

MC542

Organização de Computadores Teoria e Prática

2007

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Circuitos Lógicos

VHDL e Blocos Básicos

"DDCA - (Capítulo 4)

"

"FDL"

Título do Capítulo Abordado

Sumário

- **VHDL**
 - **Escopo**
 - » **Arquitetura**
 - » **Visibilidade**
 - **Componentes**
 - » **Declaração**
 - » **Instanciação**
 - » **Exemplos**
 - **Configuração (Simplificada)**
 - » **Exemplos**
 - **Package**
 - **Package Body**

Escopo

- **Áreas declarativas**

- Onde são definidos os sinais internos, variáveis, constantes, subprogramas, aliases
- Áreas declarativas existem para packages, entidades, arquiteturas, subprogramas, processos e blocos
- OBS.: A área não declarativa é chamada de área de comandos

Escopo

- Área declarativa em Arquitetura
 - Declarações no topo da arquitetura são “visíveis” em toda a arquitetura

```
ARCHITECTURE exemplo OF circuito IS
```

```
    CONSTANT cte : time := 10 ns;
```

```
    SIGNAL tmp : integer;
```

```
    SIGNAL cnt : bit;
```

```
BEGIN
```

```
    ....
```

Escopo

- Escopo de uma declaração vai do identificador até a declaração END da região em que foi declarado
- Limites:
 - Componente
 - » Entidade
 - Arquitetura
 - Bloco
 - Processo
 - Subprograma

Escopo

- Na linguagem VHDL é possível a utilização de identificadores homônimos com diferentes significados, dependendo do contexto onde é definido cada identificador ele pode assumir diferentes significados no nível lógico.
- Um sinal definido dentro da parte declarativa de um componente, entidade, arquitetura, bloco, processo ou subprograma tem o escopo controlado dentro deste contexto. Desta forma é possível a utilização de nomes idênticos para indicações de sinais distintos.
- Para a distinção de sinais homônimos, cada sinal definido em VHDL pode ser acessado por seu endereço completo, indicando biblioteca, package, componente, arquitetura, processo e nome do sinal na forma:

biblioteca.componente.arquitetura.processo.sinal

Visibilidade

- Estabelece o significado dos identificadores
- As declarações são visíveis somente no seu escopo
- Uso de endereços em identificadores
 - var1 := architecture2.cte;
 - var2 := process1.cte;

Componentes

- Descrito pelo par **entidade e arquitetura**.
 - OBS.: Um modelo VHDL é dito estrutural se faz uso de instanciação de componentes.
- Usado na Arquitetura
 - Declaração de componente
 - » define o tipo do módulo
 - Instaciação de componenente
 - » define seu uso em um projeto
- Instanciação condicional
- Modelamento Estrutural

Componentes

- Declaração de um componente

```
component identifier [is]
  [generic (generic_interface_list);]
  [port (port_interface_list);]
end component [identifier];
```

OBS.: Similar a ENTIDADE

Componentes

- Exemplo

component flip-flop is

generic (Tprop, Tsetup, Thold : delay);

port (clk: in bit; clr : in bit; d : in bit; q : out bit);

end component flip_flop;

Componentes

- Instanciação

instatiation_label:

component componente_name

[**generic map** (generic_association_list)]

port map (port_association_list);

Componentes - Declaração

- **Exemplo:**

```
entity reg4 is
```

```
    port ( clk, clr : in bit;  d : in bit_vector(0 to 3);
```

```
          q : out bit_vector(0 to 3) );
```

```
end entity reg4;
```

```
architecture struct of reg4 is
```

```
    component flipflop is
```

```
        generic (Tprop, Tsetup, Thold : delay_length );
```

```
        port (clk : in bit;  clr : in bit;  d : in bit;
```

```
              q : out bit );
```

```
    end component flipflop;
```

