Capítulo 2-Projeto Lógico Combinacional H

Profa. Eliete Caldeira

Módulos padrão aritméticos

- Módulos-padrão aritméticos: comparadores, multiplicadores

Comparador de magnitude de 1 bit

- Comparando os bits A e B
  - A = B (*equal*) se:

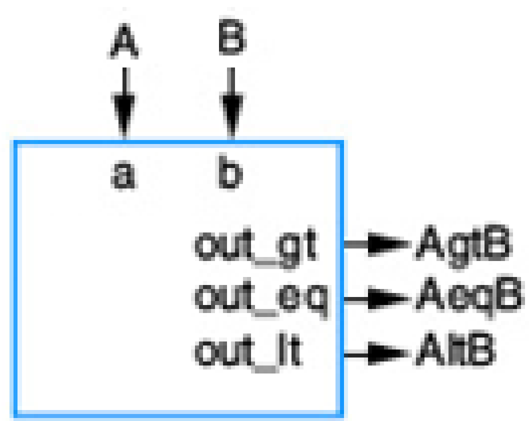
    $$AeqB = A.B + A'.B' = A \text{ xnor } B$$

  - A > B (*greater than*) se :

    $$AgtB = A.B'$$

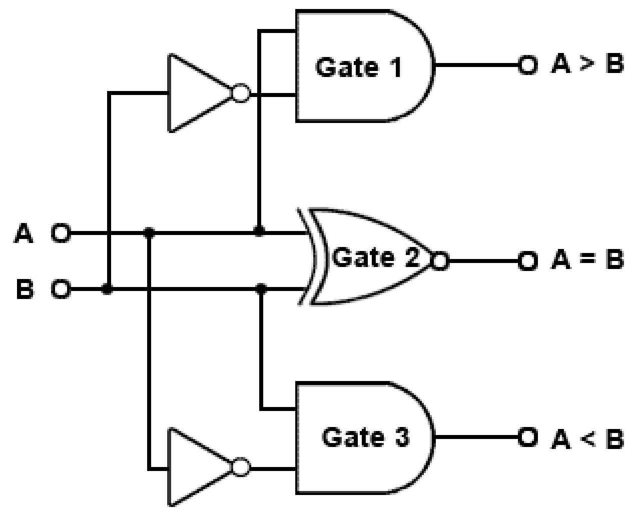
  - A < B (*less than*) se:

    $$AltB = A'.B$$

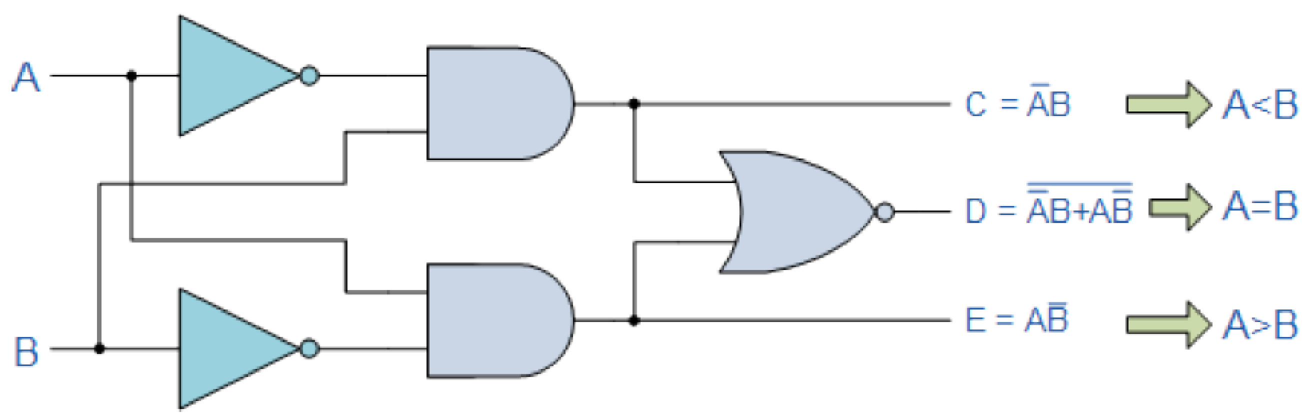


Comparador de magnitude de 1 bit

- Implementação



ou



Comparador de magnitude de 2 bits

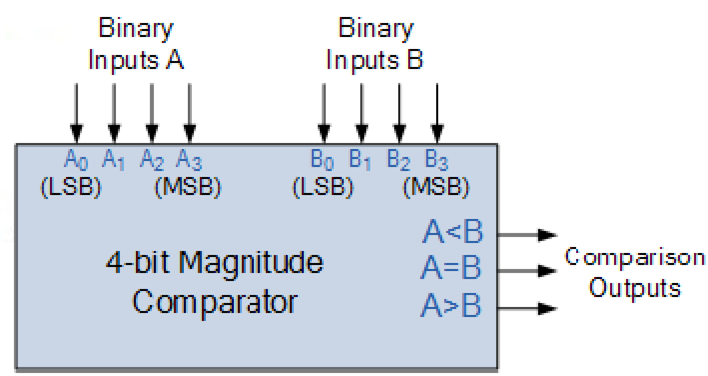
- Considere A = A1A0 e B = B1B0
- A > B se:
  - A1 > B1 ou A1 = B1 e A0 > B0
  - **AgtB = A1.B1' + (A1 xnor B1).A0.B0'**
- A = B se:
  - A1 = B1 e A0 = B0
  - **AeqB = (A1 xnor B1).(A0 xnor B0)**
- A < B se:
  - A1 < B1 ou A1 = B1 e A0 < B0
  - **AltB = A1'.B1 + (A1 xnor B1).A0'.B0**

