IC-UNICAMP

# MC 613

IC/Unicamp
2012s1
Prof Guido Araújo
Prof Mario Côrtes

# Circuitos Aritméticos

# Tópicos

- Representação sinal-magnitude
- Representação $K_1$ e $K_2$
- Somadores half-adder e full-adder
- Somador/subtrator
- Somador com overflow

# Representação de Números Negativos

IC-UNICAMP

| $b_3\ b_2\ b_1\ b_0$ | Sinal Magnitude | Complemento de 1 | Complemento de 2 |
|---|---|---|---|
| 0111 | +7 | 7 | +7 |
| 0110 | +6 | 6 | +6 |
| 0101 | +5 | 5 | +5 |
| 0100 | +4 | 4 | +4 |
| 0011 | +3 | 3 | +3 |
| 0010 | +2 | 2 | +2 |
| 0001 | +1 | 1 | +1 |
| 0000 | +0 | 0 | 0 |
| 1000 | -0 | -7 | -8 |
| 1001 | -1 | -6 | -7 |
| 1010 | -2 | -5 | -6 |
| 1011 | -3 | -4 | -5 |
| 1100 | -4 | -3 | -4 |
| 1101 | -5 | -2 | -3 |
| 1110 | -6 | -1 | -2 |
| 1111 | -7 | -0 | -1 |

Table 5.2 Interpretation of four-bit signed integers

# Representação Sinal Magnitude

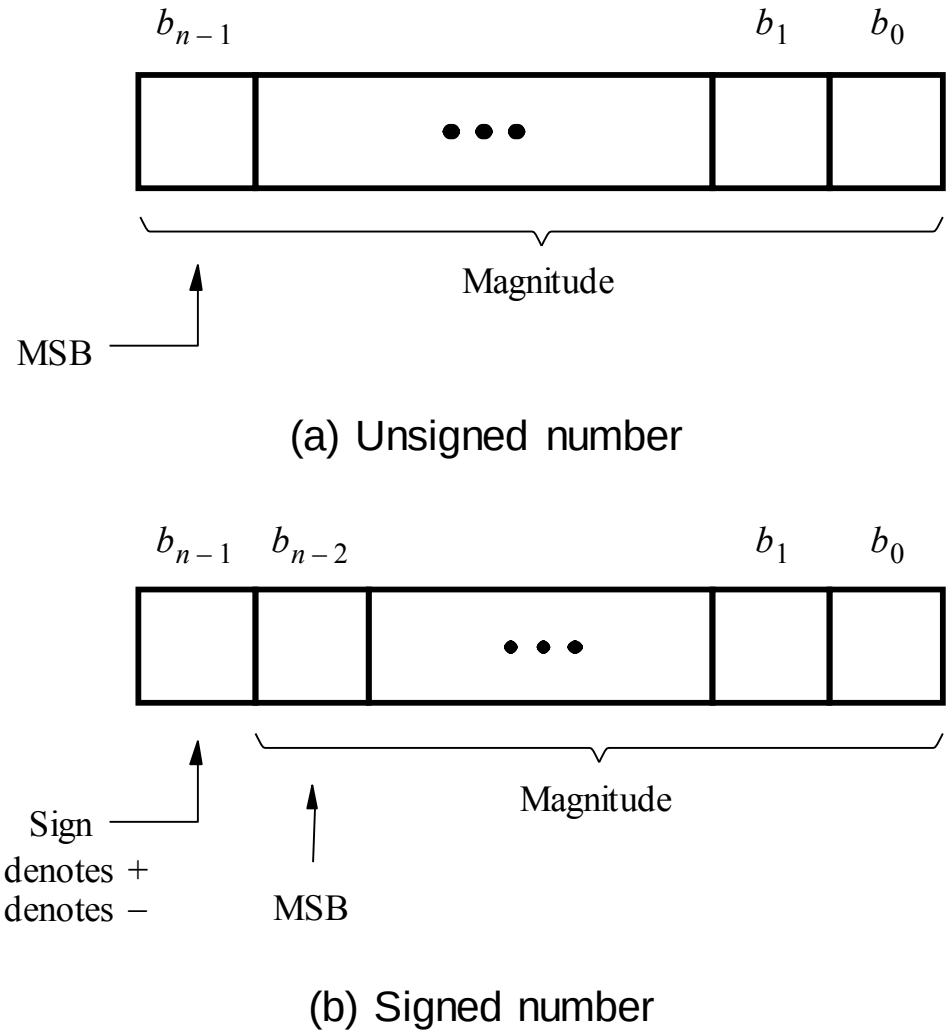



Figure 5.8 Formats for representation of integers

IC-UNICAMP

# Representação de Números Negativos

- Sinal e Magnitude
  - 1 bit de signal, N-1 bits de magnitude
  - O bit de sinal é o mais significativo (mais a esquerda)
    - Número negativo: 1
    - Número positivo: 0
  - Exemplo, representação de ± 5 com 4-bit:

$$-5 = 1101_2$$

$$+5 = 0101_2$$

  - Intervalo de um número *N*-bit sinal/magnitude:

$$[-(2^{N-1}-1), 2^{N-1}-1]$$

# Adição e Subtração Sinal e Magnitude

- Exemplo: -5 + 5

```
  1 1 0 1
+ 0 1 0 1
-------
1 0 0 1 0
```

- Duas representações para 0 (± 0):

```
1000
0000
```

IC-UNICAMP

# Representação de Números Negativos

## Complemento de 1 ($K_1$)

Em complemento de "Um" o número negativo $K_1$, com n-bits, é obtido subtraíndo seu positivo P de $2^n - 1$

$$K_1 = (2^n - 1) - P$$

Exemplo: se n = 4 então:

$$K_1 = (2^4 - 1) - P$$
$$K_1 = (16 - 1) - P$$
$$K_1 = (1111)_2 - P$$

P = 7 -> $K_1$ = ?

$$7 = (0111)_2$$
$$-7 = (1111)_2 - (0111)_2$$
$$-7 = (1000)_2$$

# Representação de Números Negativos

- Complemento de 1 ($K_1$)
  - Regra Prática

$$K_1 = (2^n - 1) - P$$

$$K_1 = 11\ldots11 - (p_{n-1} \ldots p_0)$$

$$K_1 = (\overline{p}_{n-1} \ldots \overline{p}_0)$$

# Representação de Números Negativos

## Complemento de 2 ($K_2$)

Em complemento de "Dois" o número negativo K, com n-bits, é obtido subtraíndo seu positivo P de $2^n$

$$K_2 = 2^n - P$$

Exemplo: se n = 4 então:

$K_2 = 2^4 - P$
$K_2 = 16 - P$
$K_2 = (10000)_2 - P$

P = 7 -> $K_2$= ?
$7 = (0111)_2$
$-7 = (10000)_2 - (0111)_2$
$-7 = (1001)_2$

# Representação de Números Negativos

- Complemento de 2 ($K_2$)
  - Regra Prática

$$K_2 = 2^n - P \longrightarrow K_2 = (2^n - 1) + 1 - P$$

$$K_2 = (2^n - 1) - P + 1$$

$$K_2 = 11\ldots11 - (p_{n-1} \ldots p_0) + 1$$

$$K_2 = \overline{(p_{n-1} \ldots p_0)} + 1 = K_1(P) + 1$$

IC-UNICAMP

# Representação de Números Negativos

- Complemento de 2 ($K_2$)
  - Maior número positivo de 4-bit: $0111_2$ ($7_{10}$)
  - Maior número negativo de 4-bit: $1000_2$ ($-2^3 = -8_{10}$)
  - O most significant bit também indica o sinal (1 = negativo, 0 = positivo)
  - Intervalo de um número de *N*-bit: $[-2^{N-1}, 2^{N-1}-1]$

# Adição em $K_2$

| | | | |
|---|---|---|---|
| (+ 5) | 0 1 0 1 | (–5) | 1 0 1 1 |
| + (+ 2) | + 0 0 1 0 | + (+ 2) | + 0 0 1 0 |
| (+ 7) | 0 1 1 1 | (–3) | 1 1 0 1 |

| | | | |
|---|---|---|---|
| (+ 5) | 0 1 0 1 | (–5) | 1 0 1 1 |
| + (–2) | + 1 1 1 0 | + (–2) | + 1 1 1 0 |
| (+ 3) | 1 0 0 1 1 | (–7) | 1 1 0 0 1 |
| | ignore | | ignore |

Figure 5.10 Examples of 2's complement addition

# Subtração em $K_2$

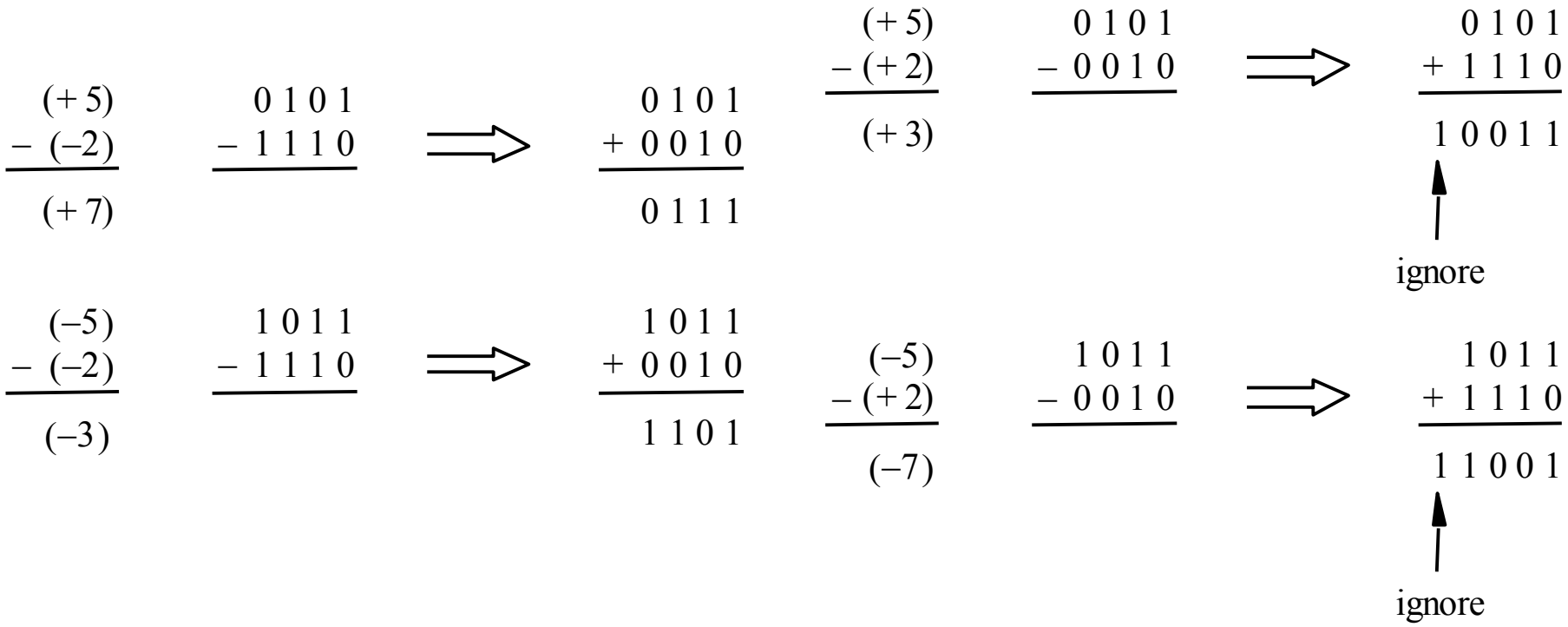



Figure 5.11 Examples of 2's complement subtraction

# Half-adder

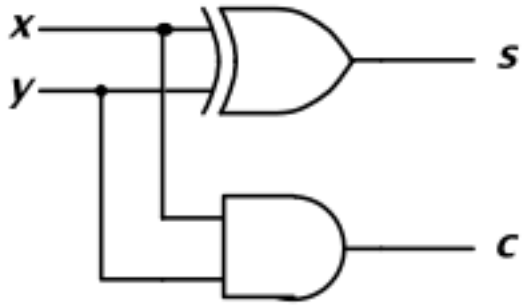
| | | | | |
|---|---|---|---|---|
| $x$ | 0 | 0 | 1 | 1 |
| $+y$ | $+0$ | $+1$ | $+0$ | $+1$ |
| $c\ s$ | 0 0 | 0 1 | 0 1 | 1 0 |