Componentes - Instanciação

begin

bit0 : component flipflop

generic map (Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns)

port map (clk => clk, clr => clr, d => d(0), q => q(0));

bit1 : component flipflop

generic map (Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns)

port map (clk => clk, clr => clr, d => d(1), q => q(1));

bit2 : component flipflop

generic map (Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns)

port map (clk => clk, clr => clr, d => d(2), q => q(2));

bit3 : component flipflop

generic map (Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns)

port map (clk => clk, clr => clr, d => d(3), q => q(3));

end architecture struct;

Configuração

```
configuration identifier of entity_name is
  for architecture_name
    {for component_specification
      binding_indication;
    end for;}
  end for;
end [ configuration] [ identifier];
```

Configuração

```
FOR nome_da_instancia|others|all: nome_componente --  
    component_specification  
    USE ENTITY especificação_da_entidade; -- binding_indication  
END FOR;
```

Exemplos:

```
FOR inst51: xor_gate  
    USE ENTITY lib_projeto.xor(arq_rtl);  
END FOR;
```

```
FOR bit0, bit1: flipflop  
    use entity work.edge_triggered_Dff(basic);  
end for;
```

```
FOR others: flipflop  
    use entity work.edge_triggered_Dff(rtl);  
end for;
```


Exemplo

Architecture rtl of top is

Component and2

port(a, b: in std_logic;
c: out std_logic);

End component;

Component latchD

port(d, clk : in std_ulogic;
q, notq : out std_logic);

End component;

For all : and2 use entity work.and2(rtl);

For all : latchD use entity
work.latchD(rtl);

signal Q, NOTQ : std_ulogic := '1';

Begin

inst_latchD: latchD
port map(d1,clk,Q,NOTQ);

inst_and2_a: and2
port map(d1,Q,S1);

inst_and2_b: and2
port map(d2,NOTQ,S3);

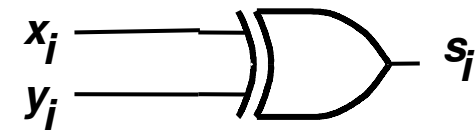
End rtl;

OBS.: d1, d2 e clk são sinais
de entrada e S1, S2 e S3
são sinais de saída

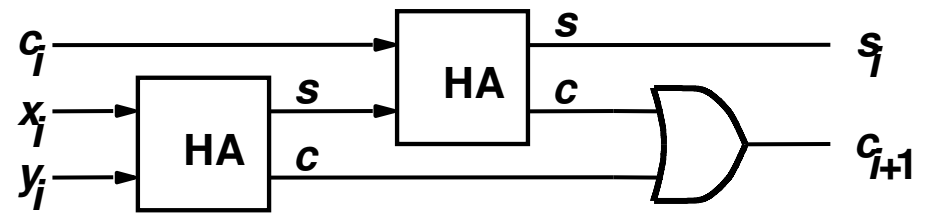
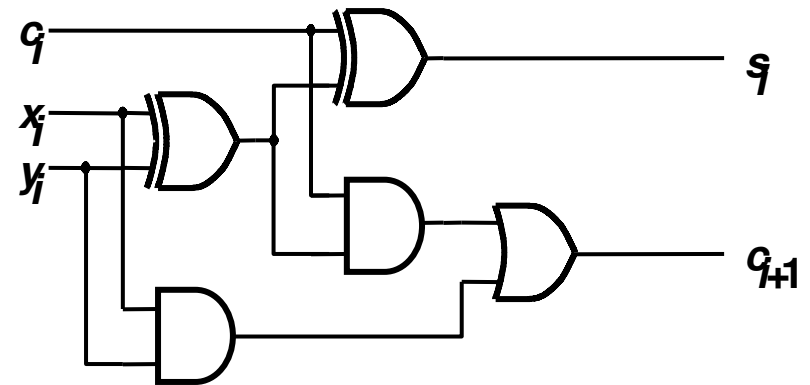
Adição de Números sem Sinal

$$\begin{array}{r}
 0101 \\
 + 0011 \\
 \hline
 0110
 \end{array}$$

$$\begin{array}{r}
 0x \\
 + 0y \\
 \hline
 cs
 \end{array}$$



		Carry	Soma
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Adição de Números sem Sinal com Múltiplos Bits

$$\begin{array}{r} X = x_4 x_3 x_2 x_1 x_0 \quad 01111 \quad (15)_{10} \\ + Y = y_4 y_3 y_2 y_1 y_0 \quad 01010 \quad (10)_{10} \\ \hline \quad 1110 \quad \leftarrow \text{Generated carries} \\ \hline S = s_4 s_3 s_2 s_1 s_0 \quad 11001 \quad (25)_{10} \end{array}$$

