

# Projeto de Sistemas Digitais

## Capítulo 5 – Representação Numérica e Circuitos Aritméticos

Ricardo Pannain  
[pannainr@gmail.com](mailto:pannainr@gmail.com)

## Conversão Decimal-Binária

Convert  $(857)_{10}$

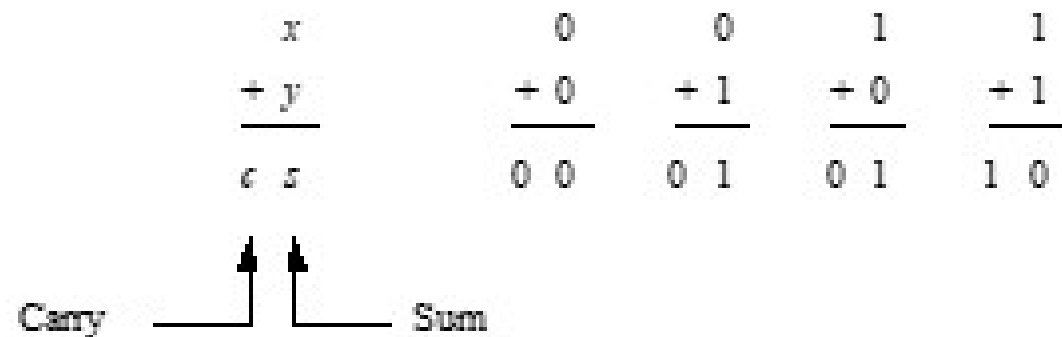
				Remainder	
$857 \div 2$	$=$	428		1	LSB
$428 \div 2$	$=$	214		0	
$214 \div 2$	$=$	107		0	
$107 \div 2$	$=$	53		1	
$53 \div 2$	$=$	26		1	
$26 \div 2$	$=$	13		0	
$13 \div 2$	$=$	6		1	
$6 \div 2$	$=$	3		0	
$3 \div 2$	$=$	1		1	
$1 \div 2$	$=$	0		1	MSB

Result is  $(1101011001)_2$

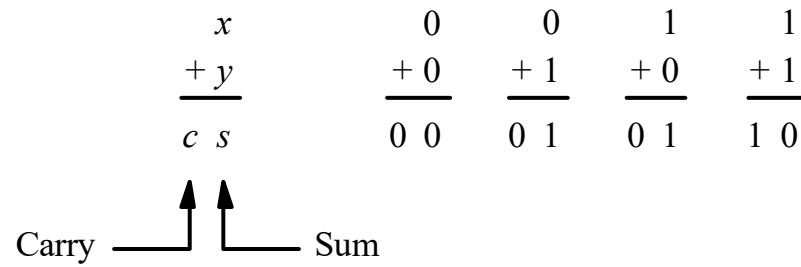
# Números em diferentes Bases

Decimal	Binary	Octal	Hexadecimal
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12

Projetar um circuito que gere o resultado e o carry out de uma soma de dois números de 1 bit.



x	y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



(a) The four possible cases

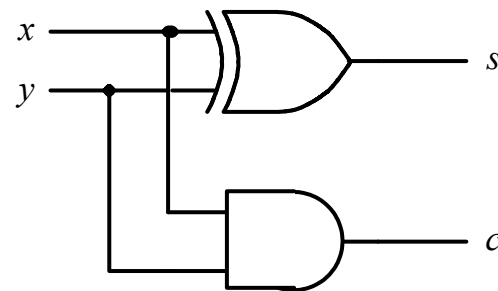
$$\text{Sum} = x \text{ XOR } y$$

$$\text{Carry} = x \cdot y$$

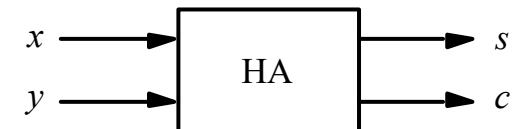
Meio Somador  
(Half - adder)

$x \quad y$		Carry $c$	Sum $s$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table



(c) Circuit



(d) Graphical symbol

## Adição Binária

$$\begin{array}{rcl}
 X = x_4x_3x_2x_1x_0 & 0\ 1\ \boxed{1}\ 1\ 1 & (15)_{10} \\
 + Y = y_4y_3y_2y_1y_0 & 0\ 1\ 0\ 1\ 0 & (10)_{10} \\
 \hline
 & 1\ \boxed{1}\ 1\ 0 & \leftarrow \text{Generated carries} \\
 \hline
 S = s_4s_3s_2s_1s_0 & 1\ 1\ \boxed{0}\ 0\ 1 & (25)_{10}
 \end{array}$$

# Somador Completo (Full-adder)

$c_i$	$x_i$	$y_i$	$c_{i+1}$	$s_i$
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

(a) Truth table

$x_i y_i$	00	01	11	10
$c_i$				
0		1		1
1	1		1	

$$s_i = x_i \oplus y_i \oplus c_i$$

$x_i y_i$	00	01	11	10
$c_i$				
0			1	
1		1	1	1

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

(b) Karnaugh maps

OBS

$$S = x' y' c + x' y c' + x y c + x y' c' = x \text{ XOR } y \text{ XOR } c$$

OBS

$$S = x' y' c + x' y c' + x y c + x y' c' = x \text{ XOR } y \text{ XOR } c$$

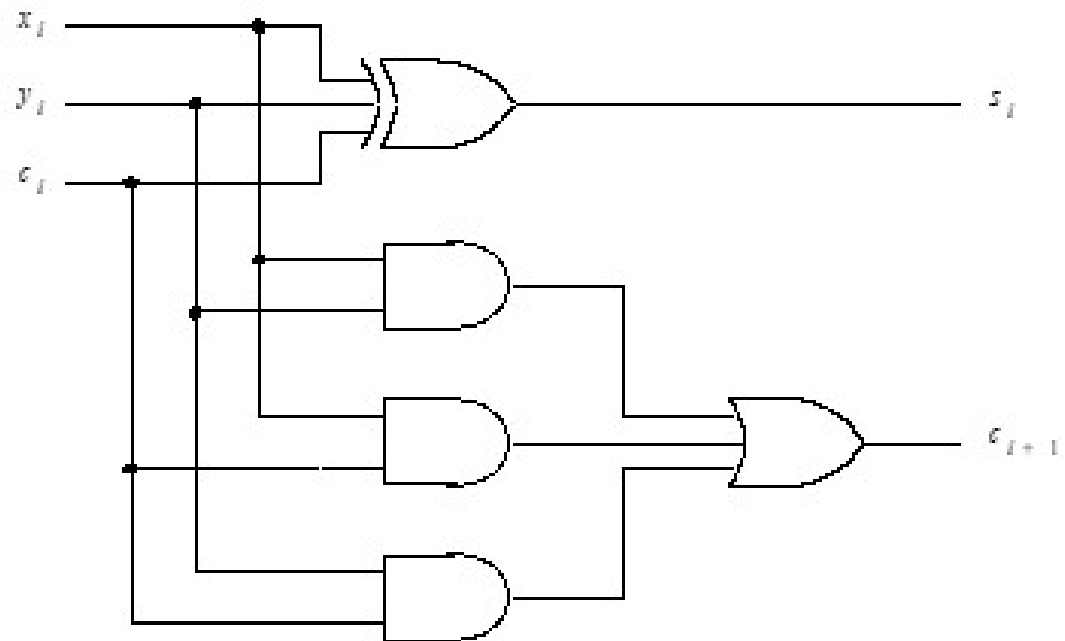
$$S = (x' y' + xy) c + (x' y + xy') c' = a' c + a c' = a \text{ xor } c = x \text{ XOR } y \text{ XOR } c$$

$$a = x' y + xy' = x \text{ XOR } y$$

$$a' = x' y' + xy = x \text{ XNOR } y$$



## Somador Completo (Full-adder)



(c) Circuit

# Somador Completo (Full-adder)

$c_i$	$x_i$	$y_i$	$c_{i+1}$	$s_i$
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

(a) Truth table

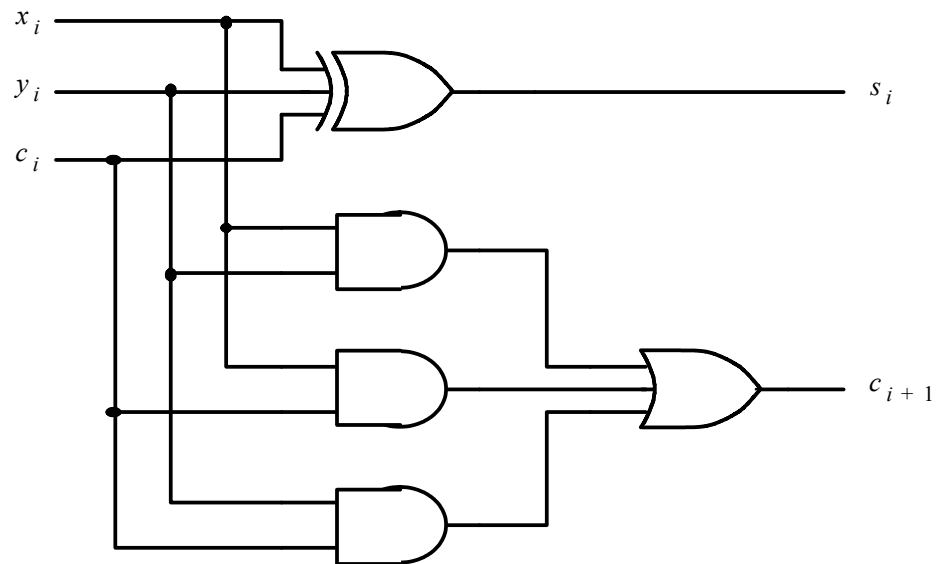
$c_i \backslash x_i y_i$	00	01	11	10
0		1		1
1	1		1	

$$s_i = x_i \oplus y_i \oplus c_i$$

$c_i \backslash x_i y_i$	00	01	11	10
0			1	
1		1	1	1

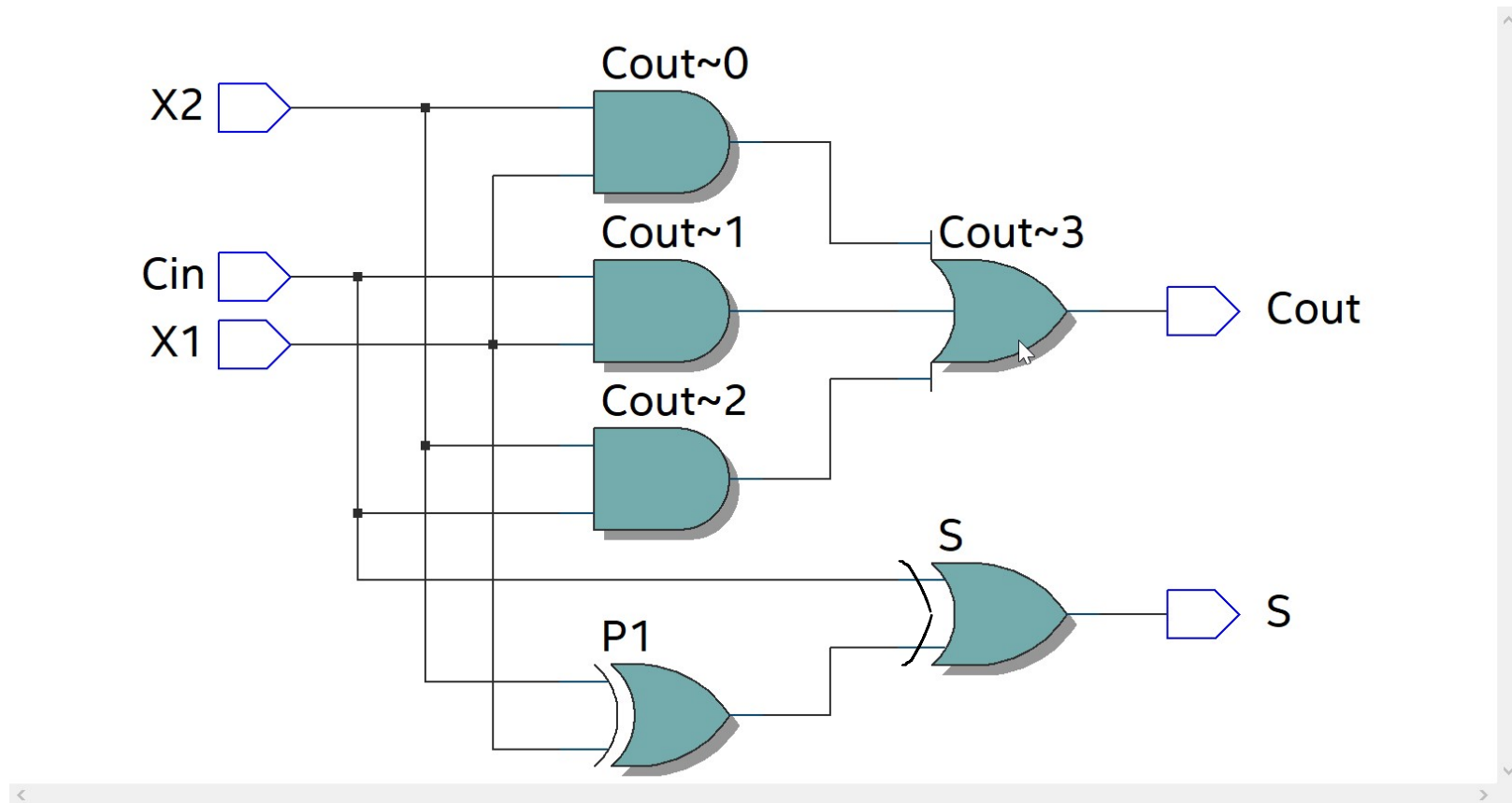
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