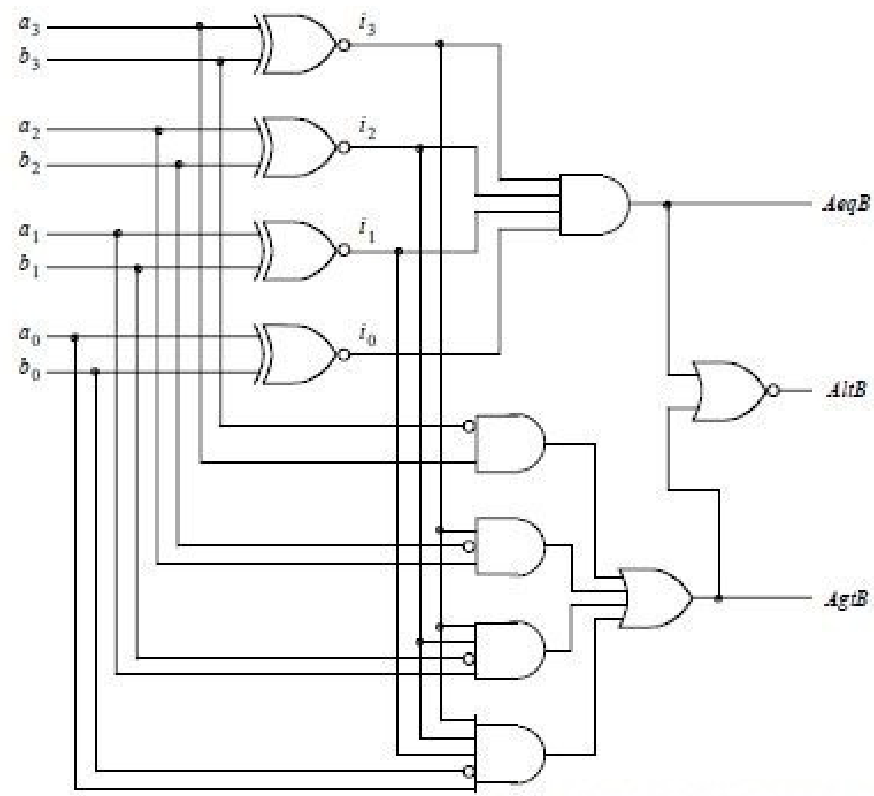
Comparador de magnitude de 4 bits

- Considere A = A3A2A1A0 e B = B3B2B1B0
- A > B se:
  - A3 > B3 ou A3 = B3 e A2 > B2 ou A3 = B3 e A2 = B2 e A1>B1 ou A3 = B3 e A2 = B2 e A1=B1 e A0>B0
  - **AgtB = A3.B3' + (A3 xnor B3).A2.B2'+ (A3 xnor B3).(A2 xnor B2).A1.B1'+ (A3 xnor B3).(A2 xnor B2). (A1 xnor B1).A0.B0'**
- A = B se:
  - A3 = B3 e A2 = B2 e A1 = B1 e A0 = B0
  - **AeqB = (A3 xnor B3).(A2 xnor B2).(A1 xnor B1).(A0 xnor B0)**
- A < B se:
  - A3 < B3 ou A3 = B3 e A2 < B2 ou A3 = B3 e A2 = B2 e A1<B1 ou A3 = B3 e A2 = B2 e A1=B1 e A0<B0
  - **AltB = A3'.B3' + (A3 xnor B3).A2'.B2+ (A3 xnor B3).(A2 xnor B2).A1'.B1+ (A3 xnor B3).(A2 xnor B2). (A1 xnor B1).A0'.B0**

Comparador de magnitude de 4 bits

- Circuito





Comparador de mais bits

- Como comparar números com mais bits?

Comparador de mais bits

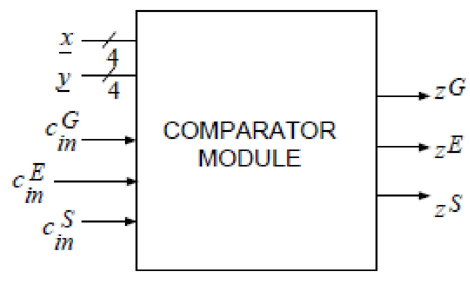
- Como comparar números com mais bits?
- Comparador encadeado, com entradas de bit superior ou de bit inferior