Carry Sum

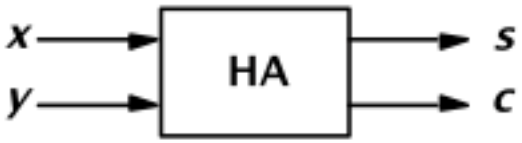
(a) The four possible cases

| $x$ | $y$ | Carry $c$ | Sum |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

(b) Truth table

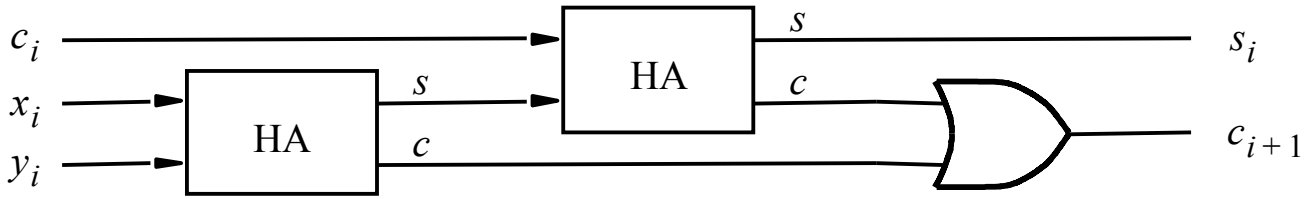



(c) Circuit

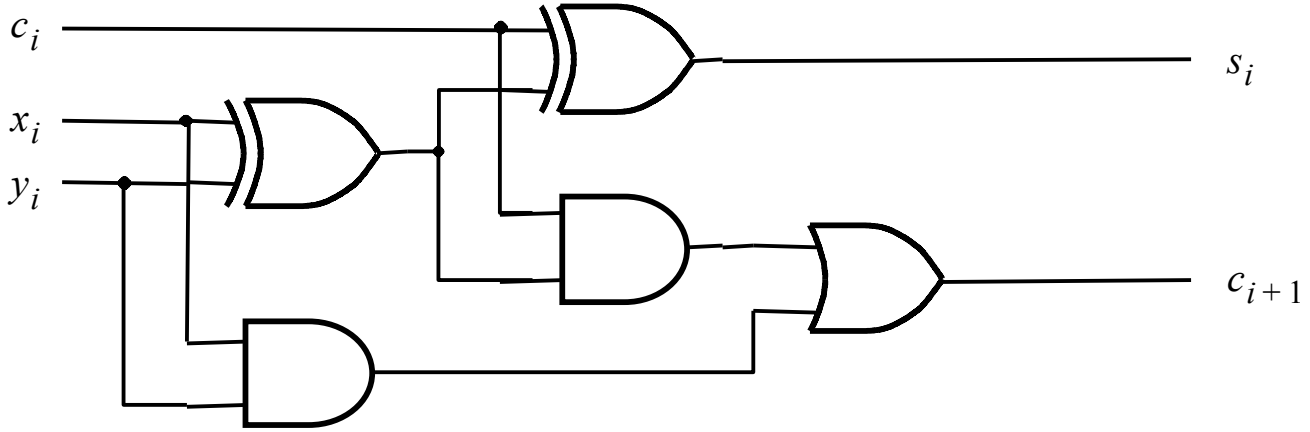



(d) Graphical symbol

# Somador com Half-adder




(a) Block diagram



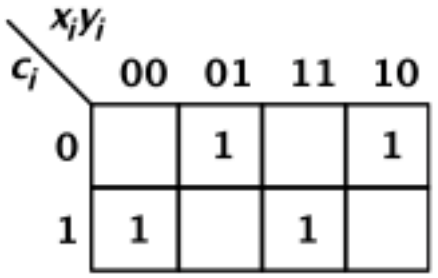

(b) Detailed diagram

Figure 5.5 A decomposed implementation of the full-adder circuit

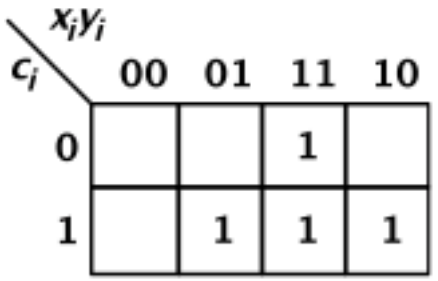
# Full-adder

IC-UNICAMP

| $c_i$ | $x_i$ | $y_i$ | $c_{i+1}$ | $s_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