(b) Karnaugh maps



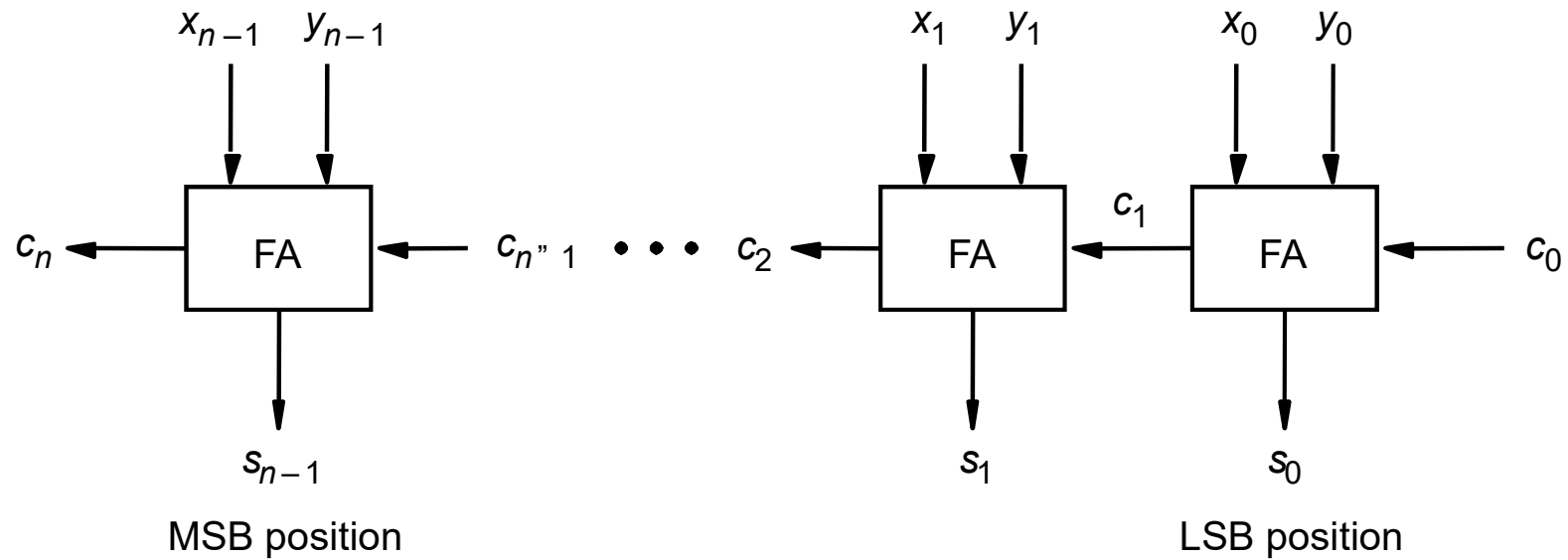
(c) Circuit

## Projetar um somador de $n$ -bits

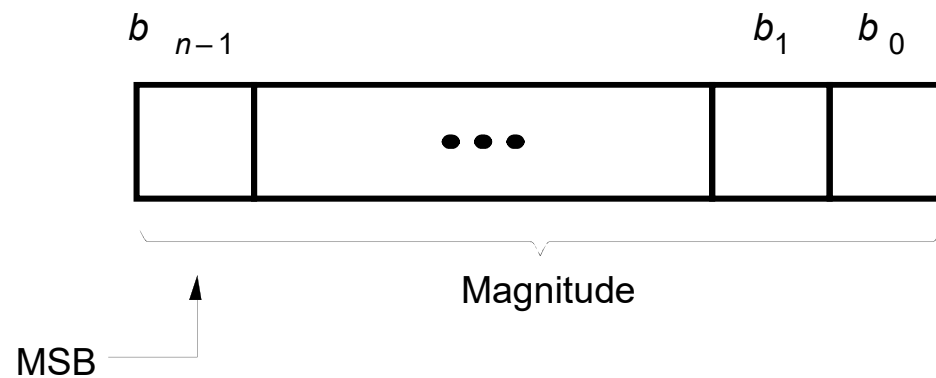


Full Adder (FA)

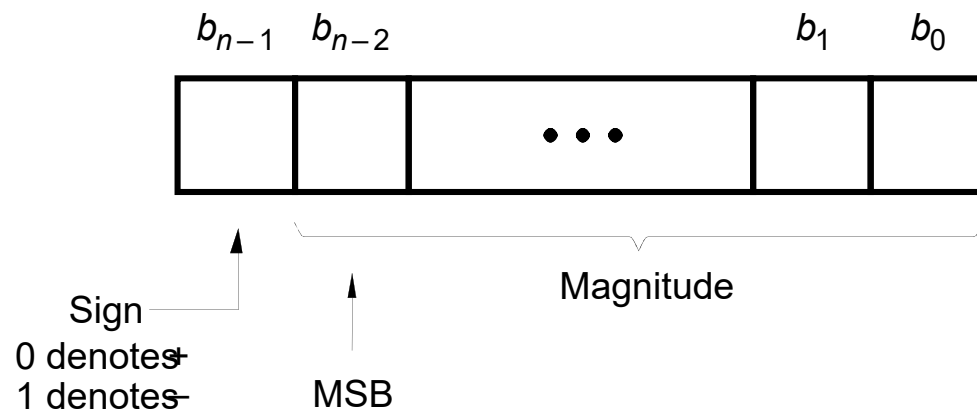
## Somador de $n$ -bits – ripple carry



# Formatos para representação de números inteiros



(a) Unsigned number



(b) Signed number

## Números inteiros sinalizados com 4 bits

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
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

SM	C1	C2
+7 -> 0111	0111	0111
-7 -> 1111	1000	1001
+5 -> 0101	0101	0101
-5 -> 1101	1010	1011

$$s = a - b = a + (-b) = a + (\text{not } b) + 1$$

## Exemplos de adição em complemento de 1

$$\begin{array}{r}
 (+5) \\
 + (+2) \\
 \hline
 (+7)
 \end{array}
 \qquad
 \begin{array}{r}
 0101 \\
 + 0010 \\
 \hline
 0111
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \\
 + (+2) \\
 \hline
 (-3)
 \end{array}
 \qquad
 \begin{array}{r}
 1010 \\
 + 0010 \\
 \hline
 1100
 \end{array}$$

$$\begin{array}{r}
 (+5) \\
 + (-2) \\
 \hline
 (+3)
 \end{array}
 \qquad
 \begin{array}{r}
 0101 \\
 + 1101 \\
 \hline
 10010 \\
 \text{Carrinho} \rightarrow 1 \\
 \hline
 0011
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \\
 + (-2) \\
 \hline
 (-7)
 \end{array}
 \qquad
 \begin{array}{r}
 1010 \\
 + 1101 \\
 \hline
 10111 \\
 \text{Carrinho} \rightarrow 1 \\
 \hline
 1000
 \end{array}$$




## Exemplos de adição em complemento de 2

$$\begin{array}{r} (+5) \quad 0101 \\ + (+2) \quad +0010 \\ \hline (+7) \quad 0111 \end{array}$$

$$\begin{array}{r} (-5) \quad 1011 \\ + (+2) \quad +0010 \\ \hline (-3) \quad 1101 \end{array}$$

$$\begin{array}{r} (+5) \quad 0101 \\ + (-2) \quad +1110 \\ \hline (+3) \quad 10011 \end{array}$$

  
ignore

$$\begin{array}{r} (-5) \quad 1011 \\ + (-2) \quad +1110 \\ \hline (-7) \quad 11001 \end{array}$$

  
ignore

$$s = a - b = a + (-b) = a + (\text{not } b) + 1$$

$$s = 5 - 2 = 0101 + (1101 + 1) = 1 \quad 0011$$

## Exemplos de subtração em complemento de 1

$$\begin{array}{r}
 (+5) \\
 - (+2) \\
 \hline
 (+3)
 \end{array}
 \quad
 \begin{array}{r}
 0101 \\
 - 0010 \\
 \hline
 \end{array}
 \Rightarrow
 \begin{array}{r}
 0101 \\
 + 1110 \\
 \hline
 10011
 \end{array}$$

↑ ignore

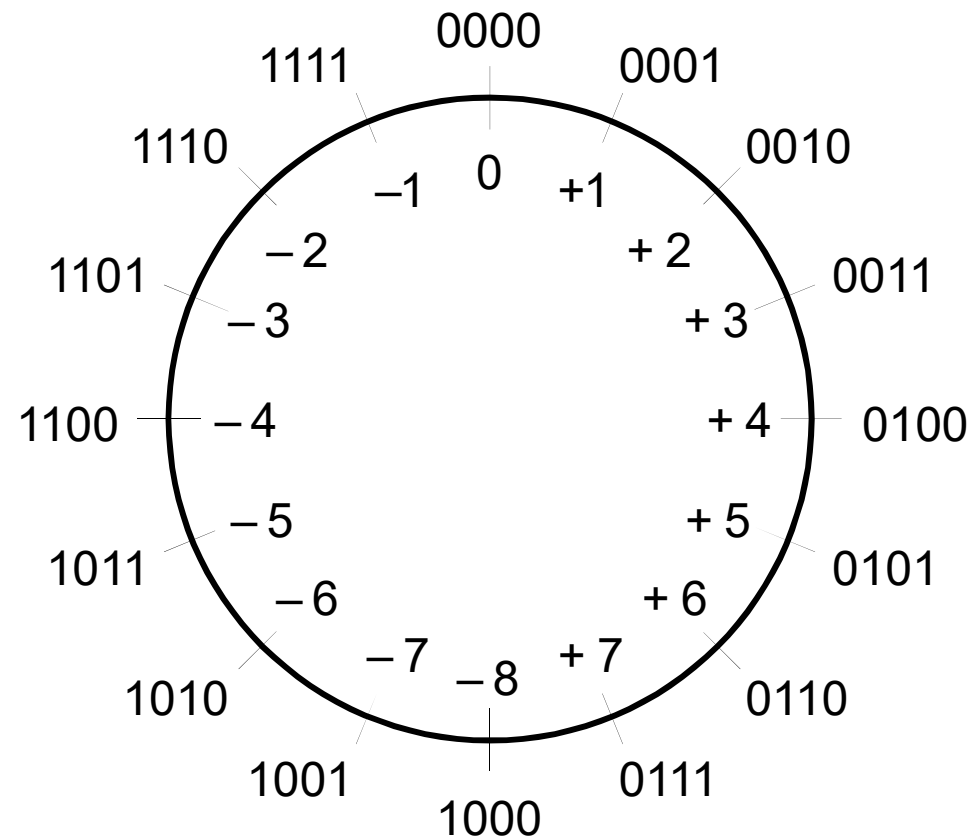
$$\begin{array}{r}
 (-5) \\
 - (+2) \\
 \hline
 (-7)
 \end{array}
 \quad
 \begin{array}{r}
 1011 \\
 - 0010 \\
 \hline
 \end{array}
 \Rightarrow
 \begin{array}{r}
 1011 \\
 + 1110 \\
 \hline
 11001
 \end{array}$$