Comparador de magnitude de 4 bits com entradas de bit inferior

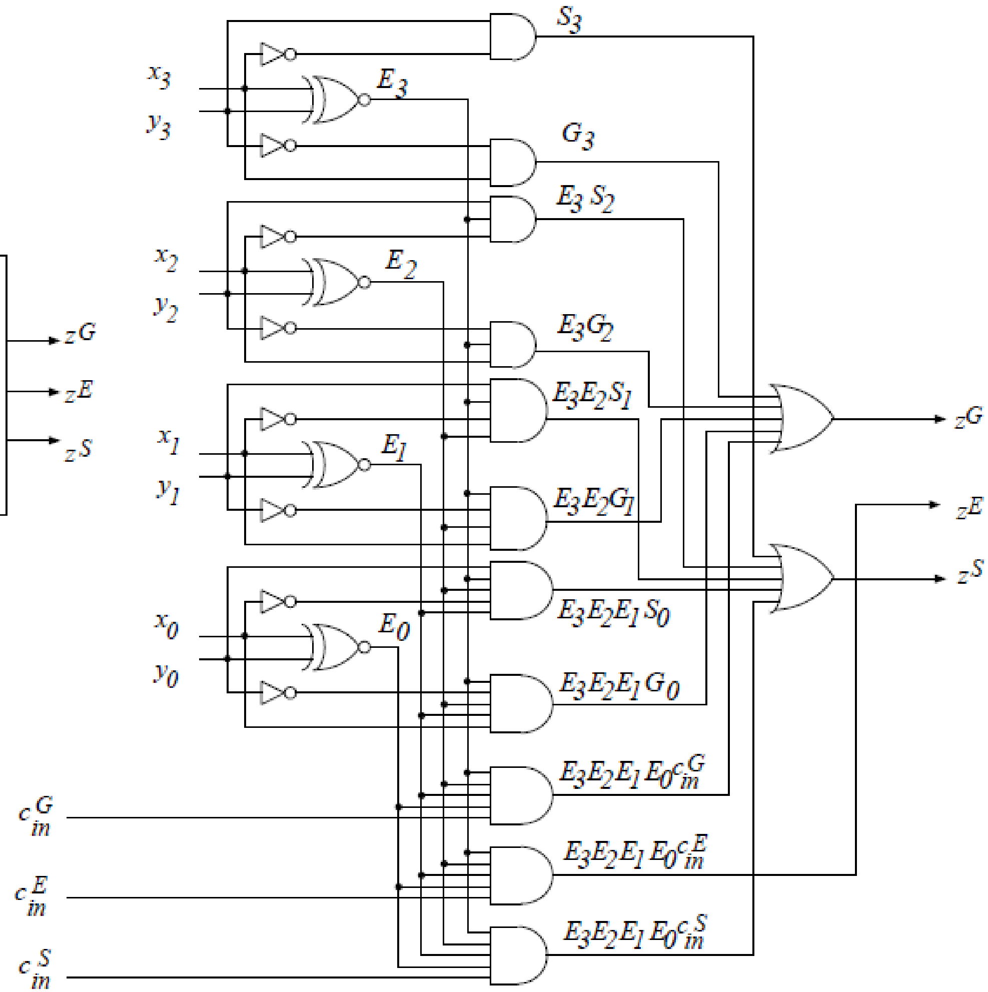
- Considere $x = x_3x_2x_1x_0$ e $y = y_3y_2y_1y_0$
- Para i de 0 a 3 seja
  - $E_i = x_i$ xnor y_i (equal)
  - $G_i = A_i.B_i'$ (greater than)
  - $S_i = A_i'.B_i$ (smaller than)
- E sejam C_{in}^G, C_{in}^E e C_{in}^S entradas de bit inferior
- $z^G = G_3+E_3.G_2+E_3.E_2.G_1+E_3.E_2.E_1.G_0 +E_3.E_2.E_1.E_0.C_{in}^G$
- $z^E = E_3.E_2.E_1.E_0.C_{in}^E$
- $z^S = S_3+E_3.S_2+E_3.E_2.S_1+E_3.E_2.E_1.S_0 +E_3.E_2.E_1.E_0.C_{in}^S$

Comparador de magnitude de 4 bits

- Circuito



(a)

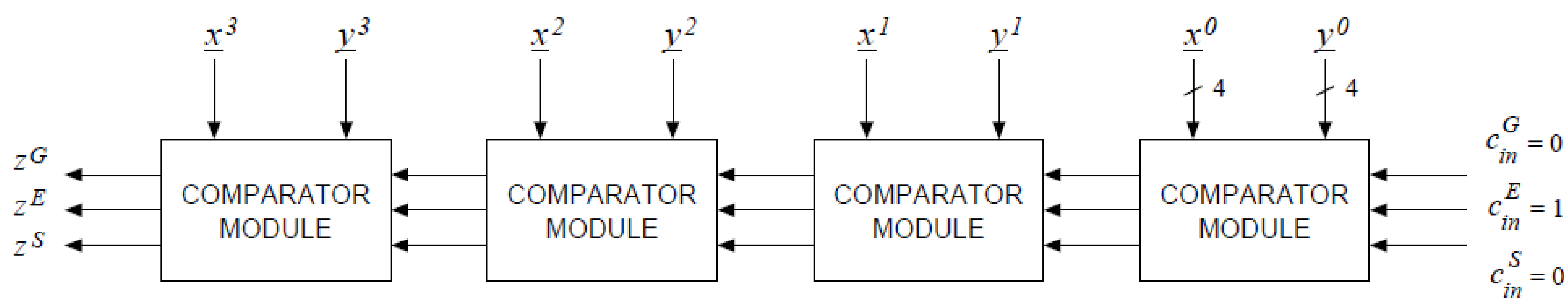


$$
\begin{aligned}
S_i &= x_i' y_i \\
E_i &= (x_i \oplus y_i)', \quad i = 0, \ldots, 3 \\
G_i &= x_i y_i'
\end{aligned}
$$

$$
\begin{aligned}
z^G &= G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0 + E_3 E_2 E_1 E_0 c_{in}^G \\
z^E &= E_3 E_2 E_1 E_0 c_{in}^E \\
z^S &= S_3 + E_3 S_2 + E_3 E_2 S_1 + E_3 E_2 E_1 S_0 + E_3 E_2 E_1 E_0 c_{in}^S
\end{aligned}
$$