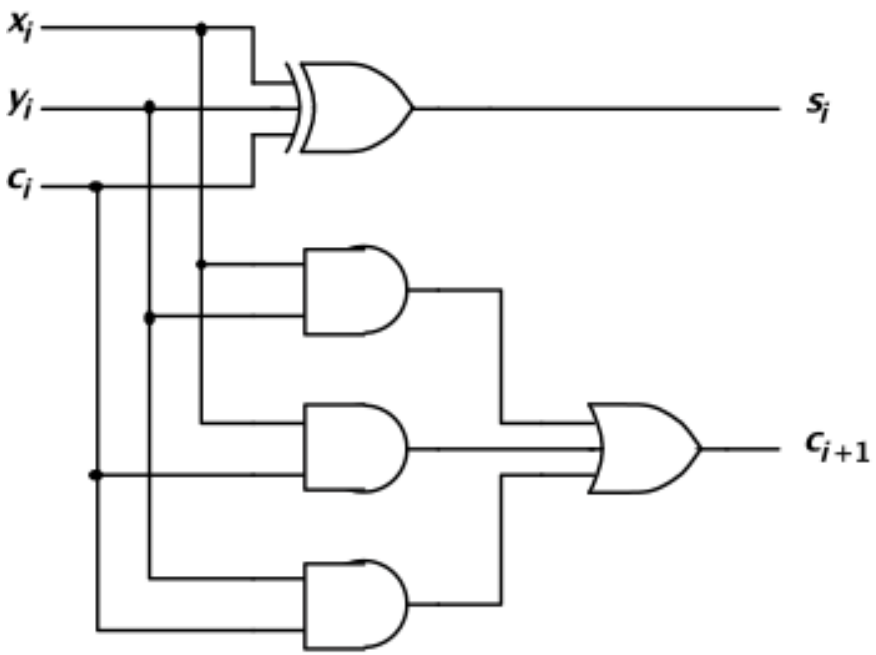



$s_{i+1} = x_i \text{ xor } y_i \text{ xor } c_i$




$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$

(b) Karnaugh maps



# Full-adder (VHDL)

```
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 ;
```

Figure 5.23 VHDL code for the full-adder

# Full-adder Package (VHDL)

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

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

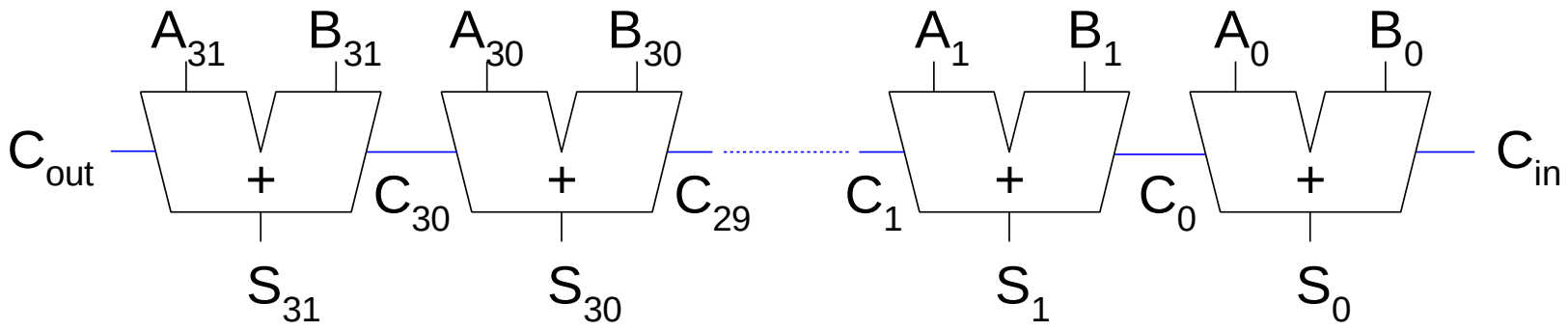
Figure 5.25 Declaration of a package

# Somador Ripple Carry

- Atraso para um somador de n bits:

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

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



IC-UNICAMP

# 4-bit Ripple Carry Adder (sinais)

```
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 ;

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 (
            x => x3, y => y3, Cin => c3, Cout => Cout, s => s3  ) ;
END Structure ;
```

Figure 5.26 Using a package for the four-bit adder

# 4-bit Ripple Carry Adder (vetores)

```
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 ;

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 ;
```