Adição de Números sem Sinal com Múltiplos Bits

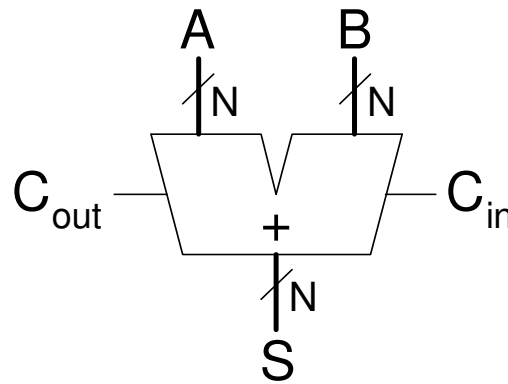
- Várias formas de propagação do carry:

Ripple-carry adders (slow)

Carry-lookahead adders (fast)

Prefix adders (faster)

OBS.: Carry-lookahead e prefix adders são rápidos para soma com vários bits porém utilizam mais hardware.

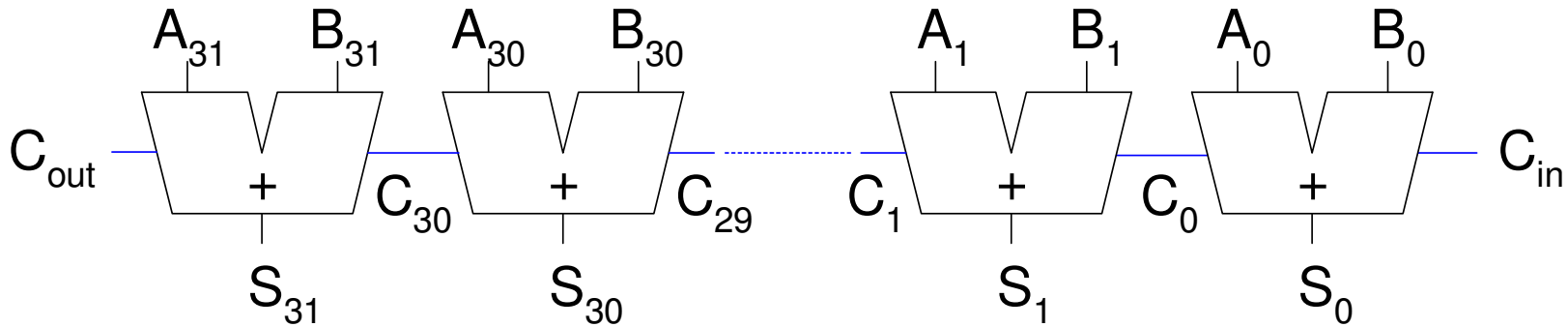


Somador Ripple Carry

- Atraso para um somador de n bits:

$$t_{\text{ripple}} = Nt_{FA}$$

Onde t_{FA} é o atraso de um full adder



Carry-Lookahead

$$C_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$C_{i+1} = x_i y_i + (x_i + y_i) c_i \implies C_{i+1} = g_i + p_i c_i$$