Comparadores de magnitude de 4 bits encadeados

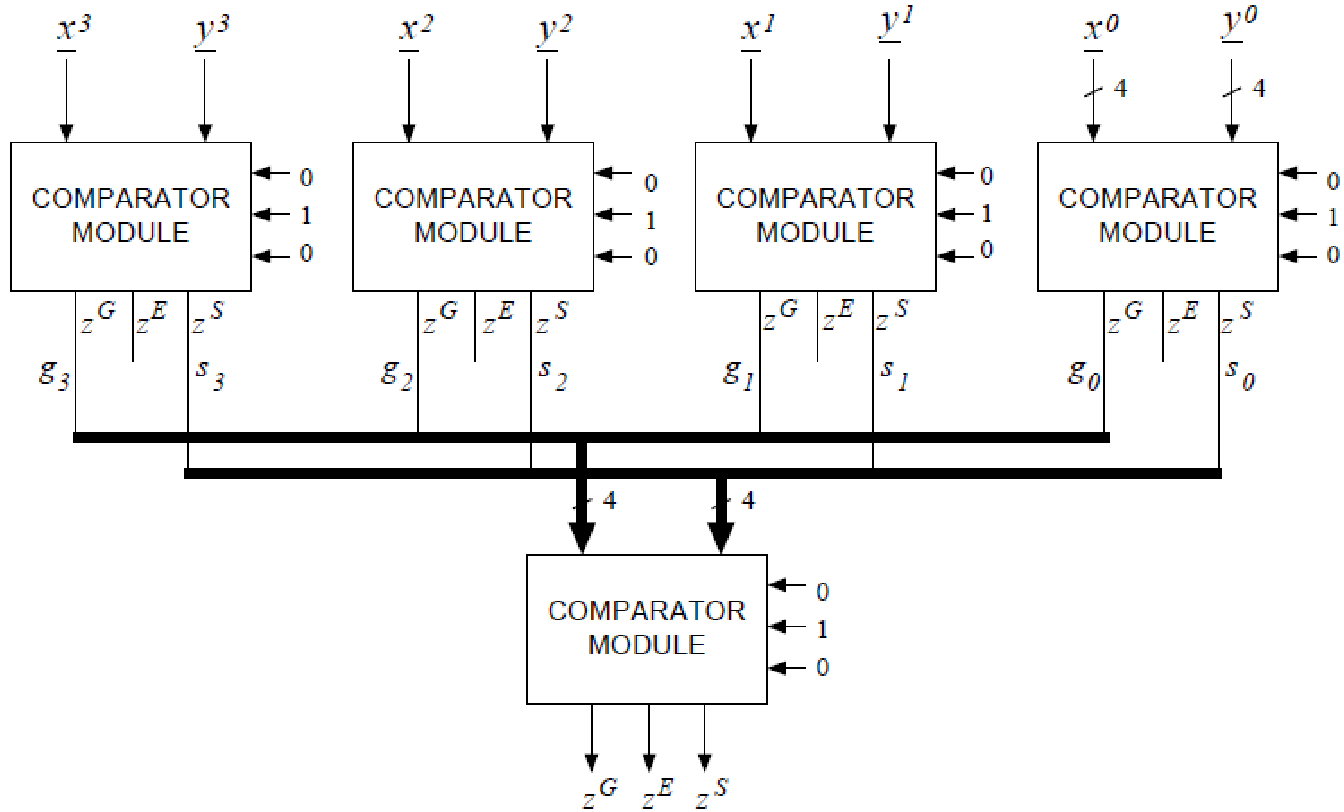
- $\underline{x}^3 = x_{15}x_{14}x_{13}x_{12}$ $\underline{x}^2 = x_{11}x_{10}x_9x_8$ $\underline{x}^1 = x_7x_6x_5x_4$ $\underline{x}^0 = x_3x_2x_1x_0$



- A informação dos bits inferiores é propagada entre os módulos gerando atraso

Comparadores de magnitude de 16 bits em árvore

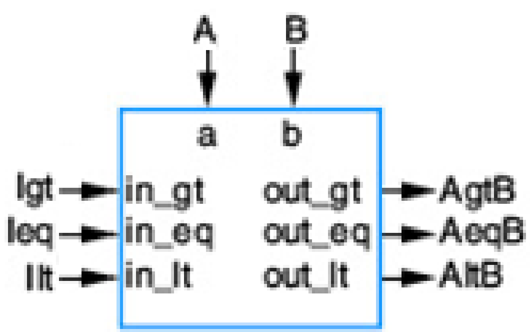
- $\underline{x}^3 = x_{15}x_{14}x_{13}x_{12}$ $\underline{x}^2 = x_{11}x_{10}x_9x_8$ $\underline{x}^1 = x_7x_6x_5x_4$ $\underline{x}^0 = x_3x_2x_1x_0$



- Reduz atraso a dois módulos

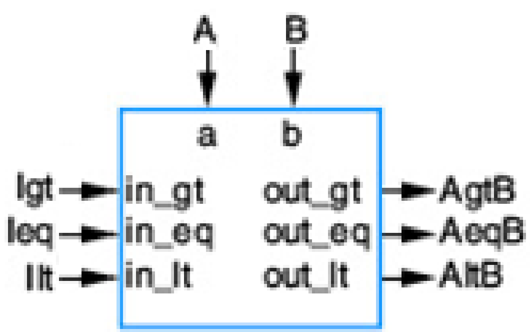
Comparador de magnitude de 4 bits com entradas de bit superior

- Se as entradas vêm de bit superior, como devem ser as equações do módulo?



