INF01 118

Técnicas Digitais para Computação

Circuitos Aritméticos

Somadores e Subtratores

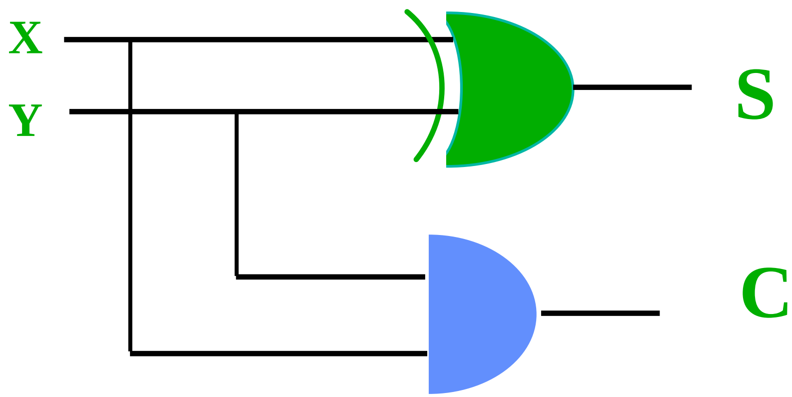
Aula 12

1. Meio Somador ou *Half-Adder* (soma 2 bits)

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

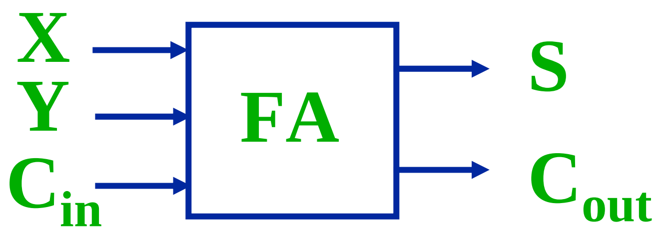
$$S = \overline{X}Y + X\overline{Y} = X \oplus Y$$

$$C = X \cdot Y$$



2. Somador Completo ou *Full-Adder* (soma 3 bits)

X	Y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$S = \bar{X}\bar{Y}C_{in} + \bar{X}Y\bar{C}_{in} + X\bar{Y}\bar{C}_{in} + XYC_{in}$$

$$C_{out} = \bar{X}YC_{in} + X\bar{Y}C_{in} + XY\bar{C}_{in} + XYC_{in}$$

S

$X \backslash YC_{in}$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

- não há aparentemente nenhuma minimização a fazer
- no entanto $S = X \oplus Y \oplus C_{in}$
- XOR é comutativo e associativo

C_{out}: Solução 1

$X \backslash YC_{in}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_{out} = XY + XC_{in} + YC_{in}$$
$$= XY + C_{in}\,(X+Y)$$

C_{out} : Solução 2

X \ YC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_{out} = XY + C_{in}\,(X \oplus Y)$$

