unsigned** do VHDL e as respetivas funções de conversão entre tipos
  - Modelar comparadores (<, =, ≠, >; *signed* e unsigned)
  - Usar adequadamente sinais numa arquitetura
- … bom trabalho prático 3, disponível no site da UC ☺
  - elearning.ua.pt
- Os componentes parametrizáveis serão objeto de estudo no trabalho prático 5