Comparador de magnitude de 4 bits com entradas de bit superior

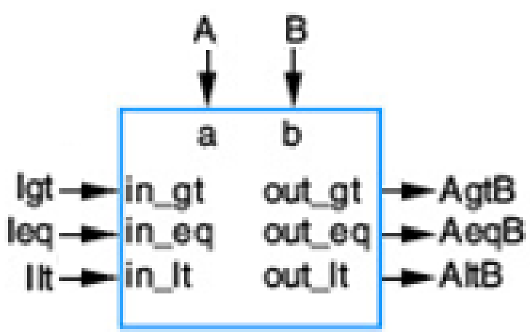
- Se as entradas vêm de bit superior, como devem ser as equações do módulo?



- AgtB = Igt ou A>B
  = Igt + A.B'
- AeqB = Ieq e A = B
  = Ieq.(A xnor B)
- AltB = Igt ou A>B
  = Ilt + A'.B

Comparador de magnitude de 4 bits com entradas de bit superior

- Se as entradas vêm de bit superior, como devem ser as equações do módulo?



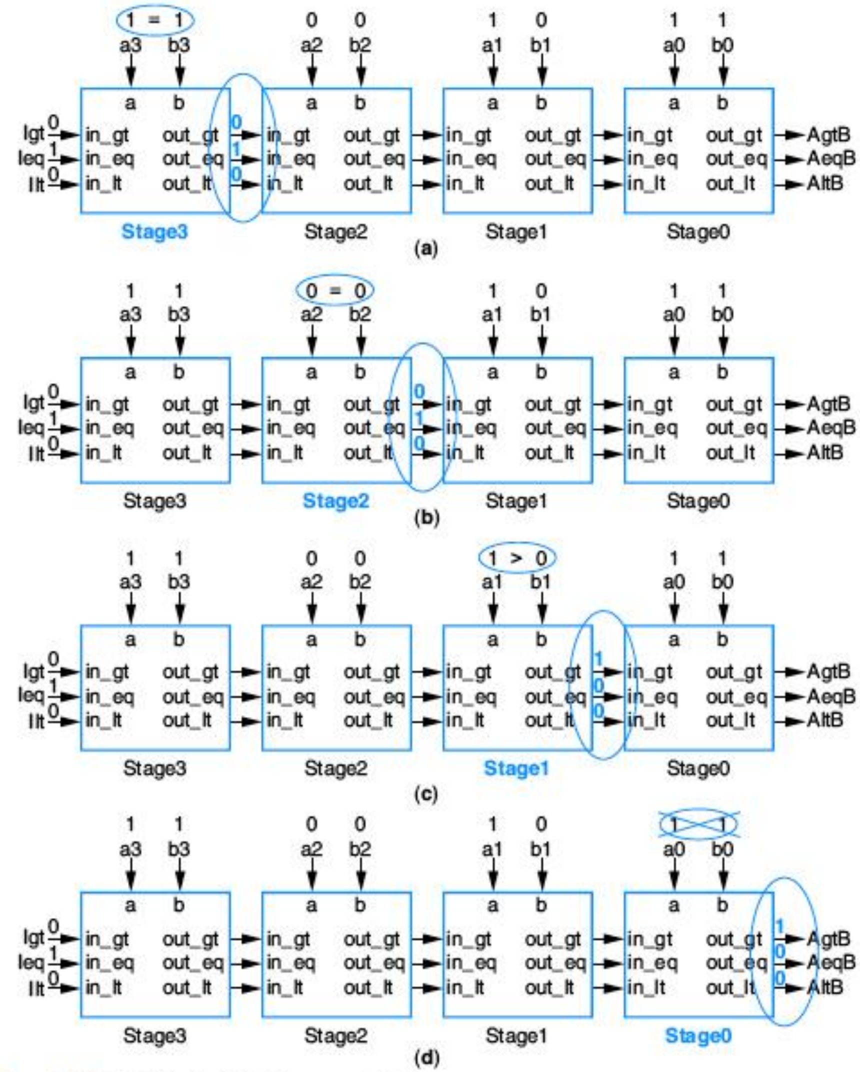


Figure 4.45 The "rippling" within a magnitude comparator.