$$C_{i+1} = g_i + p_i (g_{i-1} + p_{i-1} c_{i-1})$$

$$C_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

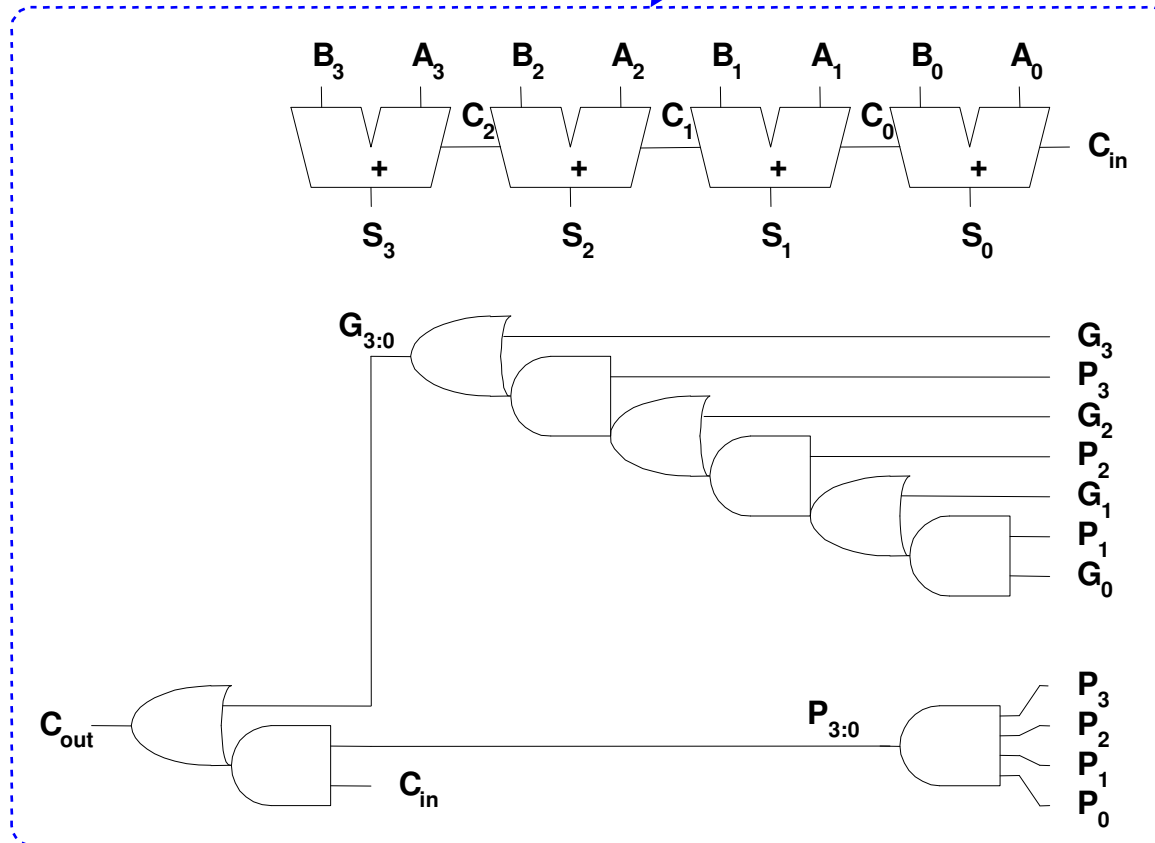
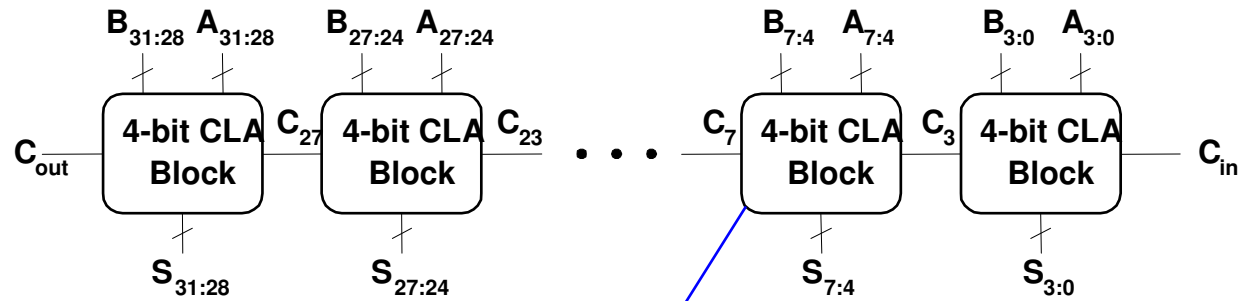
- Cálculo dos sinais Geração e Propagação para um Bloco de 4 bits