solução é preferível porque usa XOR também existente na expressão de S

- Para comprovar que as 2 soluções são equivalentes

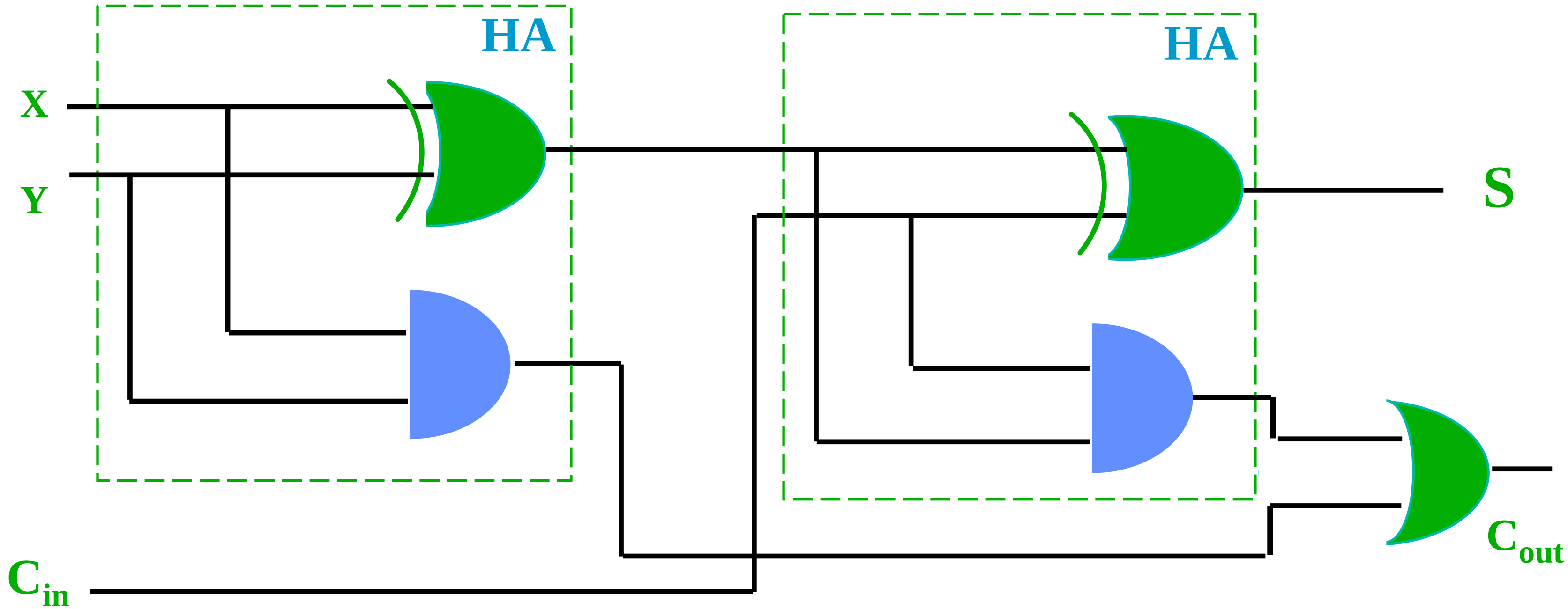
$$C_{out} = XY + \boxed{C_{in}\,(X \oplus Y)}$$

- não é =1 se X=1 e Y=1, mas este caso já é coberto pelo 1º termo
- pode-se portanto reduzir X+Y para X⊕Y

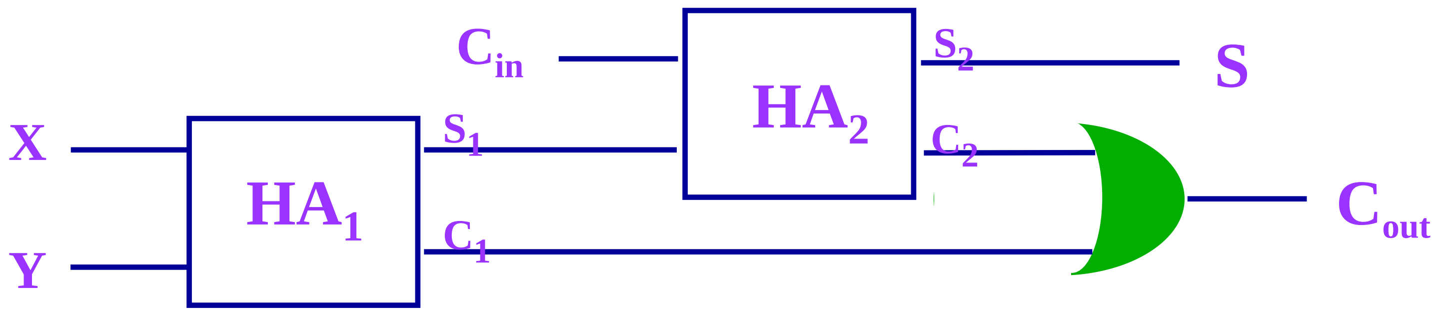
$$C_{out} = XY + \boxed{C_{in}\,(X+Y)}$$

igual a 1 se X=1, ou
Y=1, ou
X=1 e Y=1

Circuito obtido a partir das expressões para S e C_{out}



Se reconhece dois *Half-Adders* (HA's)