Figure 5.27 A four-bit adder defined using multibit signals

# Descrição Comportamental

```
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 DOWNTO 0) ;
           S     : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END adder16 ;

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

Figure 5.28 VHDL code for a 16-bit adder

# Somador/Subtrator

$$K_2 = (\overline{p_{n-1}} \ldots \overline{p_0}) + 1 = K_1(P) + 1$$

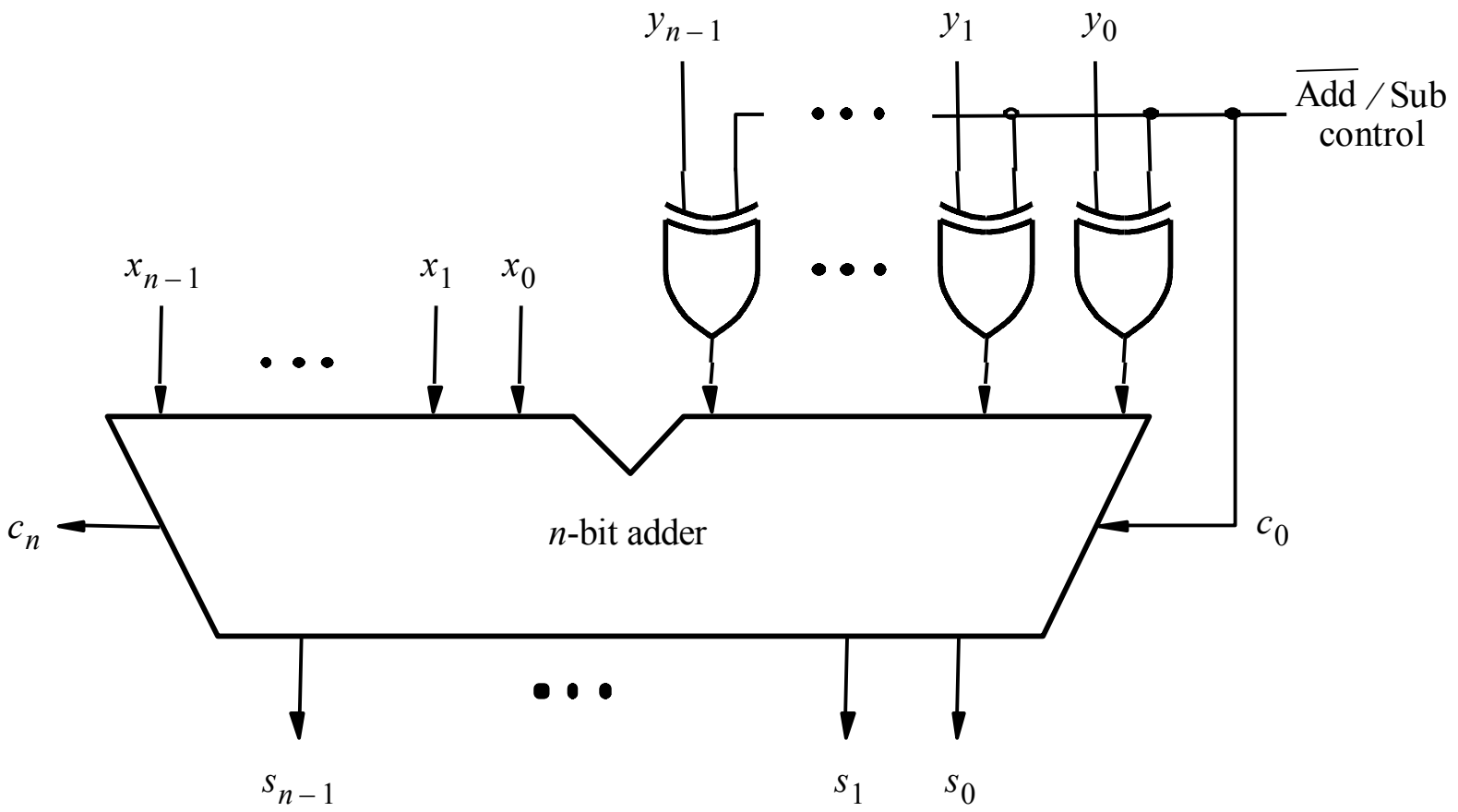


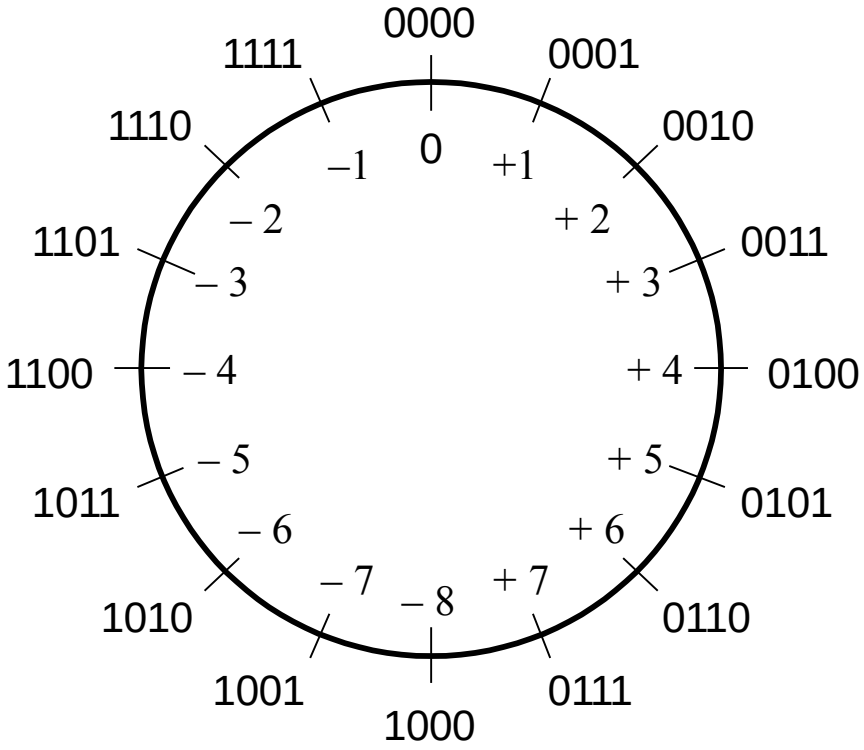

Figure 5.13 Adder/subtractor unit

# Overflow (Soma)

- A + B
  - sign(A) = sign(B): overflow possível
  - sign(A) ≠ sign(B): overflow impossivel

| | |
|---|---|
| (+ 7) | 0 1 1 1 |
| + (+ 2) | + 0 0 1 0 |
| (+ 9) | 1 0 0 1 |

| | |
|---|---|
| (–7) | 1 0 0 1 |
| + (+ 2) | + 0 0 1 0 |
| (–5) | 1 0 1 1 |

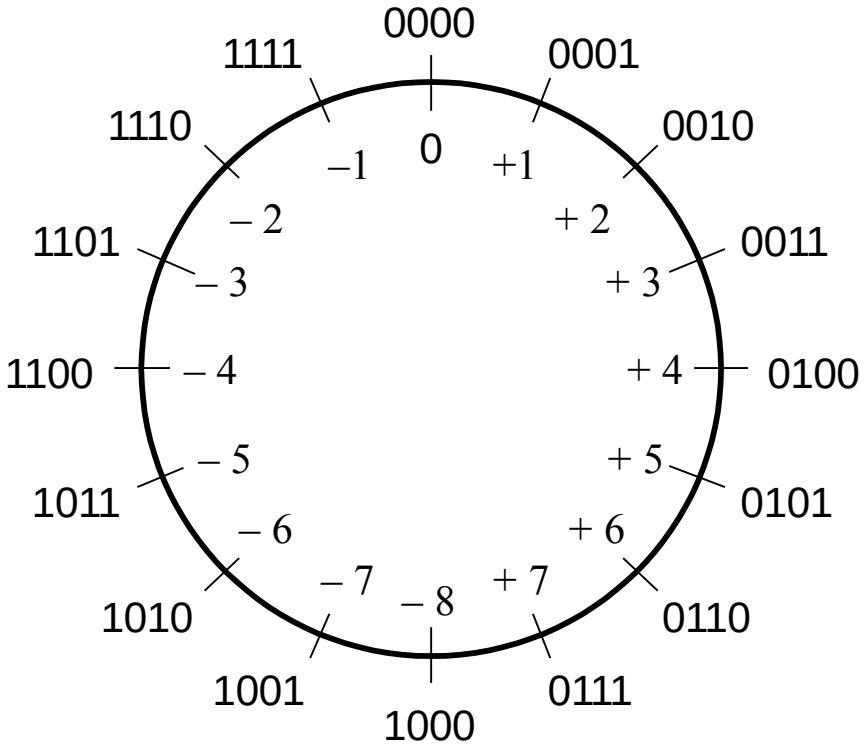


# Overflow (Subtração)

- A - B = A + (-B), reduz à soma
  - sign(A) = sign(B): overflow impossível
  - sign(A) ≠ sign(B): overflow possível

| | | |
|---|---|---|
| (+ 7) | (+ 7) | 0 1 1 1 |
| - (+2) | + (−2) | + 1 1 1 0 |
| (+ 5) | (+ 5) | 1 0 1 0 1 |

| | | |
|---|---|---|
| (−7) | (−7) | 1 0 0 1 |
| - ( +2) | + (− 2) | + 1 1 1 0 |
| (−5) | (−9) | 1 0 1 1 1 |



# Resumo Overflow

IC-UNICAMP

| A | B | S = A + B | OV |
|---|---|---|---|
| + | - | + | 0 |
| + | - | - | 0 |
| - | + | + | 0 |
| - | + | - | 0 |
| + | + | + | 0 |
| + | + | - | 1 |
| - | - | + | 1 |
| - | - | - | 0 |

$$\begin{array}{r} 1 \\ 0\ x.. \\ +\ 1\ x.. \\ \hline 1\ 0\ x.. \end{array} \qquad \begin{array}{r} 0 \\ 0\ x.. \\ +\ 1\ x.. \\ \hline 0\ 1\ x.. \end{array}$$

$$\begin{array}{r} 1 \\ 0\ x.. \\ +\ 0\ x.. \\ \hline 0\ 1\ x.. \end{array} \qquad \begin{array}{r} 0 \\ 0\ x.. \\ +\ 0\ x.. \\ \hline 0\ 0\ x.. \end{array}$$

$$\begin{array}{r} 0 \\ 1\ x.. \\ +\ 1\ x.. \\ \hline 1\ 0\ x.. \end{array} \qquad \begin{array}{r} 1 \\ 1\ x.. \\ +\ 1\ x.. \\ \hline 1\ 1\ x.. \end{array}$$

$V = C_n(S)$ xor $C_{n-1}(S)$

# Overflow em $K_2$

| | | | |
|---|---|---|---|
| $(+7)$ | $0\ 1\ 1\ 1$ | $(-7)$ | $1\ 0\ 0\ 1$ |
| $+\ (+2)$ | $+\ 0\ 0\ 1\ 0$ | $+\ (+2)$ | $+\ 0\ 0\ 1\ 0$ |
| $(+9)$ | $1\ 0\ 0\ 1$ | $(-5)$ | $1\ 0\ 1\ 1$ |
| | $c_4 = 0$ | | $c_4 = 0$ |
| | $c_3 = 1$ | | $c_3 = 0$ |
| $(+7)$ | $0\ 1\ 1\ 1$ | $(-7)$ | $1\ 0\ 0\ 1$ |
| $+\ (-2)$ | $+\ 1\ 1\ 1\ 0$ | $+\ (-2)$ | $+\ 1\ 1\ 1\ 0$ |
| $(+5)$ | $1\ 0\ 1\ 0\ 1$ | $(-9)$ | $1\ 0\ 1\ 1\ 1$ |
| | $c_4 = 1$ | | $c_4 = 1$ |
| | $c_3 = 1$ | | $c_3 = 0$ |

Figure 5.14 Examples of determination of overflow


# 4-bit Ripple Carry Adder (vetores) + overflow