Comparador de igualdade

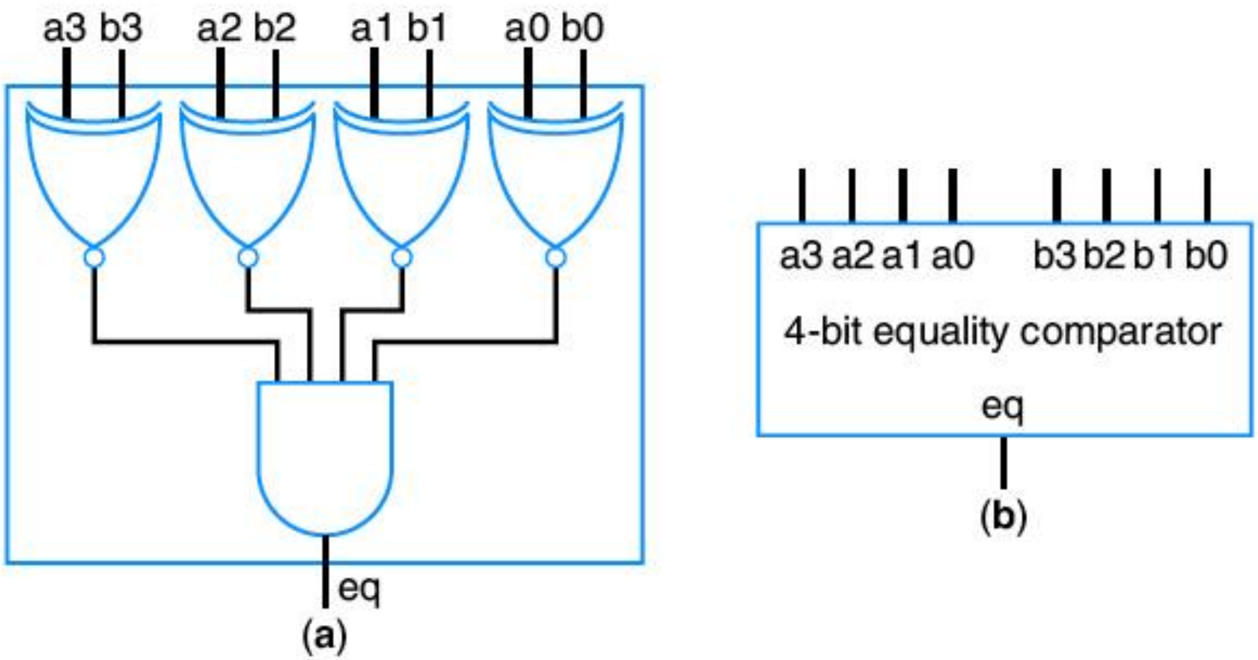


Figure 4.43 Equality comparator: (a) internal design, and (b) block symbol.

Multiplicadores

- Um multiplicador NxN é um componente de bloco operacional que:
- Multiplica dois números binários de N bits, A (o multiplicando) e B (o multiplicador), produzindo um resultado na saída de (N+N) bits

Multiplicadores

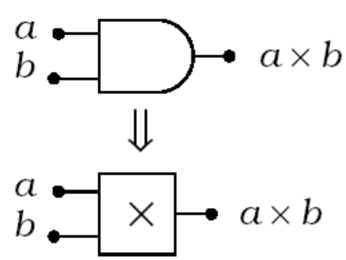
- Operação feita à mão:

```
    0110    (o número superior é chamado de multiplicando)
    0011    (o número inferior é chamado de multiplicador)
    ----    (cada uma das linhas abaixo é chamada de produto parcial)
    0110    (porque o bit mais à direita do multiplicador é 1, e 0110*1=0110)
   0110     (porque o segundo bit do multiplicador é 1, e 0110*1=0110)
  0000      (porque o terceiro bit do multiplicador é 0, e 0110*0=0000)
+0000       (porque o bit mais à esquerda do multiplicador é 0, e 0110*0=0000)
--------
00010010    (o produto é a soma de todos os produtos parciais: 18, que é 6*3)
```

Multiplicadores

- Usando variáveis para representar os bits dos operadores

```
                     a3   a2   a1   a0
                   x b3   b2   b1   b0
-----------------------------------
                     b0a3 b0a2 b0a1 b0a0     (pp1)
                b1a3 b1a2 b1a1 b1a0  0       (pp2)
           b2a3 b2a2 b2a1 b2a0  0    0       (pp3)
+ b3a3 b3a2 b3a1 b3a    0    0    0          (pp4)
-----------------------------------
p7 p6  p5   p4   p3   p2   p1   p0
```