↑ ignore

$$\begin{array}{r}
 (+5) \\
 - (-2) \\
 \hline
 (+7)
 \end{array}
 \quad
 \begin{array}{r}
 0101 \\
 - 1110 \\
 \hline
 \end{array}
 \Rightarrow
 \begin{array}{r}
 0101 \\
 + 0010 \\
 \hline
 0111
 \end{array}$$

$$\begin{array}{r}
 (-5) \\
 - (-2) \\
 \hline
 (-3)
 \end{array}
 \quad
 \begin{array}{r}
 1011 \\
 - 1110 \\
 \hline
 \end{array}
 \Rightarrow
 \begin{array}{r}
 1011 \\
 + 0010 \\
 \hline
 1101
 \end{array}$$

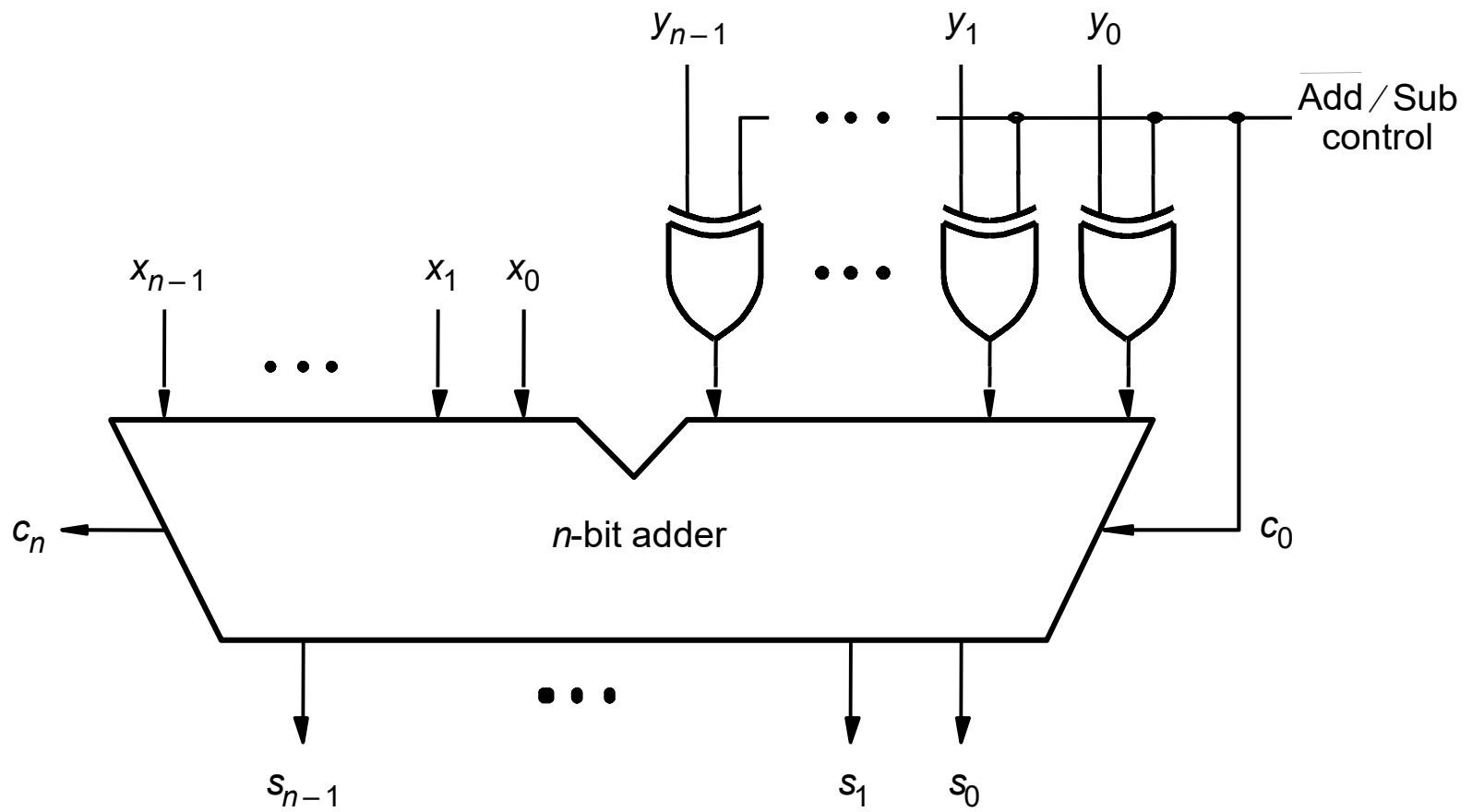
## Interpretação Gráfica de números de 4 bits em complemento de 2



Projetar um circuito somador/subtrator

$$S = A - B = A + (-B) = A + B' + 1$$

## Somador / Subtrator



## Exemplo de ocorrência de overflow

$$\begin{array}{r}
 (+7) \quad 0111 \\
 + (+2) \quad +0010 \\
 \hline
 (+9) \quad 1001 \\
 c_4 = 0 \\
 c_3 = 1
 \end{array}$$

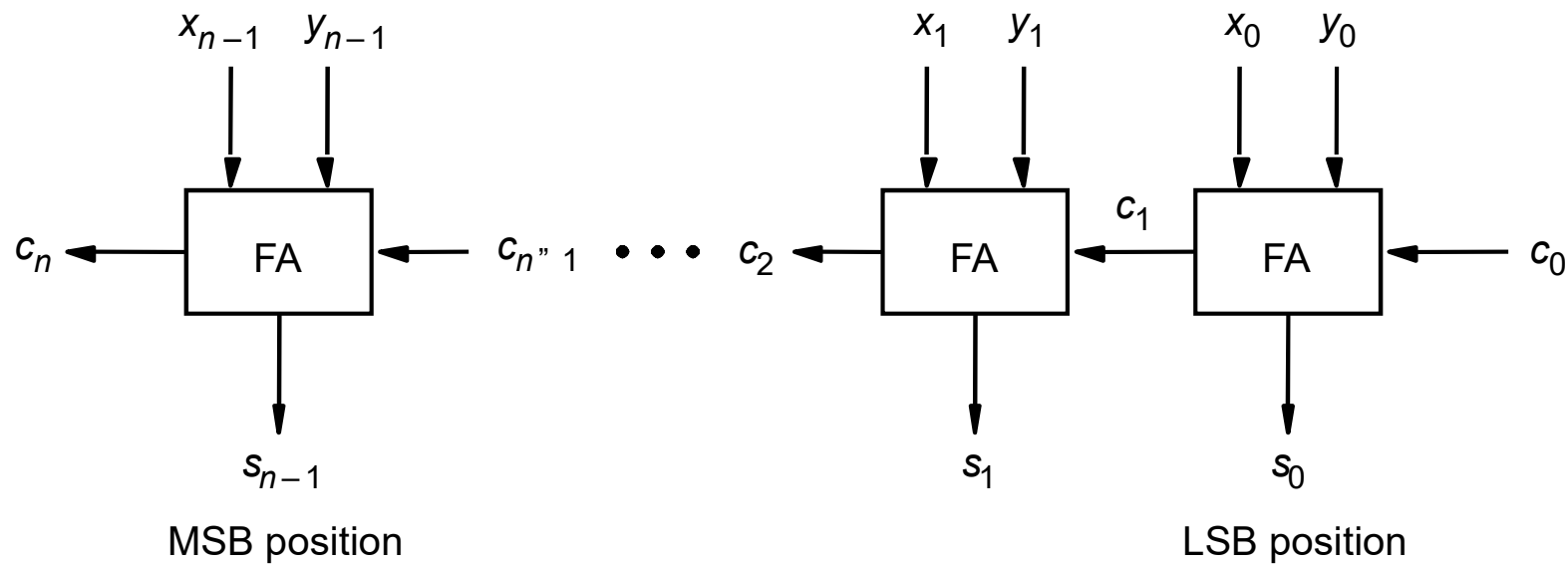
$$\begin{array}{r}
 (-7) \quad 1001 \\
 + (+2) \quad +0010 \\
 \hline
 (-5) \quad 1011 \\
 c_4 = 0 \\
 c_3 = 0
 \end{array}$$

$$\begin{array}{r}
 (+7) \quad 0111 \\
 + (-2) \quad +1110 \\
 \hline
 (+5) \quad 10101 \\
 c_4 = 1 \\
 c_3 = 1
 \end{array}$$

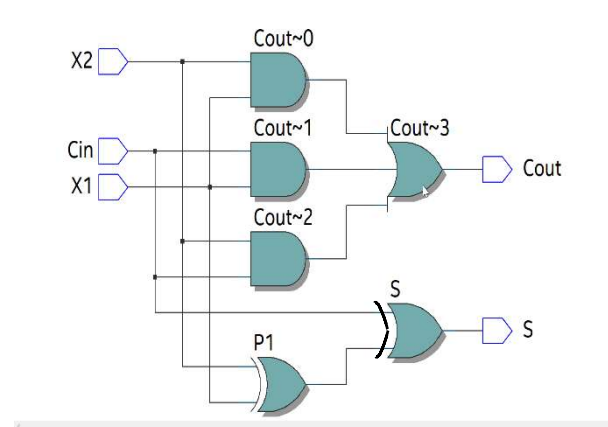
$$\begin{array}{r}
 (-7) \quad 1000 \\
 \quad 1001 \\
 + (-2) \quad +1110 \\
 \hline
 (-9) \quad 10111 \\
 c_4 = 1 \\
 c_3 = 0
 \end{array}$$

$$O_v = S_x S_y S_r' + S_x' S_y' S_r$$

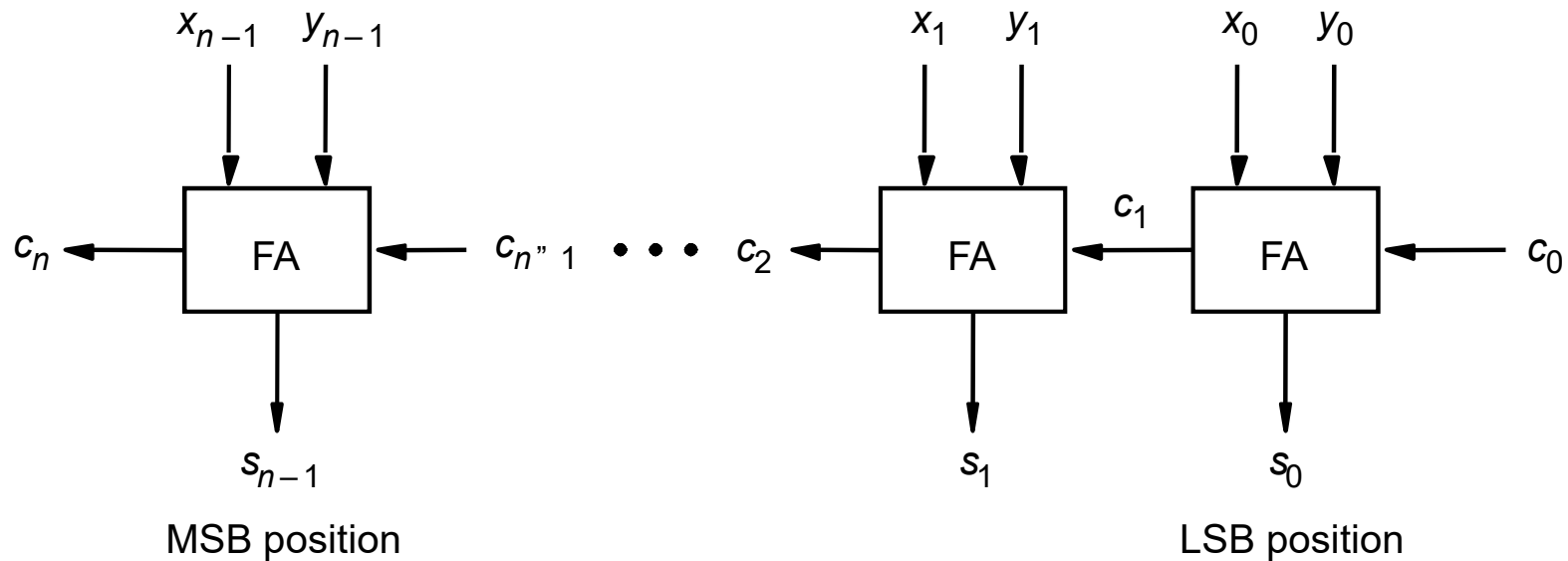
Calcular o tempo necessário para uma soma de 2 número de 4 bits,  
em um somador ripple carry,  
supondo que o atraso de uma porta seja  $t$