$$G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0))$$

$$P_{3:0} = P_3 P_2 P_1 P_0$$

$$C_i = G_{i:j} + P_{i:j} C_{i-1}$$

Carry Lookahead de 32 bits e 4 blocos



Delay no Somador Carry-Lookahead

- O delay de um somador N -bit carry-lookahead com blocos de k -bit é:

$$t_{CLA} = t_{pg} + t_{pg_block} + (N/k - 1)t_{AND_OR} + kt_{FA}$$

onde

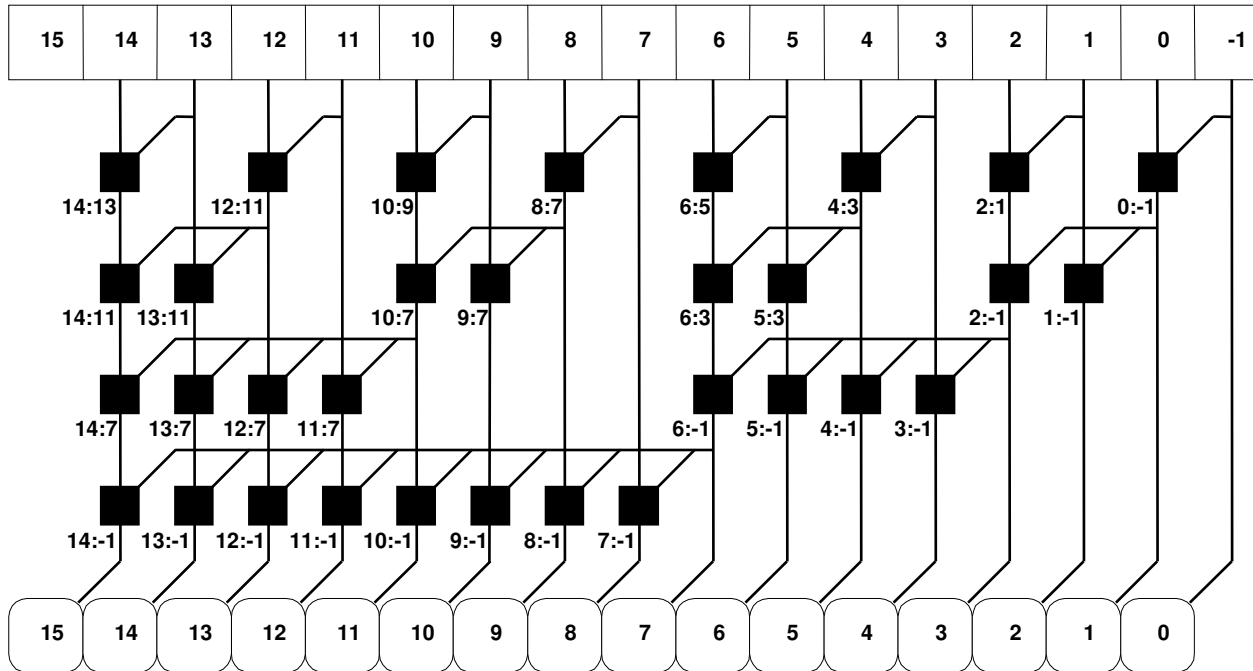
- t_{pg} é o delay de um gate da geração e propagação
 - t_{pg_block} é o delay dos gates da geração e propagação do bloco
 - t_{AND_OR} é o delay de C_{in} à porta AND/OR final de C_{out} no bloco CLA de k -bit
- O delay de um somador carry-lookahead de N -bit é em geral muito mais rápido que um somador ripple-carry para $N > 16$

Prefix Adder

- Extende o cálculo de G e P para tornar o somador ainda mais rápido.
 - G e P para par de colunas
 - G e P para blocos de 4
 - G e P para blocos de 8
 - G e P para blocos de ...