a	b	$a \times b$
0	0	0
0	1	0
1	0	0
1	1	1

Multiplicadores

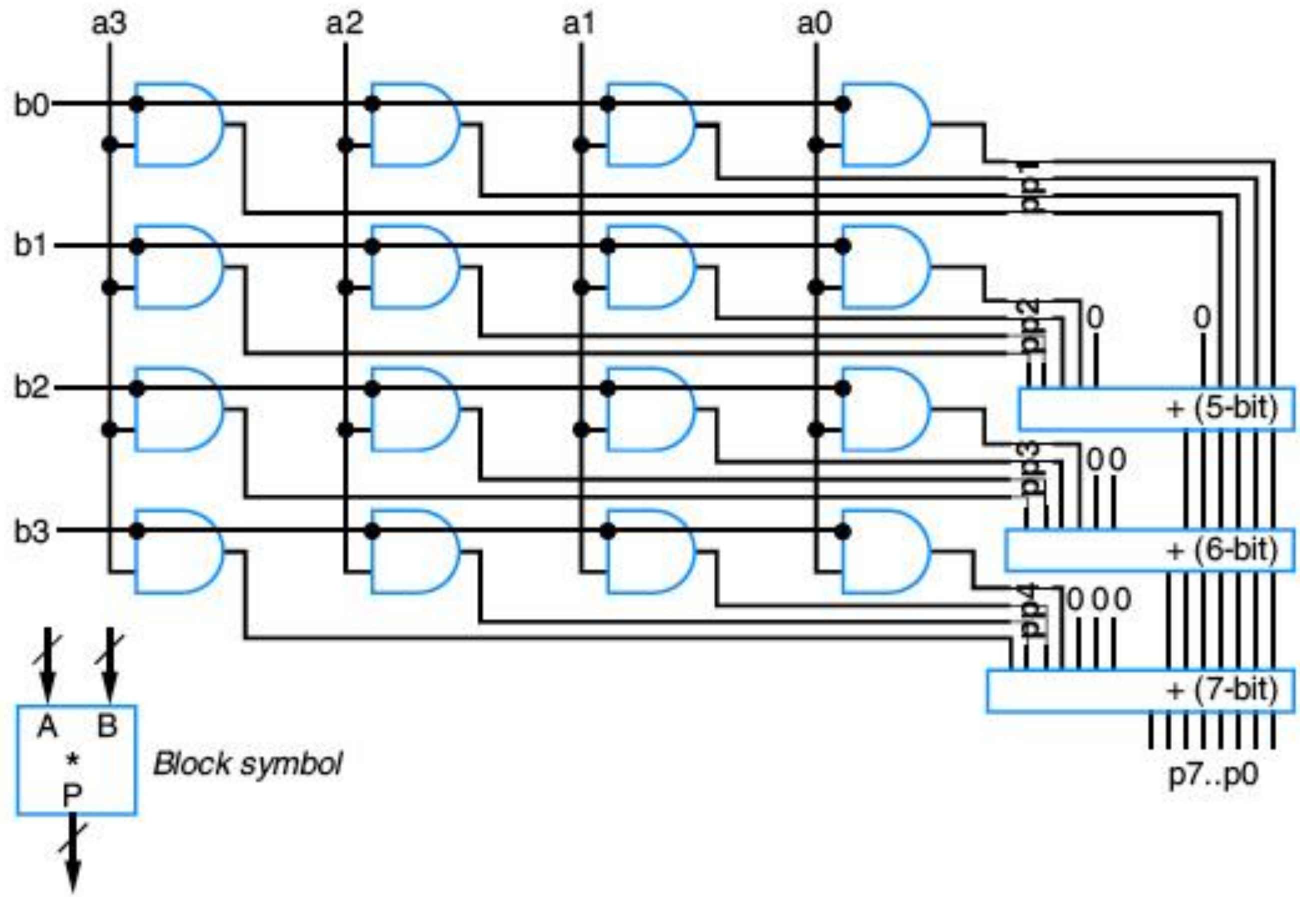


Figure 4.63 Internal design of a 4-bit by 4-bit array-style multiplier.

Multiplicadores

- Outra implementação...

Multiplicadores

- Se x tem n bits e y tem m bits:
  - $0 \leq x \leq 2^n-1$ (multiplicando)
  - $0 \leq y \leq 2^m-1$ (multiplicador)
  - $0 \leq z \leq (2^n-1)(2^m-1)$ (produto)
- A função em alto nível é:
  - $z = x.y$