Full Adder



Calcular o tempo necessário para uma soma de 2 número de 4 bits, supondo que o atraso de uma porta seja  $t$

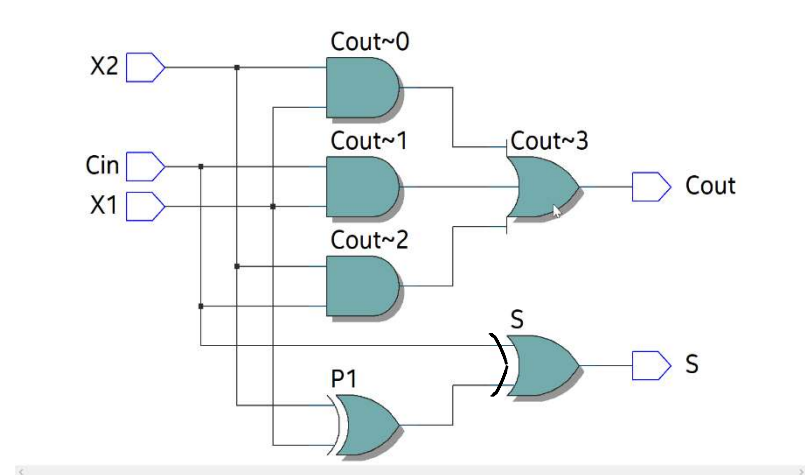


$s_0 \rightarrow 2t$   
 $c_1 \rightarrow 2t$

$s_1 \rightarrow 3t$   
 $c_2 \rightarrow 4t$

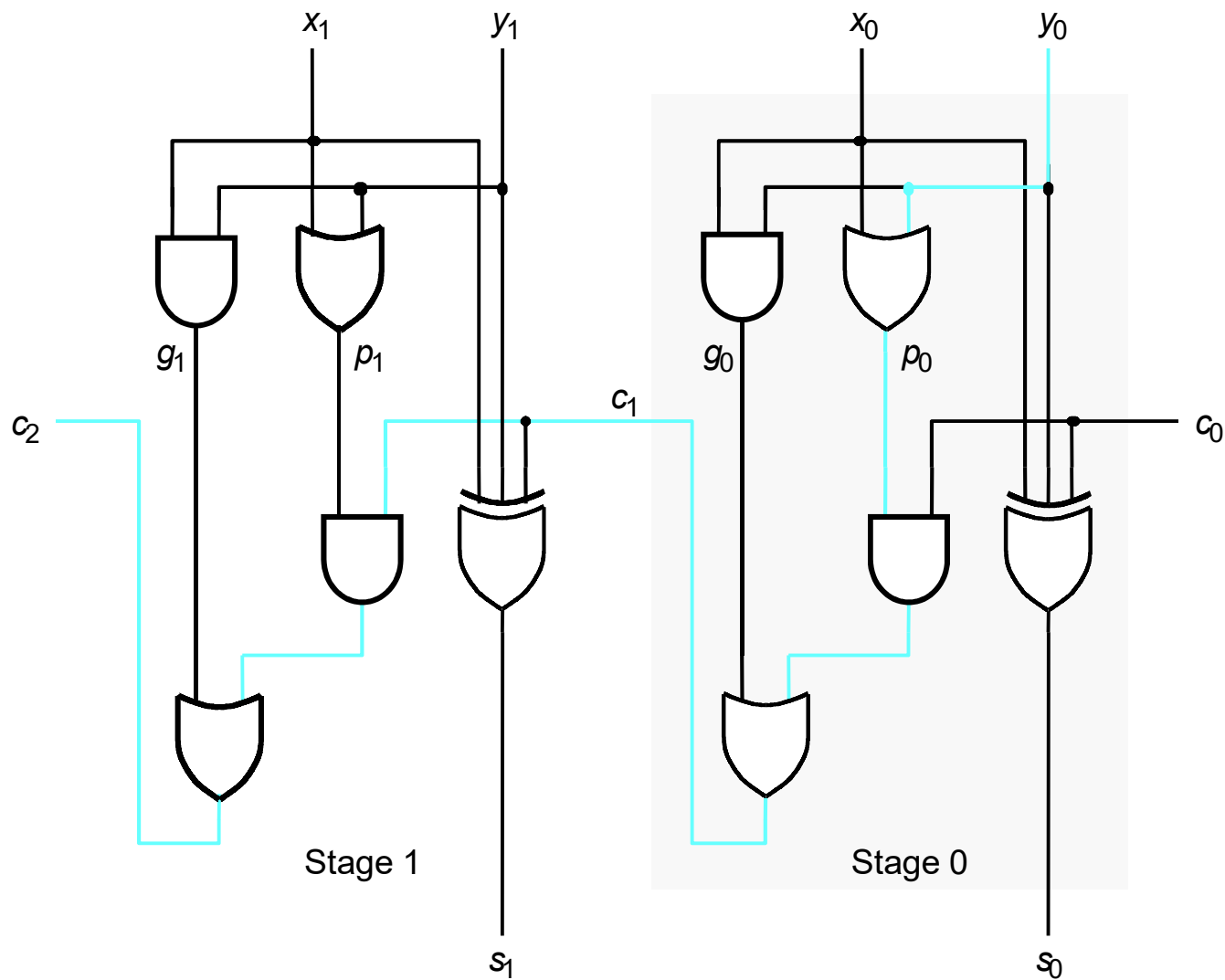
$s_2 \rightarrow 5t$   
 $c_3 \rightarrow 6t$

$s_3 \rightarrow 7t$   
 $c_4 \rightarrow 8t$





## Somador ripple-carry com geração/propagação de sinais



## Somador carry-lookahead

Como fazer com que um somador trabalhe mais rápido ?

Obs:  $g_i = x_i y_i$

$p_i = (x_i + y_i)$

$$c_1 = x_0 y_0 + x_0 c_0 + y_0 c_0 = g_0 + p_0 c_0 \rightarrow$$

$$c_2 = x_1 y_1 + x_1 c_1 + y_1 c_1 = g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0 \rightarrow$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \rightarrow$$

$$c_4 = g_3 + p_3 c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \rightarrow$$

## Somador carry-lookahead

Como fazer com que um somador trabalhe mais rápido ?

Obs:  $g_i = x_i y_i$

$p_i = (x_i + y_i)$

$$c_1 = x_0 y_0 + x_0 c_0 + y_0 c_0 = g_0 + p_0 c_0 \rightarrow 3t$$

$$c_2 = x_1 y_1 + x_1 c_1 + y_1 c_1 = g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0 \rightarrow 3t$$

$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \rightarrow 3t$$

$$c_4 = g_3 + p_3 c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \rightarrow 3t$$

## Somador carry-lookahead

Como fazer com que um somador trabalhe mais rápido ?

Obs:  $g_0 = x_0y_0$

$$(x_0 + y_0)c_0 = p_0c_0 \rightarrow p_0 = x_0 + y_0$$

$$c_1 = x_0y_0 + x_0c_0 + y_0c_0 = g_0 + p_0c_0 \rightarrow 3t$$

$$s_0 \rightarrow 2t$$

$$c_2 = x_1y_1 + x_1c_1 + y_1c_1 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0c_0 \rightarrow 3t$$

$$s_1 \rightarrow 4t$$

$$c_3 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

$$s_2 \rightarrow 4t$$

$$c_4 = g_3 + p_3c_3 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$$

$$s_3 \rightarrow 4t$$

$$c_4 \rightarrow 3t$$

$$s_0 \rightarrow 2t$$

$$c_1 = g_0 + p_0c_0 \rightarrow 3t$$

$$s_1 \rightarrow 4t$$

Somador carry-lookahead

FANIN = 3

$$c_2 = g_1 + p_1g_0 + p_1p_0c_0 \rightarrow 3t$$

$$s_2 \rightarrow 4t$$

$$c_3 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0 \rightarrow 4t$$

$$p_i \text{ e } g_1 \rightarrow 1t$$

$$p_2p_1g_0, r = p_2p_1p_0 \rightarrow 2t$$

$$p_2p_1p_0c_0, g_2 + p_2g_1 + p_2p_1g_0 \rightarrow 3t$$

$$\text{Or final} \rightarrow 4t$$

$$s_3 \rightarrow 5t$$

$$c_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0 \rightarrow 4t$$

1t para  $p_i$  e  $g_i$

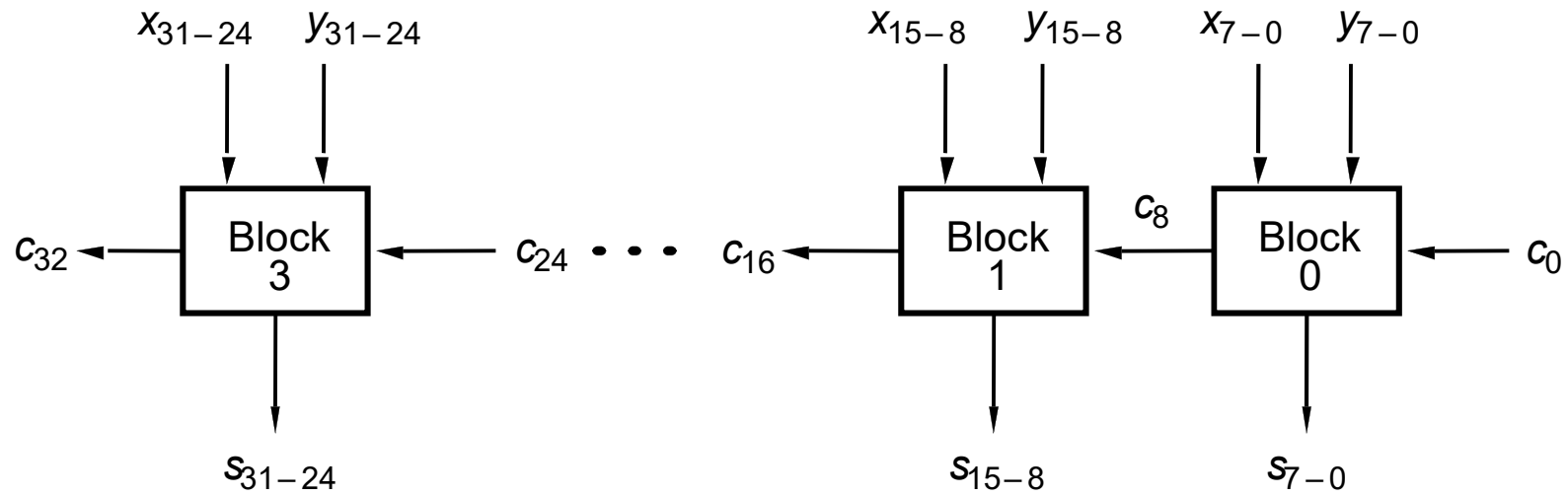
2t ands ( $p_3g_2, p_3p_2g_1, p_3p_2, p_1g_0, p_3p_2p_1, p_0c_0$ )