$$S_i = (A_i \text{ xor } B_i) \text{ xor } C_{i-1}$$

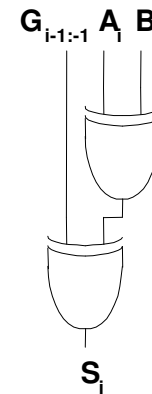
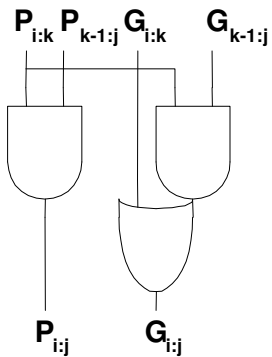
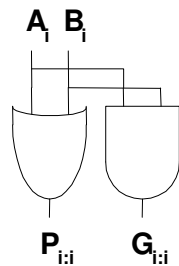
Prefix Adder



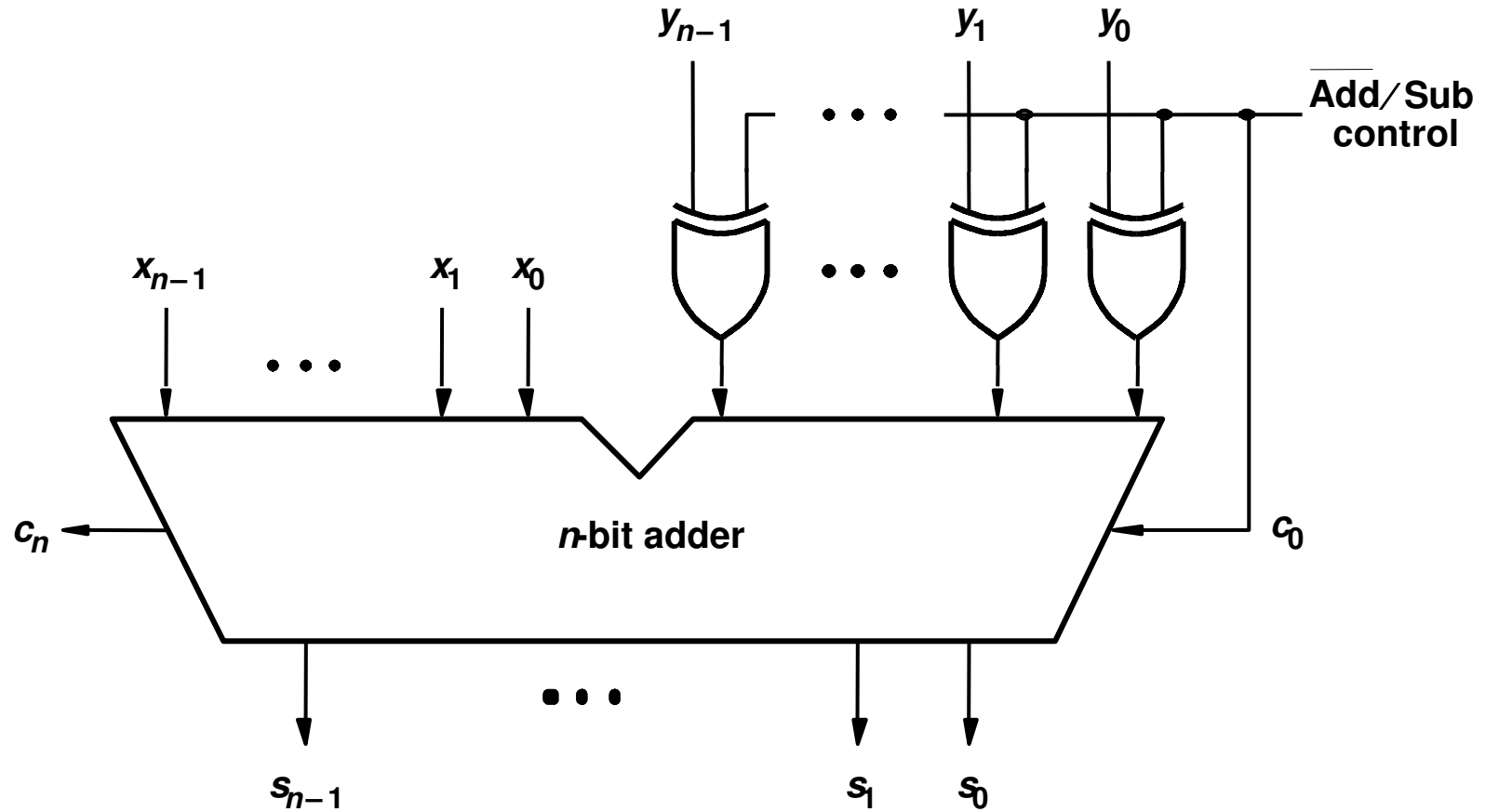
Legend



$i:j$



Unidade Somador-Subtrator



Package

- Declarações comuns a todo o projeto.
 - Exemplo: constantes, sinais, tipos de dados, subprogramas etc
- Package
 - Declarações de Tipos, Subtipos, constantes, interface de procedimentos e de funções
- Packge Body
 - Contém o corpo dos subprogramas definidos no Packge

Package

PACKAGE label IS

-- declarações;

END label;

Package BODY label IS

-- Corpo de subprogramas;

END label;

Package - Exemplo

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
  
ENTITY fulladd IS  
    PORT (Cin, x, y : IN      STD_LOGIC ;  
          s, Cout  : OUT   STD_LOGIC ) ;  
END fulladd ;  
  
ARCHITECTURE LogicFunc OF fulladd IS  
BEGIN  
    s <= x XOR y XOR Cin ;  
    Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ;  
END LogicFunc ;
```

Package - Exemplo

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY adder4 IS
    PORT (Cin           : IN    STD_LOGIC ;
          x3, x2, x1, x0 : IN    STD_LOGIC ;
          y3, y2, y1, y0 : IN    STD_LOGIC ;
          s3, s2, s1, s0 : OUT   STD_LOGIC ;
          Cout          : OUT   STD_LOGIC ) ;
END adder4 ;
```

```
ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1, c2, c3 : STD_LOGIC ;