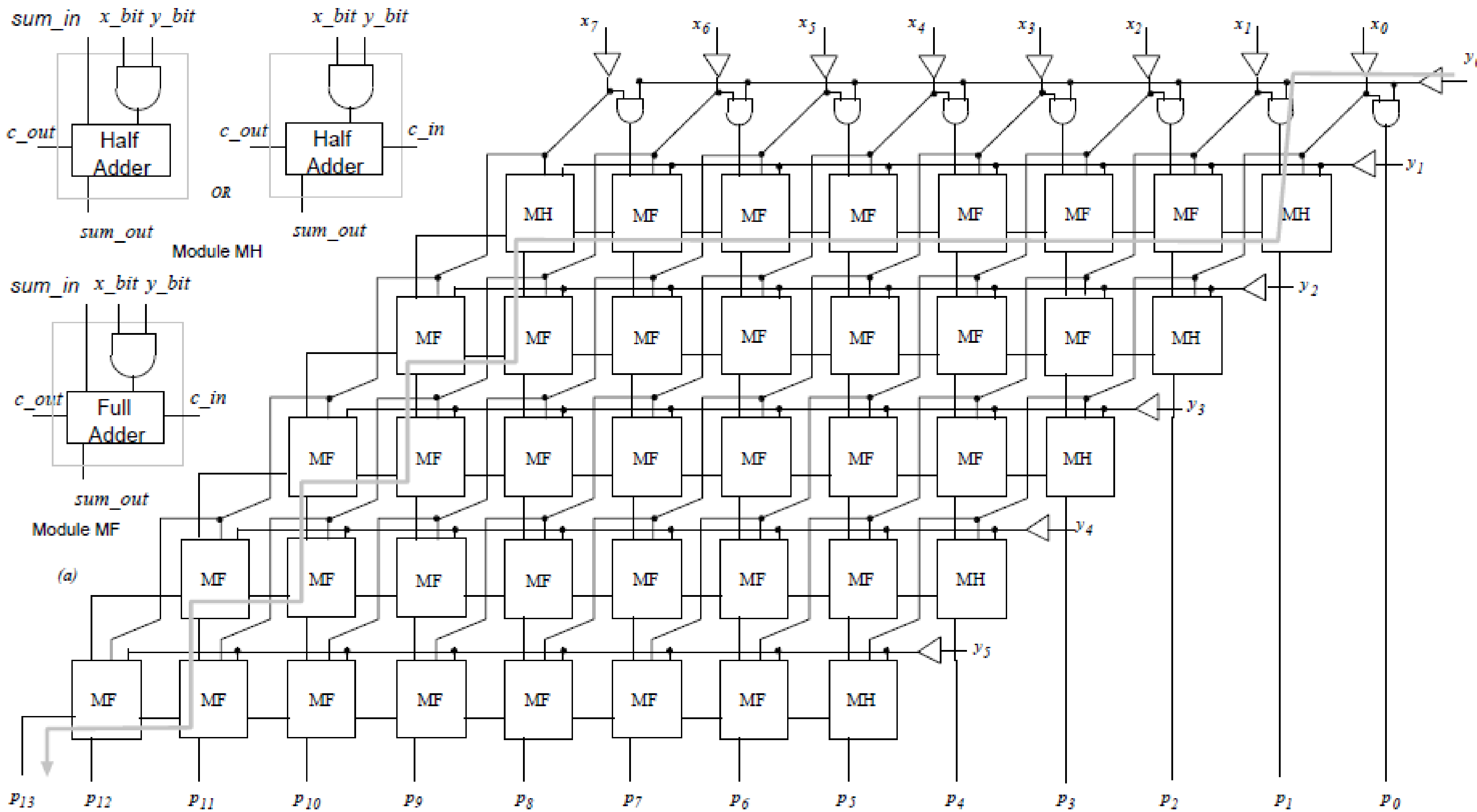
Multiplicadores

- Multiplicar cada bit de y por x resulta em

$$
\begin{array}{cccccccccccccc}
 & & & & & & x_7y_0 & x_6y_0 & x_5y_0 & x_4y_0 & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 \\
 & & & & & x_7y_1 & x_6y_1 & x_5y_1 & x_4y_1 & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & & & & x_7y_2 & x_6y_2 & x_5y_2 & x_4y_2 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & \\
 & & & x_7y_3 & x_6y_3 & x_5y_3 & x_4y_3 & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & \\
 & & x_7y_4 & x_6y_4 & x_5y_4 & x_4y_4 & x_3y_4 & x_2y_4 & x_1y_4 & x_0y_4 & & & & \\
 & x_7y_5 & x_6y_5 & x_5y_5 & x_4y_5 & x_3y_5 & x_2y_5 & x_1y_5 & x_0y_5 & & & & &
\end{array}
$$

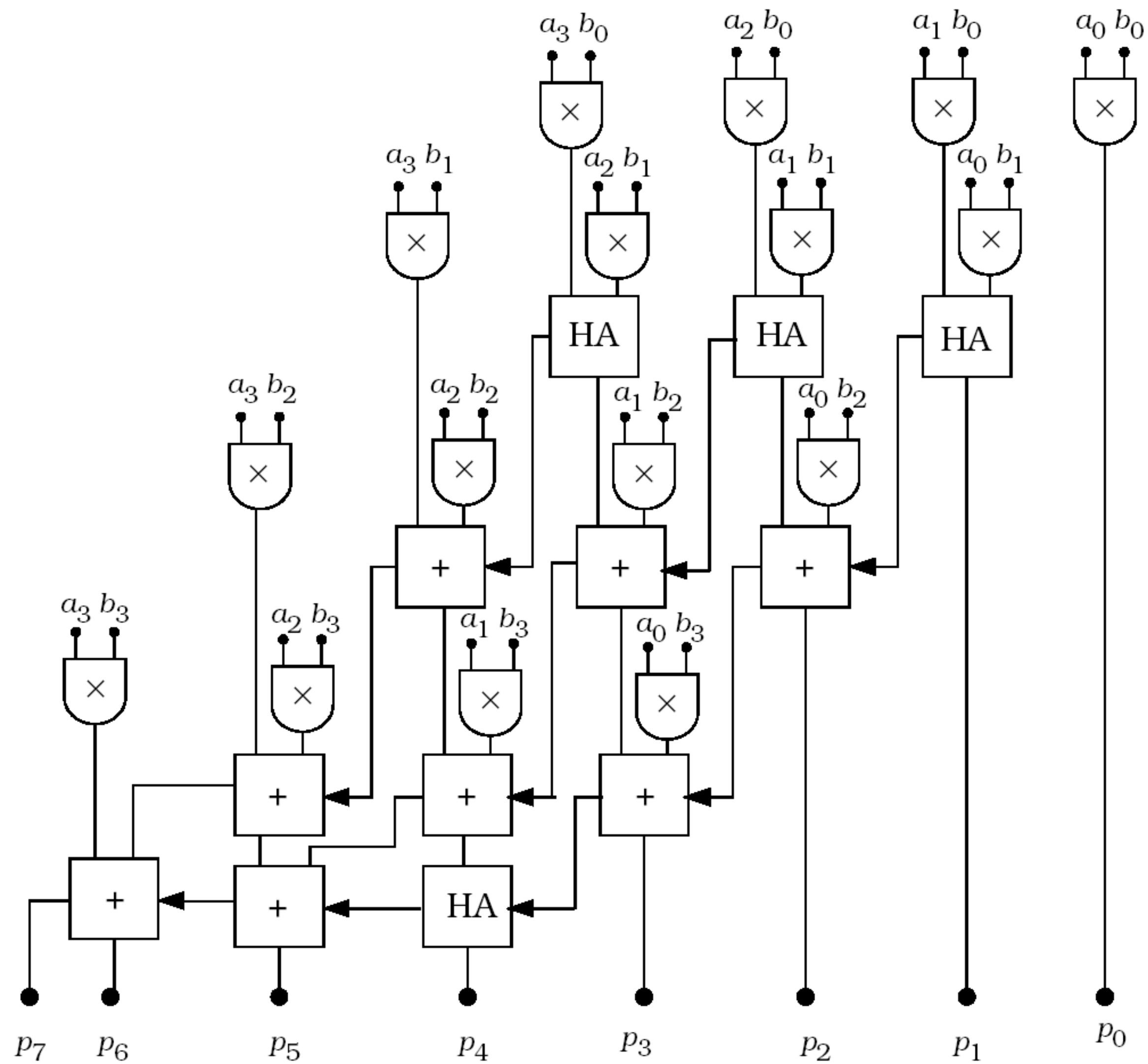
  onde cada operação é um AND
- Após é preciso somar usando somadores binários

Multiplicador com carry propagado



Multiplicador com atraso reduzido

- 4x4 bits



Para ser continuado...