3t  $g_3 + p_3g_2 + p_3p_2g_1; p_3p_2p_1g_0; p_3p_2p_1p_0c_0$

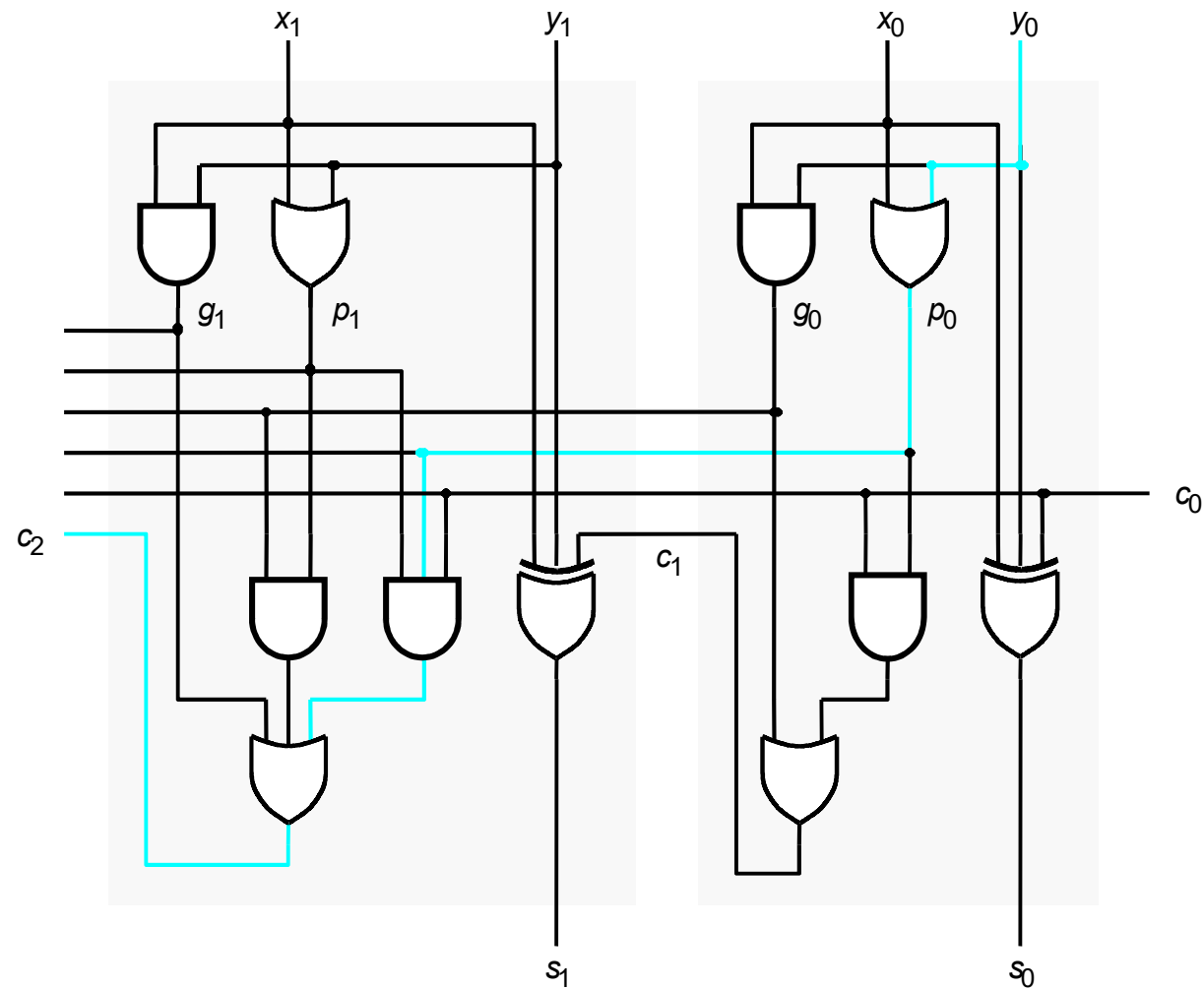
4t finalizo o OR;

$$c_4 \rightarrow 4t$$

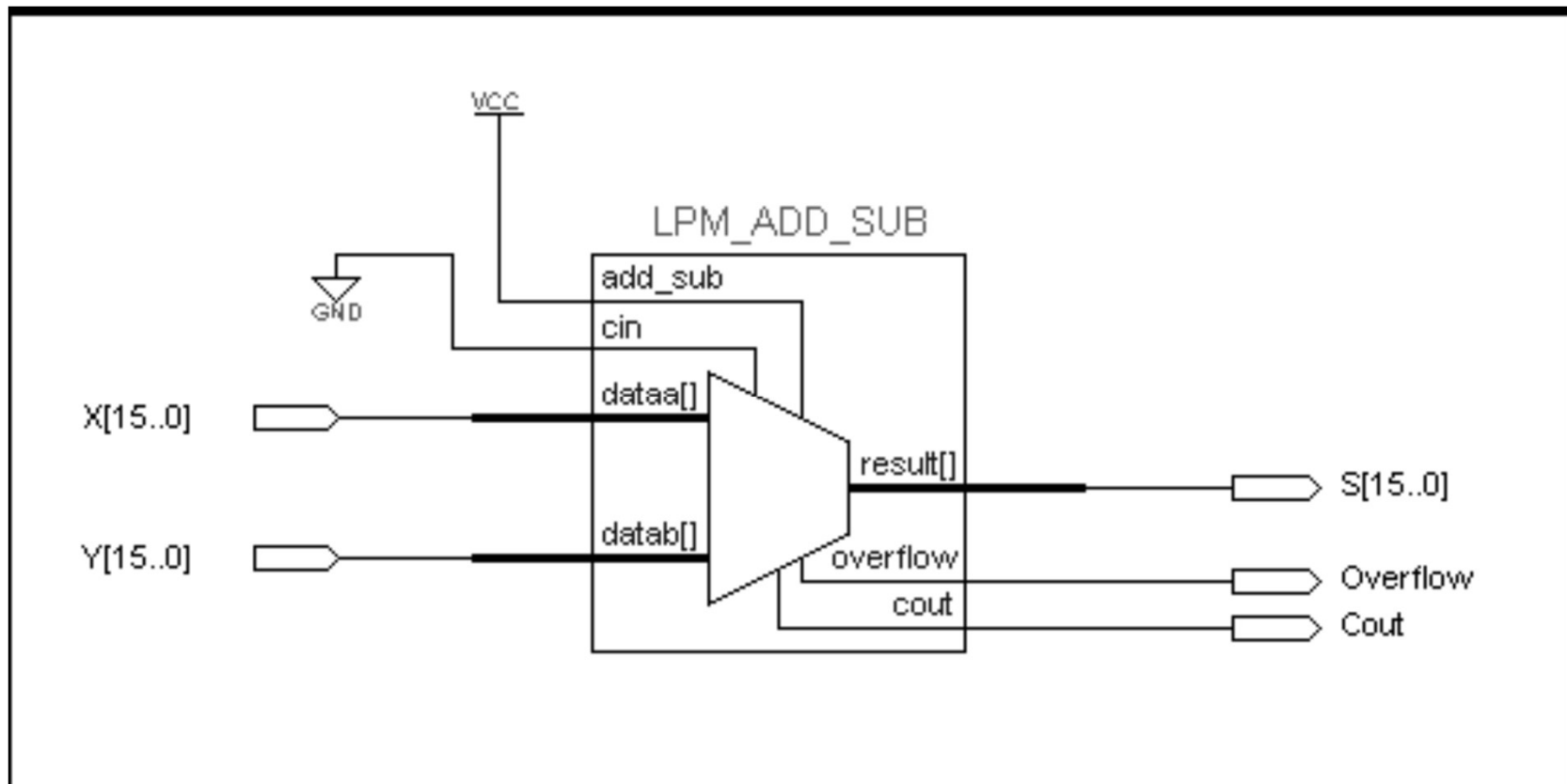
## Somador carry-lookahead com ripple-carry entre blocos



## Dois estágios de um somador carry-lookahead

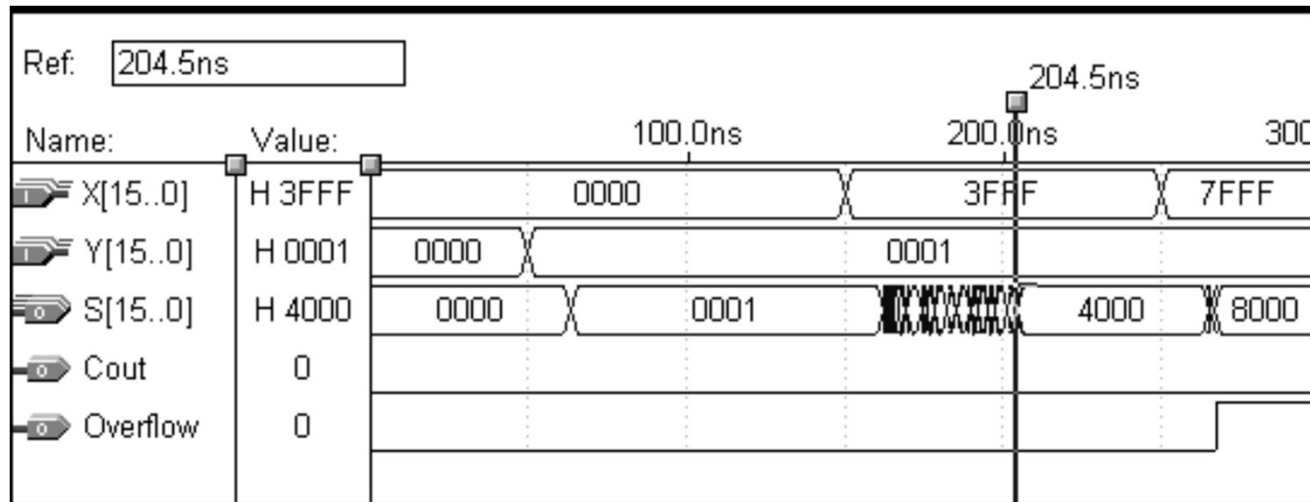


## Esquemático usando um módulo LPM adder/subtractor

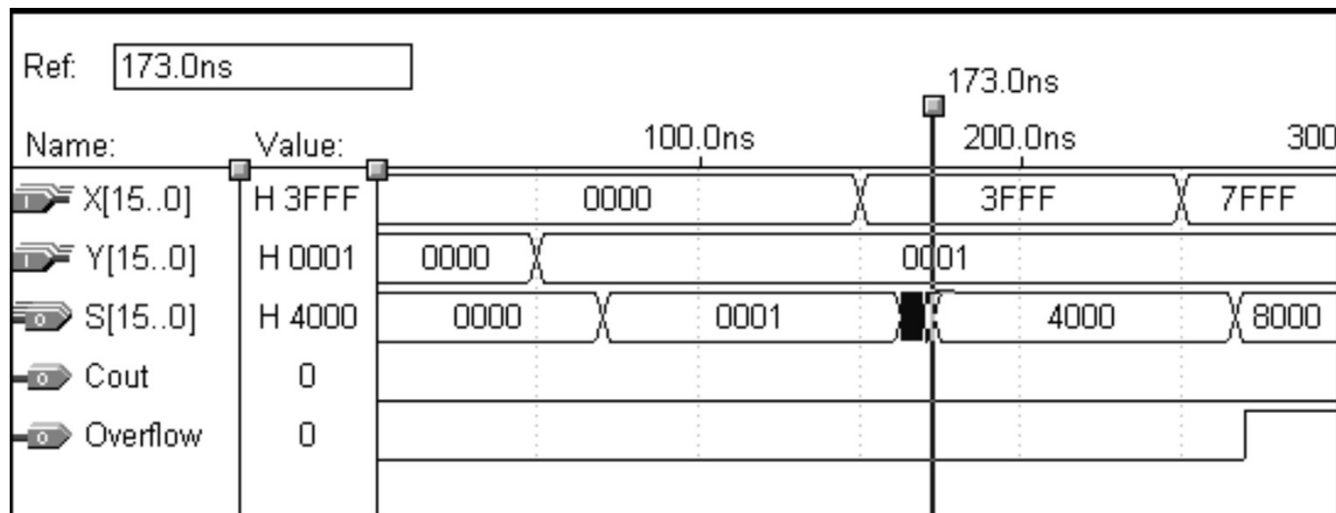




## Resultado da simulação de um módulo LPM adder/subtrator



Optimized for cost



Optimized for speed

## Código VHDL para um full-adder

```
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 ;
```

## Código VHDL para um somador de 4 bits

```
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 ;
    COMPONENT fulladd
        PORT (    Cin, x, y : IN        STD_LOGIC ;
                 s, Cout   : OUT       STD_LOGIC ) ;
    END COMPONENT ;
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 ( c3, x3, y3, s3, c4 ) ;

END Structure ;
```