```
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, Overflow     : OUT  STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL C : STD_LOGIC_VECTOR(1 TO 4) ;
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), C(4) ) ;
    Overflow <= C(3) XOR C(4);
    Cout <= C(4);
END Structure ;
```

IC-UNICAMP

# Descrição Comportamental
# Como incluir overflow?

```
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 DOWNTO 0) ;
           S     : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END adder16 ;

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

Figure 5.28 VHDL code for a 16-bit adder

# 16-bit Adder com Overflow

IC-UNICAMP

```
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 ;

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 ;
```

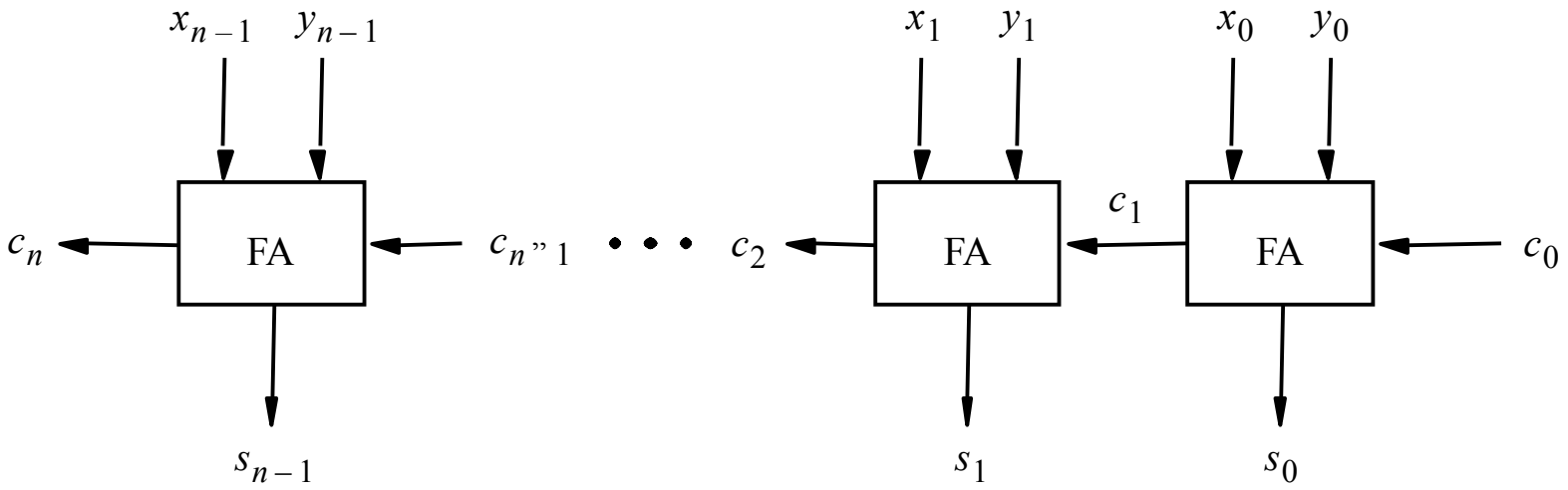
Figure 5.29 A 16-bit adder with carry and overflow

# Somador Ripple Carry

- Atraso para um somador de n bits:

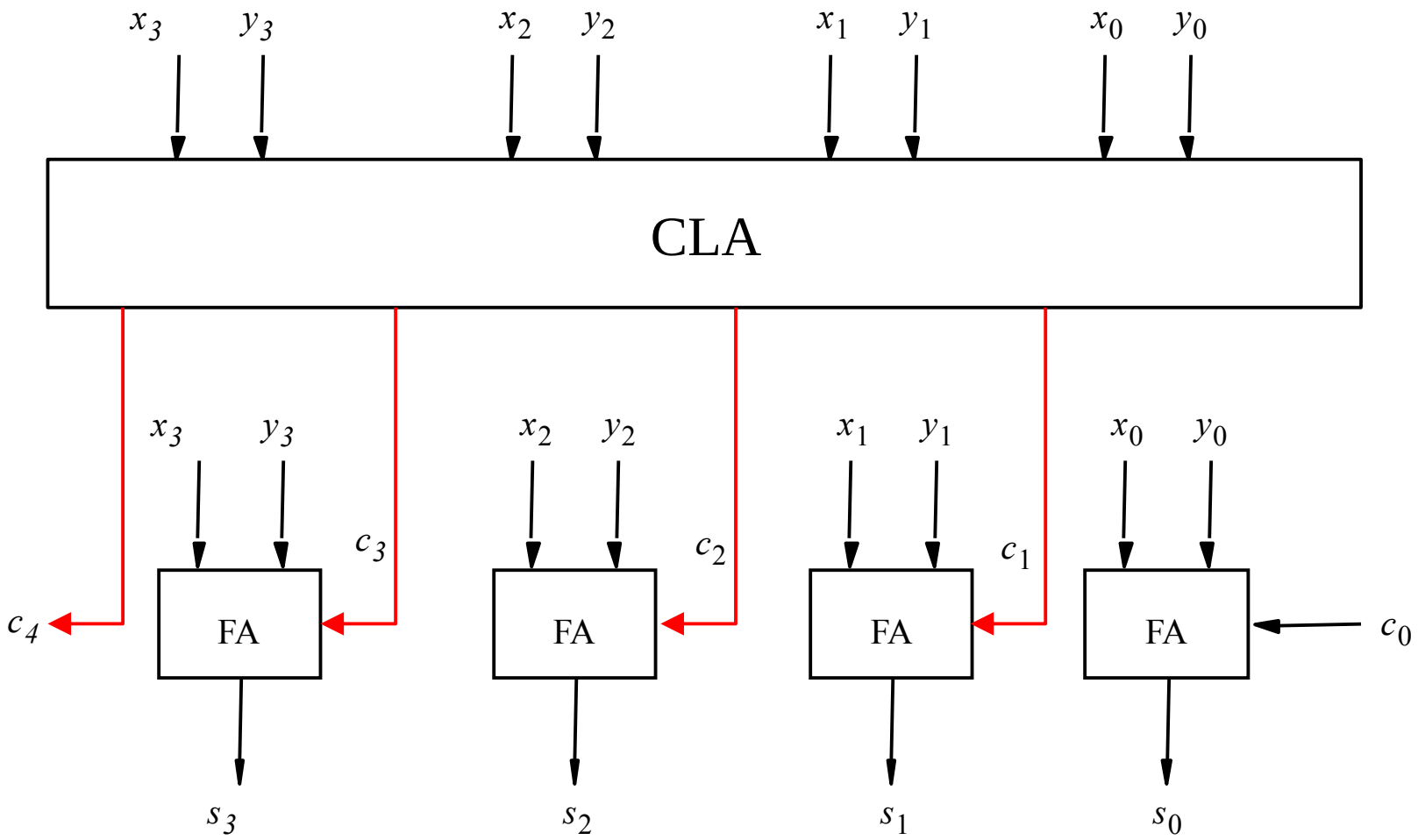
$$t_{ripple} = Nt_{FA}$$

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



# Antecipação de Carry: Carry Look Ahead (CLA)

- Aplicado para módulo de 4 bits



# CLA: Generate e Propagate

- Para gerar carries com atraso menor e fixo
- Observar para o bit i
  - Carry é gerado sempre independente das entradas e dos carries de nível anterior:
    - $g_i = x_i\, y_i$
  - Carry é propagado sempre independente das entradas e dos carries de nível anterior:
    - $p_i = x_i + y_i$
    - observar que um carry de entrada é morto/killed se:
      - $\sim x_i \,.\, \sim y_i$
      - Que é exatamente $\sim p_i$

# CLA: Como gerar os carries a partir de g e p

$c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 c_1$ $\quad c_2 = g_1 + p_1\, g_0 + p_1\, p_0 c_0$

$c_3 = g_2 + p_2 c_2$ $\quad c_3 =$

$c_4 = g_3 + p_3 c_3$ $\quad c_4 =$

- Atraso:
  - entradas $\Rightarrow g_i\, p_i$ (1G)
  - $g_i\, p_i \Rightarrow$ carry (2G) : 1 AND seguido de 1 OR
  - carry $\Rightarrow$ saídas (2G)
- Total: 5G, independente de n

# Codificação em BCD

"No mundo há 10 tipos de pessoas: as que sabem contar em binário e as que não sabem"

# BCD

| Decimal digit | BCD code |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Table 5.3 Binary-coded decimal digits

# Adição Usando BCD

| | | |
|---|---|---|
| X | 0 1 1 1 | 7 |
| + Y | + 0 1 0 1 | + 5 |
| Z | 1 1 0 0 | 12 |
| | + 0 1 1 0 | |
| carry → | 1 0 0 1 0 | |
| | S = 2 | |

| | | |
|---|---|---|
| X | 1 0 0 0 | 8 |
| + Y | + 1 0 0 1 | + 9 |
| Z | 1 0 0 0 1 | 17 |
| | + 0 1 1 0 | |
| carry → | 1 0 1 1 1 | |
| | S = 7 | |

Passou de 10? Remove 10:

$S - 10 = S - 9 - 1$
$= S + K_2(9_{10}) - 1$
$= S + K_1(9_{10}) + 1 - 1$
$= S + \text{not}(1001_2)$
$= S + 0110_2$
$= S + 6_{10}$

Raciocínio Alternativo
Passou de 10?
Remove 10 (carry=1)