```

-- Componente sem uso de Package

```
COMPONENT fulladd
    PORT (Cin, x, y : IN    STD_LOGIC ;
          s, Cout  : OUT   STD_LOGIC ) ;
END COMPONENT ;
```

Package - Exemplo

BEGIN

stage0: fulladd PORT MAP (Cin, x0, y0, s0, c1) ;

stage1: fulladd PORT MAP (c1, x1, y1, s1, c2) ;

stage2: fulladd PORT MAP (c2, x2, y2, s2, c3) ;

**stage3: fulladd PORT MAP (Cin => c3, Cout => Cout,
x => x3, y => y3, s => s3) ;**

END Structure ;

Package - Exemplo

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
  
-- Componente com uso de Package  
  
PACKAGE fulladd_package IS  
  COMPONENT fulladd  
    PORT ( Cin, x, y : IN  STD_LOGIC ;  
          s, Cout  : OUT  STD_LOGIC ) ;  
  END COMPONENT ;  
END fulladd_package ;
```

Package - Exemplo

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
USE work.fulladd_package.all ;  
  
ENTITY adder4 IS  
    PORT (Cin           : IN  STD_LOGIC ;  
          x3, x2, x1, x0 : IN  STD_LOGIC ;  
          y3, y2, y1, y0 : IN  STD_LOGIC ;  
          s3, s2, s1, s0 : OUT STD_LOGIC ;  
          Cout          : OUT STD_LOGIC ) ;  
END adder4 ;
```

Package - Exemplo

ARCHITECTURE Structure OF adder4 IS

SIGNAL c1, c2, c3 : STD_LOGIC ;

BEGIN

stage0: fulladd PORT MAP (Cin, x0, y0, s0, c1) ;

stage1: fulladd PORT MAP (c1, x1, y1, s1, c2) ;

stage2: fulladd PORT MAP (c2, x2, y2, s2, c3) ;

stage3: fulladd PORT MAP (Cin => c3, Cout => Cout,
x => x3, y => y3, s => s3) ;