## Declaração de um Package

```
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 ;
```

## Usando um package para somador de 4 bits

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

## Somador de 4 bits usando sinais multibit

```
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 DOWNT0 0) ;
           S         : OUT     STD_LOGIC_VECTOR(3 DOWNT0 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 ;
```

## Código VHDL code para um somador de 16-bit

```
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 ;
```

## Somador de 16-bit com carry e overflow

```
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 DOWNT0 0) ;
           S             : OUT     STD_LOGIC_VECTOR(15 DOWNT0 0) ;
           Cout, Overflow : OUT     STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
    SIGNAL Sum : STD_LOGIC_VECTOR(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 ;
```



## Use of the arithmetic    Uso de package com circuito aritmético

```
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 ;

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 ;
```

Um somador de 16-bit adder usando sinais INTEGER

```
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 ;
```

$$\begin{array}{r}
 10 \\
 \times 10 \\
 \hline
 00 \\
 10 \\
 \hline
 0100
 \end{array}$$

## Circuito multiplicador 4 X 4

Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	× 1 0 1 1
		<hr/>
		1 1 1 0
		1 1 1 0
		0 0 0 0
		1 1 1 0
		<hr/>
Product P	(154)	1 0 0 1 1 0 1 0

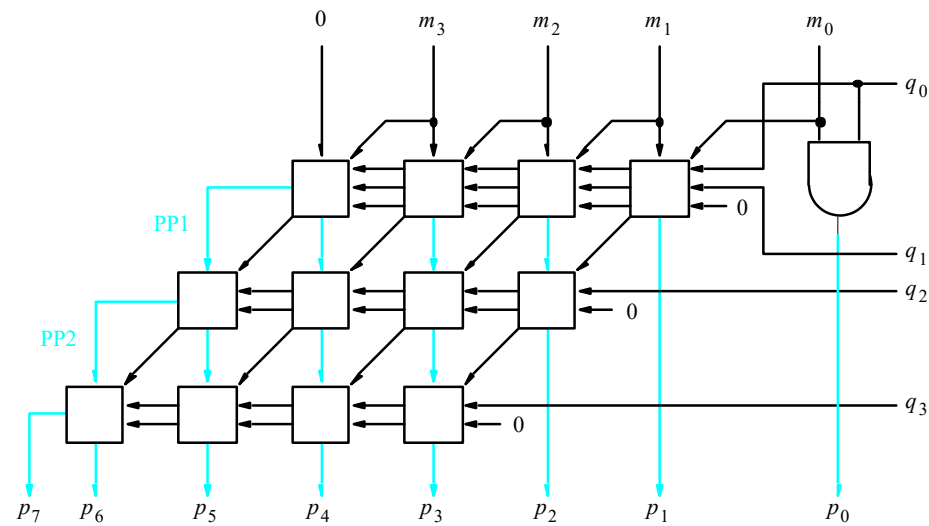
(a) Multiplication by hand

## Circuito multiplicador 4 X 4

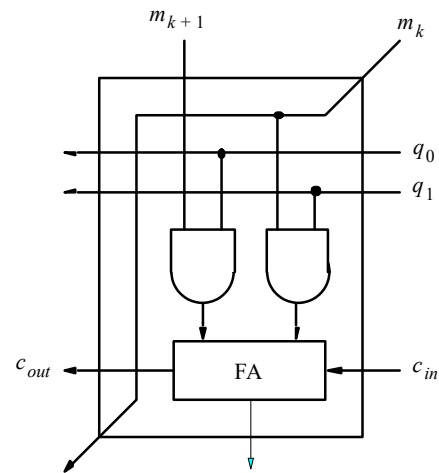
Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	× 1 0 1 1
		-----
Partial product 0		1 1 1 0
		+ 1 1 1 0
		-----
Partial product 1		1 0 1 0 1
		+ 0 0 0 0
		-----
Partial product 2		0 1 0 1 0
		+ 1 1 1 0
		-----
Product P	(154)	1 0 0 1 1 0 1 0