$S - 10 = S - (16 - 6)$
$= S + 6 - 16$
$= (S + 6) - 16$

soma

carry

# Somador em BCD

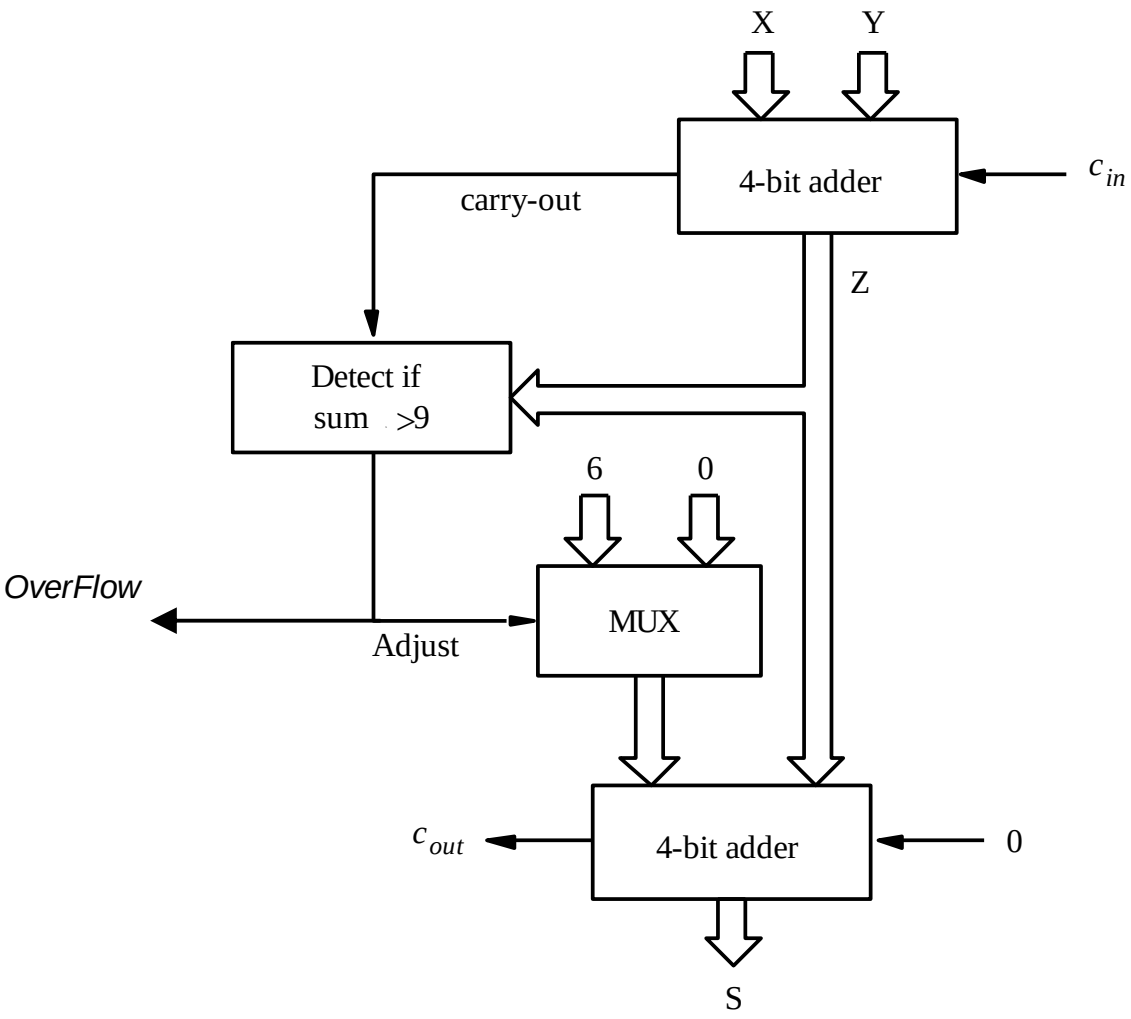



Figure 5.37 Block diagram for a one-digit BCD adder

# Somador de um Dígito BCD

$$\begin{array}{r} z_3\ z_2\ z_1\ z_0 \\ +\ 0\ \ 1\ \ 1\ \ 0 \\ \hline \end{array}$$

$c_{out} = 1$ ?

$c_{out} = d_{out} + z_2\, z_3 + z_1\, z_3$

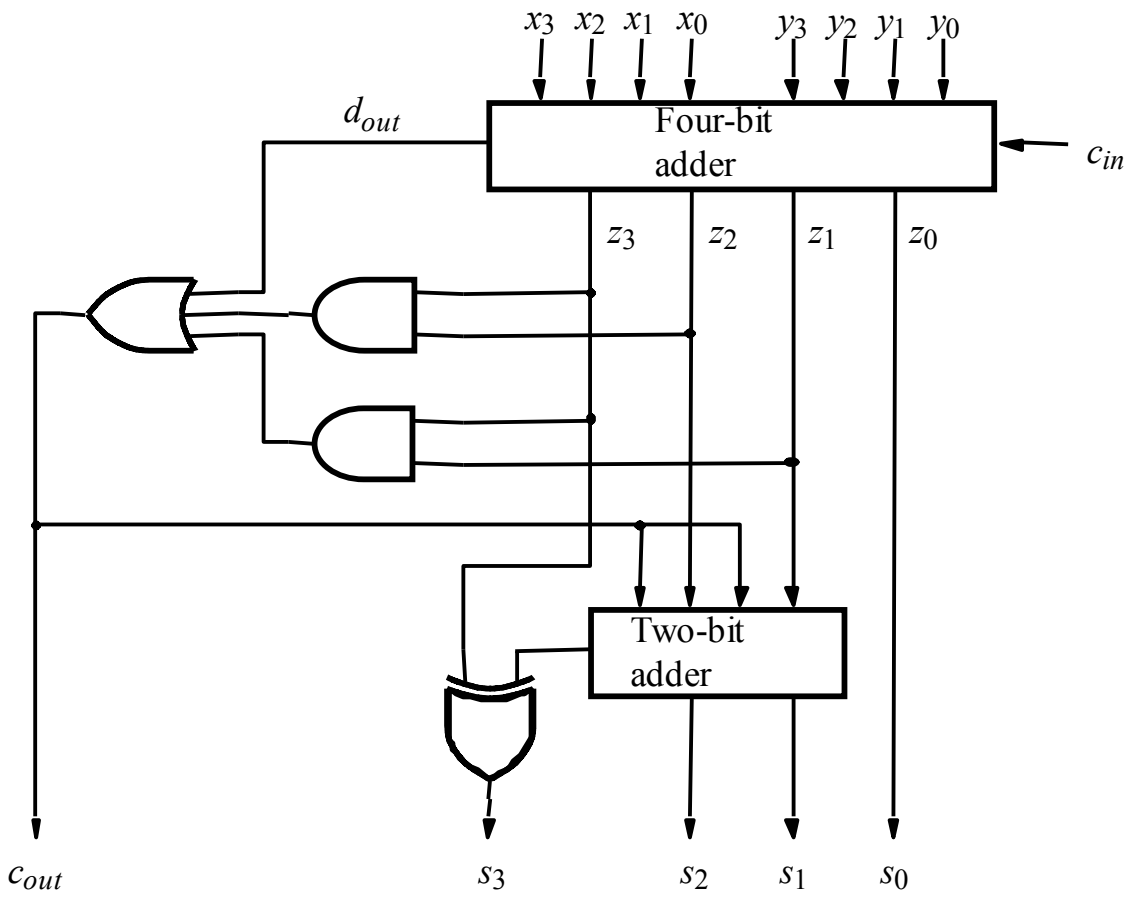



Figure 5.40 Circuit for a one-digit BCD adder

# Somador BCD

```
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 ;
```

Figure 5.38 VHDL code for a one-digit BCD adder