END Structure ;

Exemplos

Usando std_logic_vector

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.fulladd_package.all ;

ENTITY adder4 IS
    PORT (Cin    : IN    STD_LOGIC ;
          X, Y   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          S      : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          Cout   : OUT   STD_LOGIC ) ;
END adder4 ;
```

Exemplos

Usando std_logic_vector

ARCHITECTURE Structure OF adder4 IS

SIGNAL C : STD_LOGIC_VECTOR(1 TO 3) ;

BEGIN

stage0: fulladd PORT MAP (Cin, X(0), Y(0), S(0), C(1)) ;

stage1: fulladd PORT MAP (C(1), X(1), Y(1), S(1), C(2)) ;

stage2: fulladd PORT MAP (C(2), X(2), Y(2), S(2), C(3)) ;

stage3: fulladd PORT MAP (C(3), X(3), Y(3), S(3), Cout) ;

END Structure ;

Exemplos

Usando ieee.std_logic_signed

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
    PORT (X, Y : IN    STD_LOGIC_VECTOR(15 DOWNT0 0) ;
          S   : OUT  STD_LOGIC_VECTOR(15 DOWNT0 0) ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN
    S <= X + Y ;
END Behavior
```

Não tem acesso ao Cout e Overflow

Exemplos

Usando ieee.std_logic_signed

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
    PORT (Cin      : IN  STD_LOGIC ;
          X, Y    : IN  STD_LOGIC_VECTOR(15 DOWNTO 0) ;
          S      : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ;
          Cout, Overflow : OUT STD_LOGIC ) ;
END adder16 ;
```

Exemplos

Usando `ieee.std_logic_signed`

```
ARCHITECTURE Behavior OF adder16 IS
  SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
  Sum <= ('0' & X) + Y + Cin ;
  S    <= Sum(15 DOWNTO 0) ;
  Cout <= Sum(16) ;
  Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```


Exemplos

Usando ieee.std_logic_arith

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
USE ieee.std_logic_arith.all ;  
  
ENTITY adder16 IS  
    PORT (Cin      : IN  STD_LOGIC ;  
          X, Y     : IN  SIGNED(15 DOWNT0 0) ;  
          S        : OUT SIGNED(15 DOWNT0 0) ;  
          Cout, Overflow : OUT STD_LOGIC ) ;  
END adder16 ;
```

Exemplos

Usando `ieee.std_logic_arith`

```
ARCHITECTURE Behavior OF adder16 IS
    SIGNAL Sum : SIGNED(16 DOWNT0 0) ;
BEGIN
    Sum <= ('0' & X) + Y + Cin ;
    S    <= Sum(15 DOWNT0 0) ;
    Cout <= Sum(16) ;
    Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```

Exemplos

```
ENTITY adder16 IS
  PORT (X, Y : IN  INTEGER RANGE -32768 TO 32767 ;
        S   : OUT INTEGER RANGE -32768 TO 32767 ) ;
END adder16 ;
```

```
ARCHITECTURE Behavior OF adder16 IS
BEGIN
  S <= X + Y ;
END Behavior ;
```

Exemplos

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
USE ieee.std_logic_unsigned.all ;
```

```
ENTITY BCD IS
```

```
    PORT (X, Y : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;  
          S    : OUT  STD_LOGIC_VECTOR(4 DOWNTO 0) ) ;
```

```
END BCD ;
```

```
ARCHITECTURE Behavior OF BCD IS
```

```
    SIGNAL Z : STD_LOGIC_VECTOR(4 DOWNTO 0) ;  
    SIGNAL Adjust : STD_LOGIC ;
```

```
BEGIN
```

```
    Z <= ('0' & X) + Y ;  
    Adjust <= '1' WHEN Z > 9 ELSE '0' ;  
    S <= Z WHEN (Adjust = '0') ELSE Z + 6 ;
```

```
END Behavior ;
```

Exemplos