(b) Implementação da multiplicação em hardware

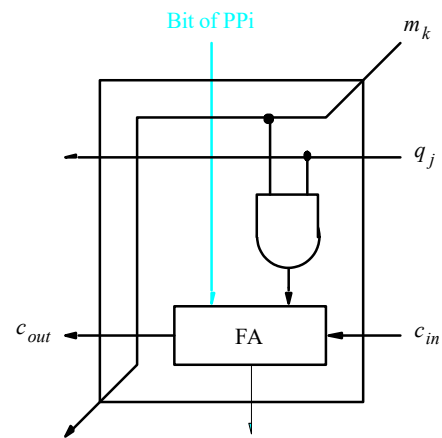
# Circuito multiplicador 4 X 4



(a) Structure of the circuit

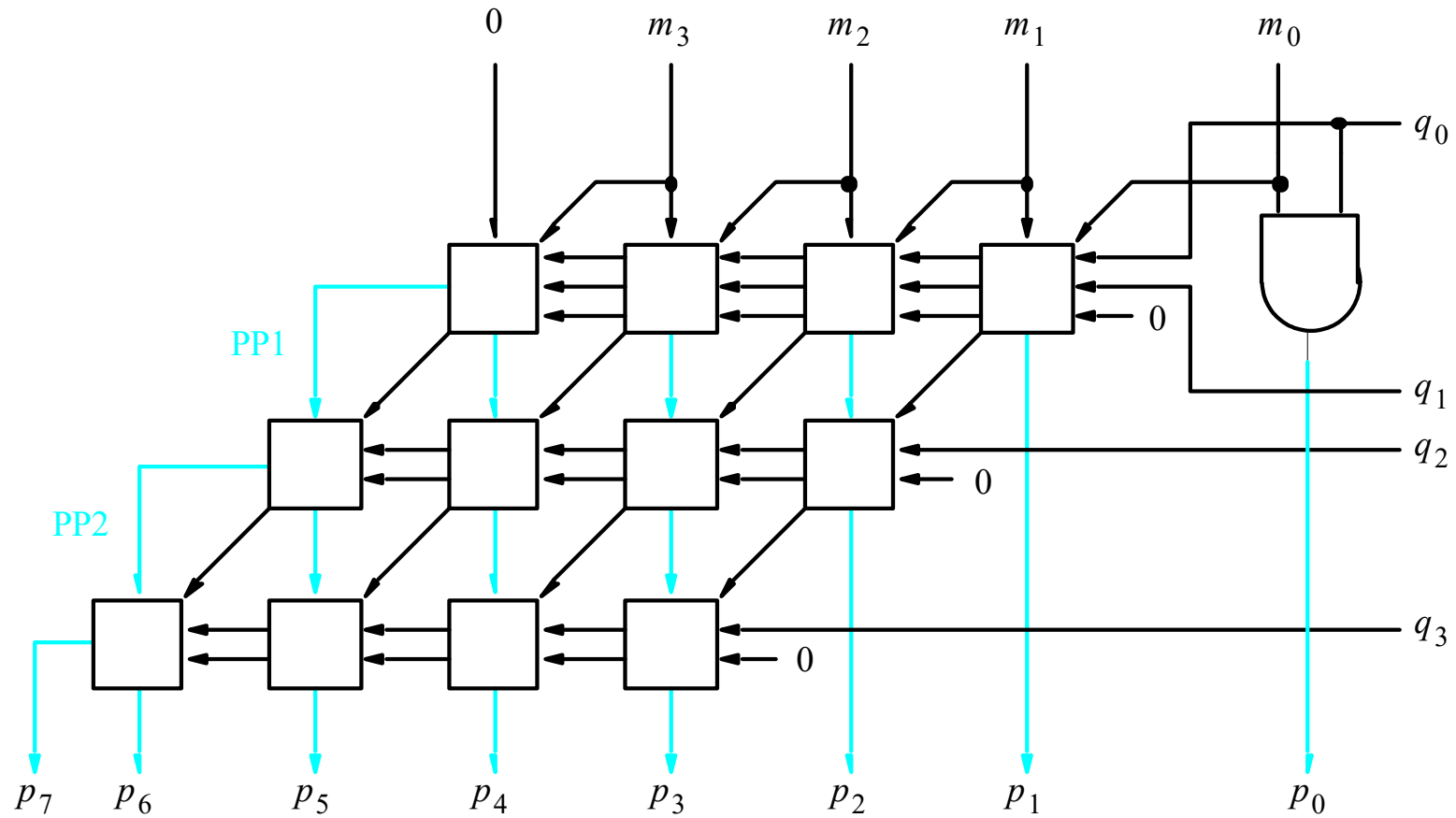


(b) A block in the top row



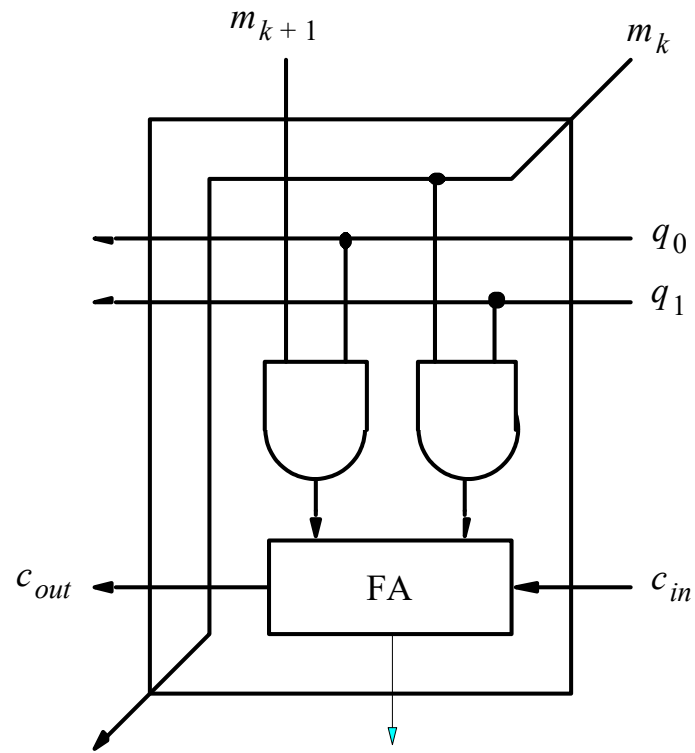
(c) A block in the bottom two rows

# Circuito multiplicador 4 X 4

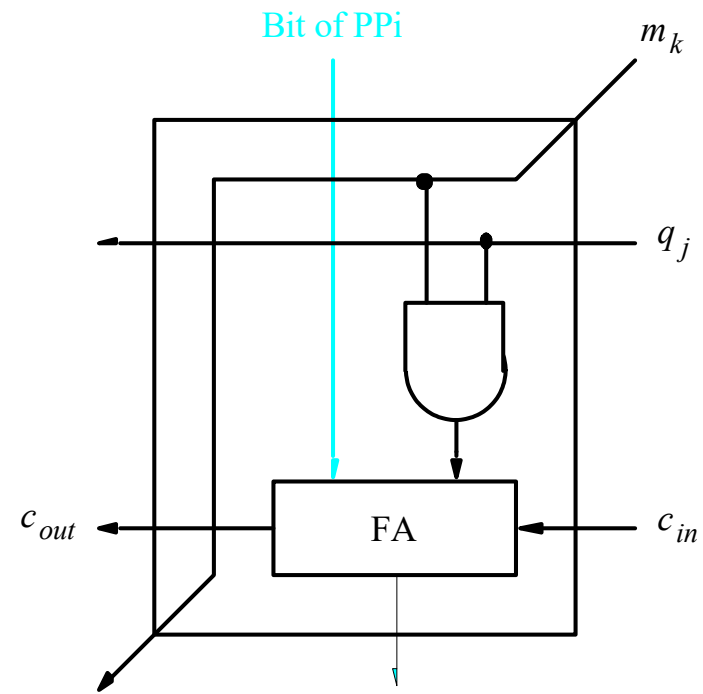


(a) Structure of the circuit

# Circuito multiplicador 4 X 4



(b) A block in the top row



(c) A block in the bottom two rows



# Multiplicação de números sinalizados

Multiplicand M	(+14)	0 1 1 1 0
Multiplier Q	(+11)	x 0 1 0 1 1
Partial product 0		0 0 0 1 1 1 0
		+ 0 0 1 1 1 0
Partial product 1		0 0 1 0 1 0 1
		+ 0 0 0 0 0 0
Partial product 2		0 0 0 1 0 1 0
		+ 0 0 1 1 1 0
Partial product 3		0 0 1 0 0 1 1
		+ 0 0 0 0 0 0
Product P	(+154)	0 0 1 0 0 1 1 0 1 0

(a) Positive multiplicand

## Multiplicação de números sinalizados

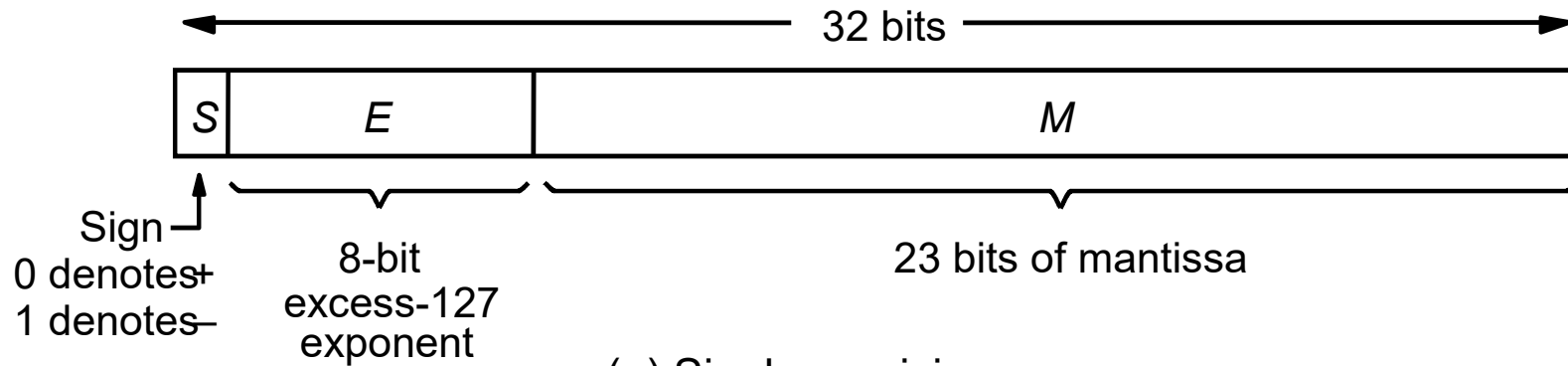
Multiplicand M	(−14)	1 0 0 1 0
Multiplier Q	(+11)	× 0 1 0 1 1
		<hr/>
Partial product 0		1 1 1 0 0 1 0
		+ 1 1 0 0 1 0
		<hr/>
Partial product 1		1 1 0 1 0 1 1
		+ 0 0 0 0 0 0
		<hr/>
Partial product 2		1 1 1 0 1 0 1
		+ 1 1 0 0 1 0
		<hr/>
Partial product 3		1 1 0 1 1 0 0
		+ 0 0 0 0 0 0
		<hr/>
Product P	(−154)	1 1 0 1 1 0 0 1 1 0

(b) Negative multiplicand

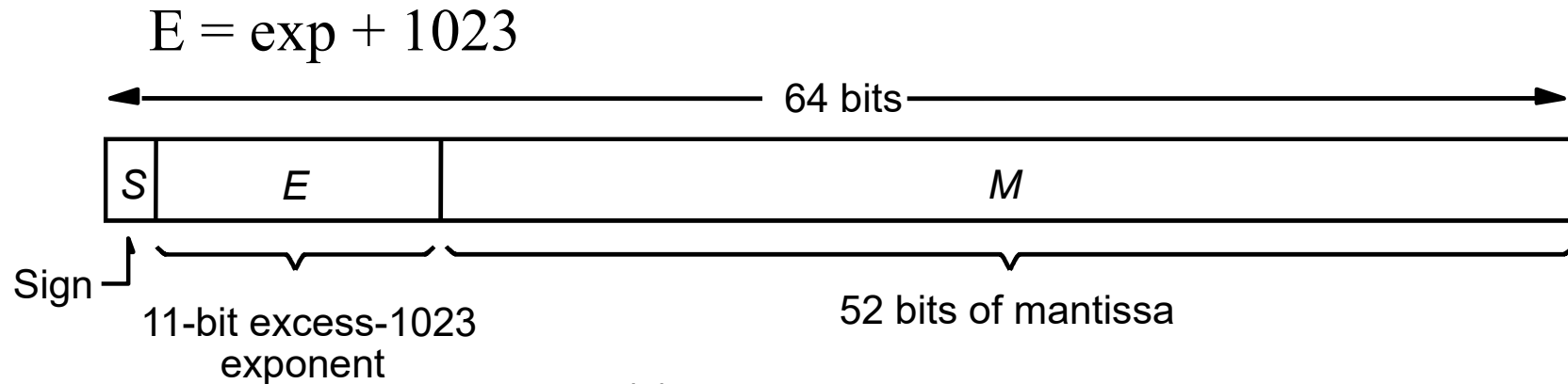
## Padrão IEEE 754 standard número de ponto flutuante

$$N = (-1)^S \times 1, M \times 2^{\text{exp}}$$

$$E = \text{exp} + 127$$



(a) Single precision



(c) Double precision

Representar  $-5,0_{10}$  no padrão IEEE 754

$$-5,0_{10} = -101,0_2 \times 2^0 = -1,0100 \times 2^2$$

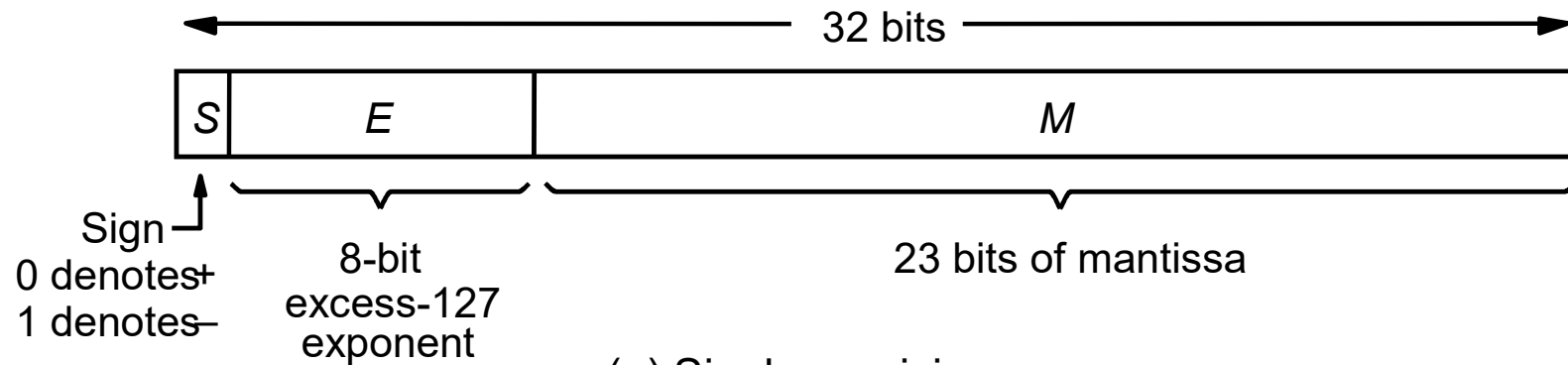
$$N = (-1)^S \times 1, M \times 2^{\text{exp}}$$

$$N = (-1)^1 \times 101,00 \times 2^0 =$$

$$N = (-1)^1 \times 1,0100 \times 2^2 =$$

$$E = 2 + 127 = 129$$

1 10000001 0100000000000000..00



(a) Single precision

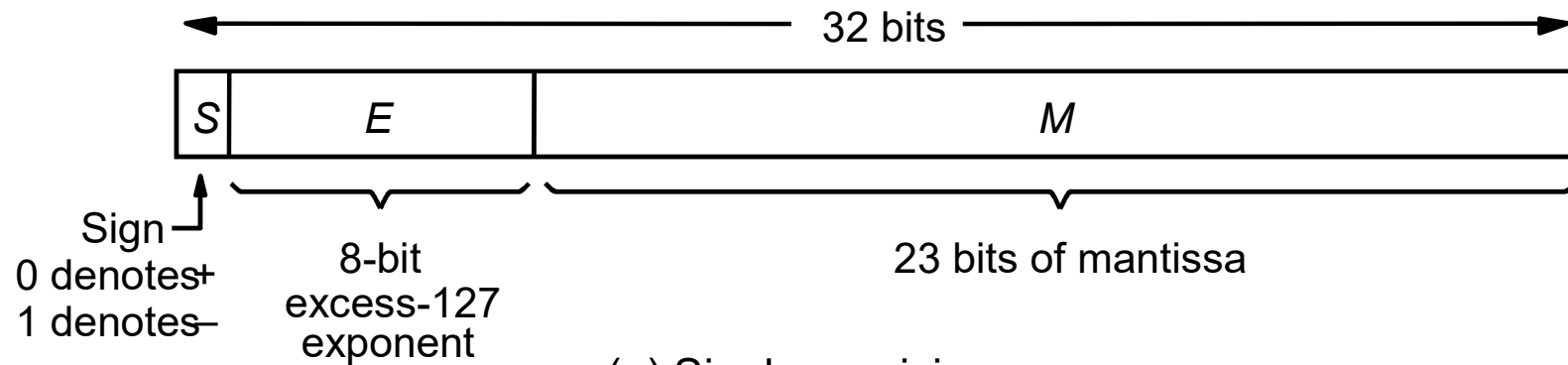
$$N = (-1)^S \times 1, M \times 2^{\text{exp}}$$

Que número é na base 10 o número no padrão IEEE 754

0 10000011 1010000...00

$$N = (-1)^0 \times 1,10100 \times 2^4 = (1 + 0,101) \times 2^4 = 1,625 \times 16 = 26,0$$

$$\text{exp} = 131 - 127 = 4$$



(a) Single precision

## Código BCD – Binary Code Decimal

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

## Adição em BCD

$$\begin{array}{r} X \quad 0011 \quad 3 \\ +Y \quad 0010 \quad +2 \\ \hline Z \quad 0101 \quad 5 \end{array}$$

Não somar 0110

$$\begin{array}{r} X \quad \quad 0111 \quad 7 \\ +Y \quad +0101 \quad +5 \\ \hline Z \quad \quad 1100 \quad 12 \end{array}$$

+ 0110

carry → 10010

S = 2

$$\begin{array}{r} X \quad \quad 1000 \quad 8 \\ +Y \quad +1001 \quad +9 \\ \hline Z \quad \quad 10001 \quad 17 \end{array}$$

+ 0110

carry → 10111

S = 7

$$\begin{array}{r} 1001 \\ +1001 \\ \hline 10010 \end{array}$$

1010

1011

1100

1101

1110

1111



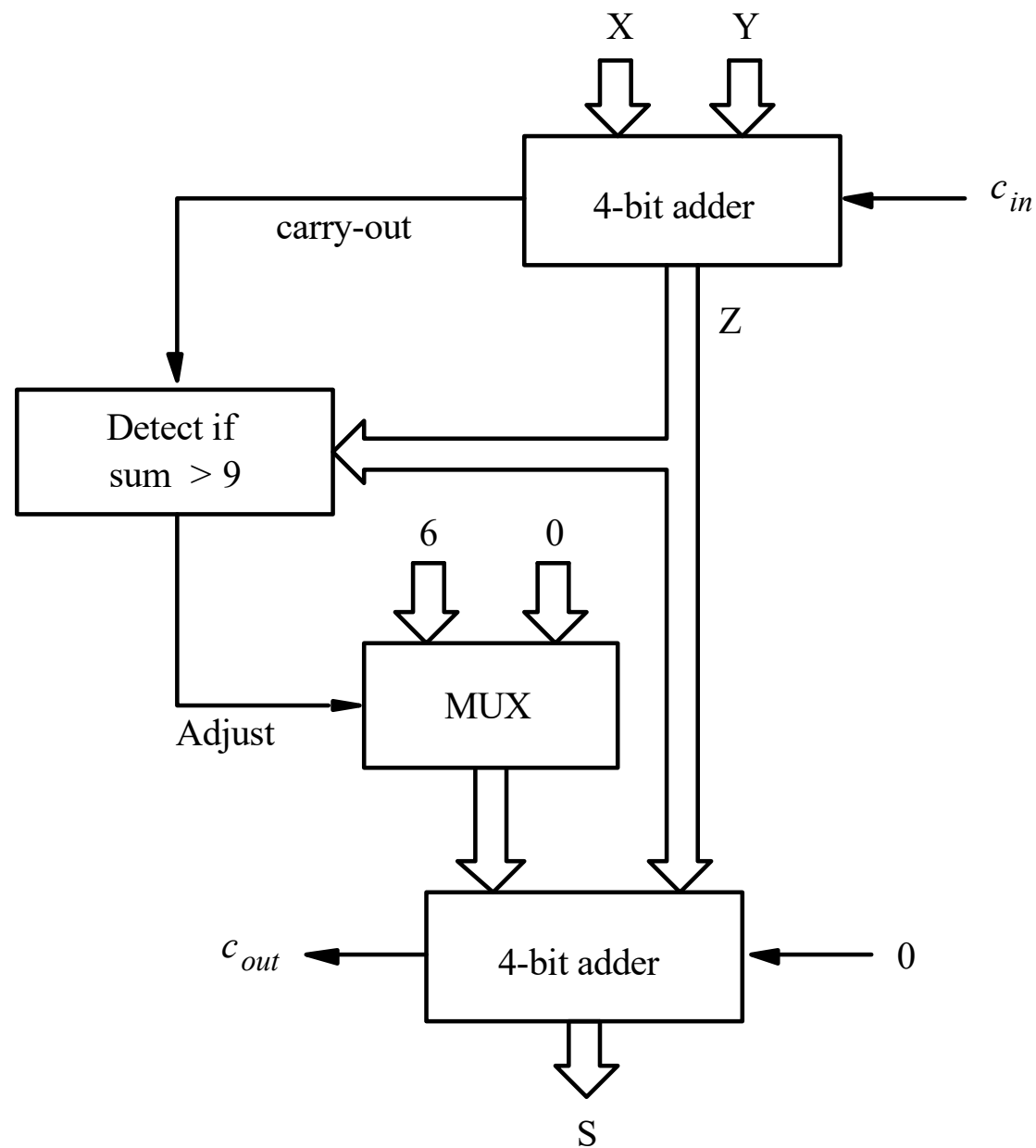
Exercício: Projetar um somador BCD de um dígito (blocos)

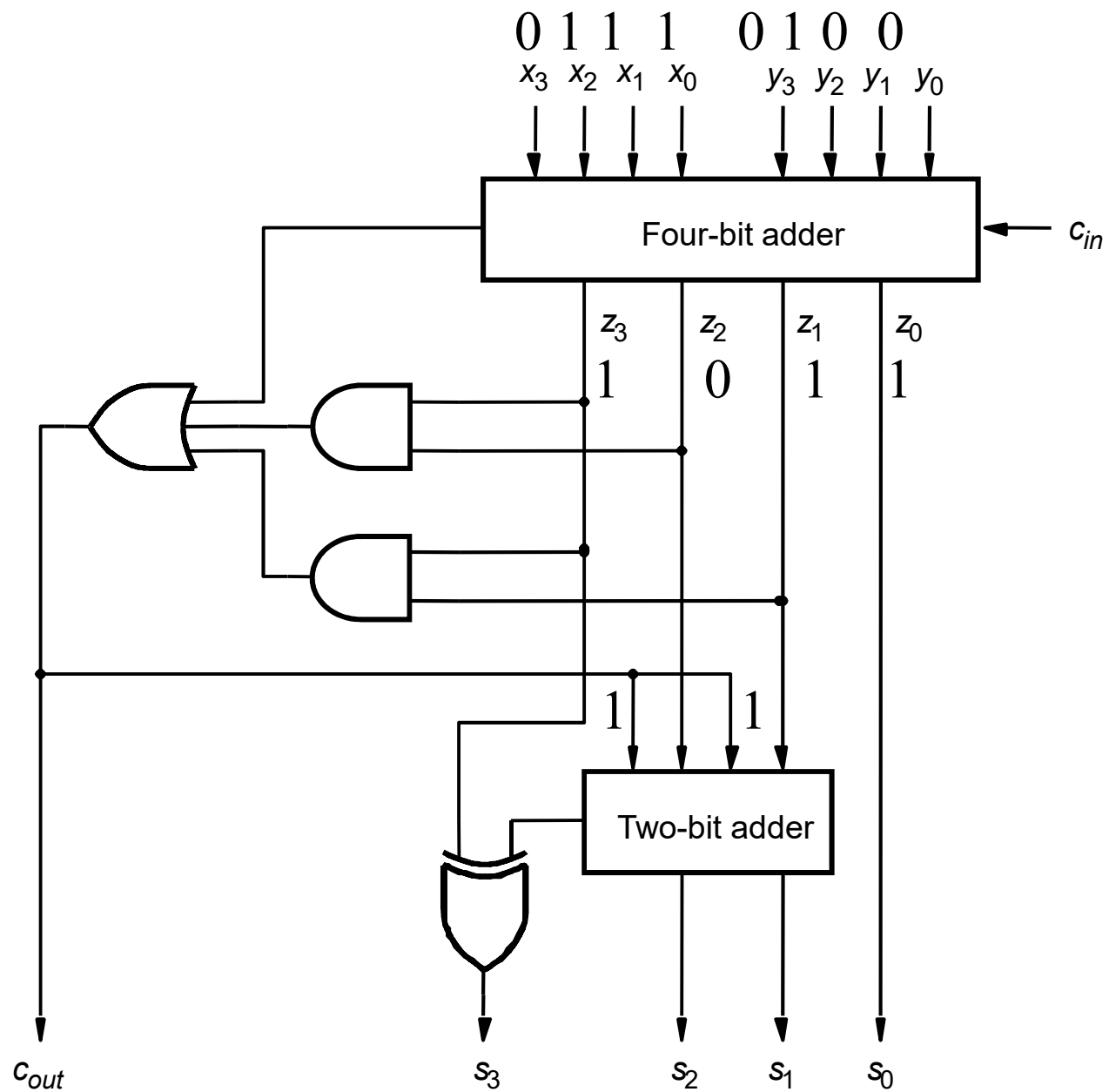
1010  
1011  
1100  
1101  
1110  
1111  
10000  
1.....

$$\begin{array}{r}
 0101 \\
 0110 \\
 \hline
 1011 \\
 0110 \\
 \hline
 1\ 0001
 \end{array}$$

$$\begin{array}{r}
 1000 \\
 1001 \\
 \hline
 10001 \\
 0110 \\
 \hline
 1\ 0111
 \end{array}$$

## Somador BCD de um dígito – diagrama de blocos





Circuito para um somador BCD de um dígito

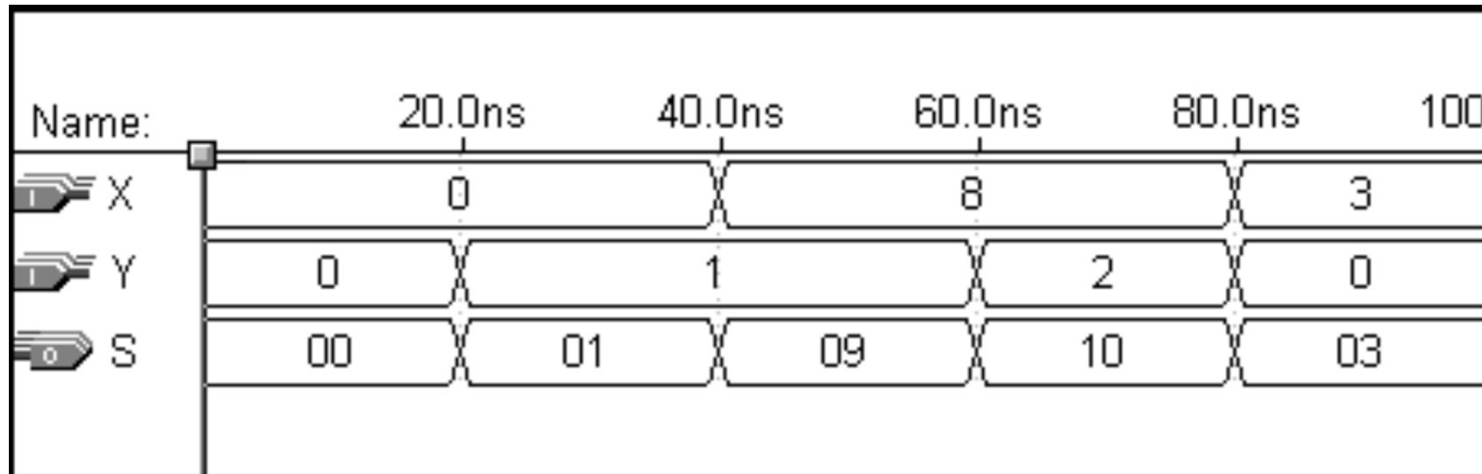
## Somador BCD de um dígito – código VHDL

```
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 ;
```

## Simulação de um somador BCD de um dígito



**Bit  
positions**

**Bit positions 654**

3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

NUL	Null/Idle	SI	Shift in
SOH	Start of header	DLE	Data link escape
STX	Start of text	DC1-DC4	Device control
ETX	End of text	NAK	Negative acknowledgement
EOT	End of transmitted	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmitted block
ACQ	Acknowledgement	CAN	Cancel (error in data)
BEL	Audible signal	EM	End of medium
BS	Back space	SUB	Special sequence
HT	Horizontal tab	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tab	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	DEL	Delete/Idle

Bit positions of code format = 6 5 4 3 2 1 0