$S_1 = X \oplus Y$

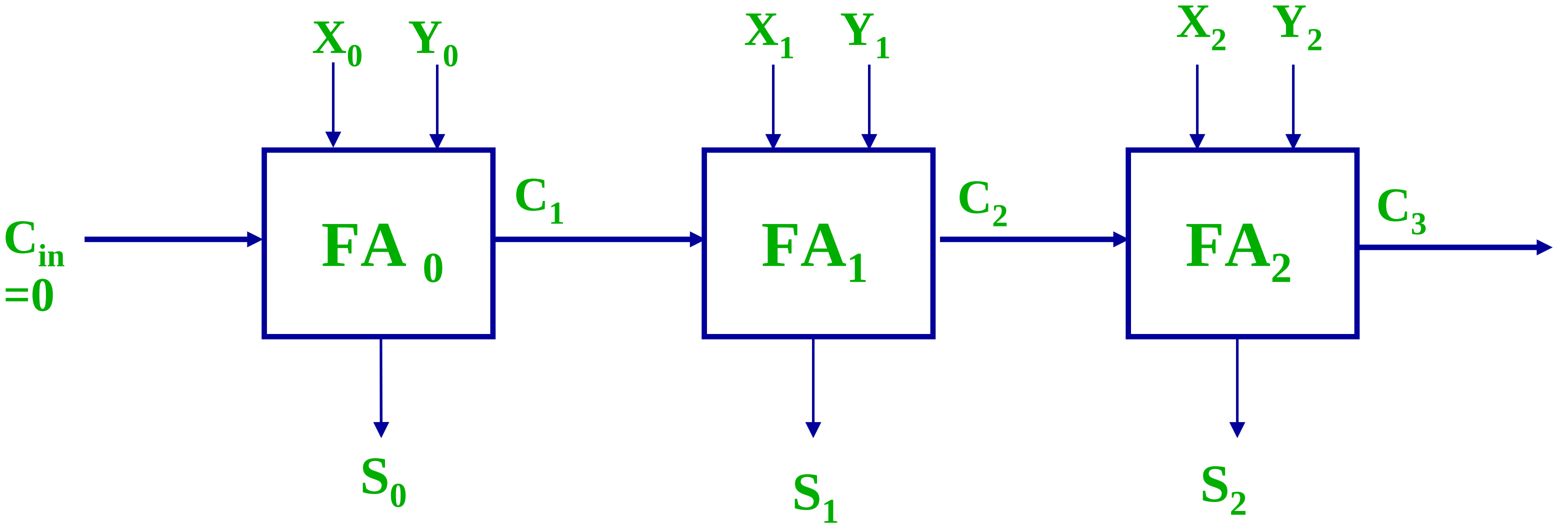
$C_1 = X \, . \, Y$

$S = S_2 = S_1 \oplus C_{in}$

$C_2 = S_1 \, . \, C_{in}$

$C_{out} = C_1 + C_2$

3. Somador de N Bits (*Ripple Carry Adder*)



4. Subtratores

Meio Subtrator (X – Y)

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

D = Diferença
B = Borrow

$$D = X \oplus Y$$
$$B = \overline{X} \cdot Y$$

Subtrator Completo : X - Y

X	Y	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = X \oplus Y \oplus B_{in}$$

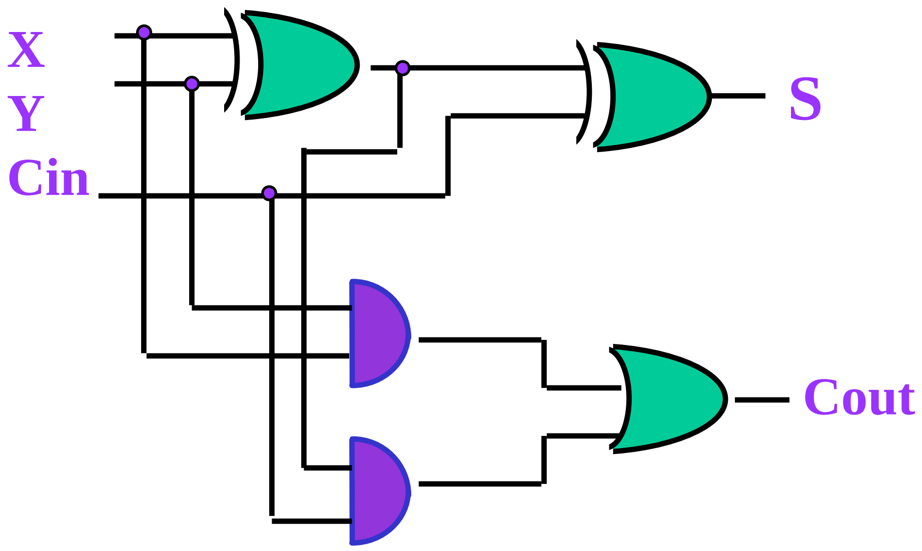
$$B_{out} = \bar{X}\bar{Y}B_{in} + \bar{X}Y\bar{B}_{in} + \bar{X}YB_{in} + XYB_{in}$$

X \ YB_{in}	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$$B_{out} = \bar{X}Y + \bar{X}B_{in} + YB_{in}$$
$$= \bar{X}Y + B_{in}\,(\bar{X} + Y)$$

5. Somador/Subtrator

Somador Completo



$S = X \oplus Y \oplus C_{in}$

$C_{out} = XY + C_{in}\,(X \oplus Y)$

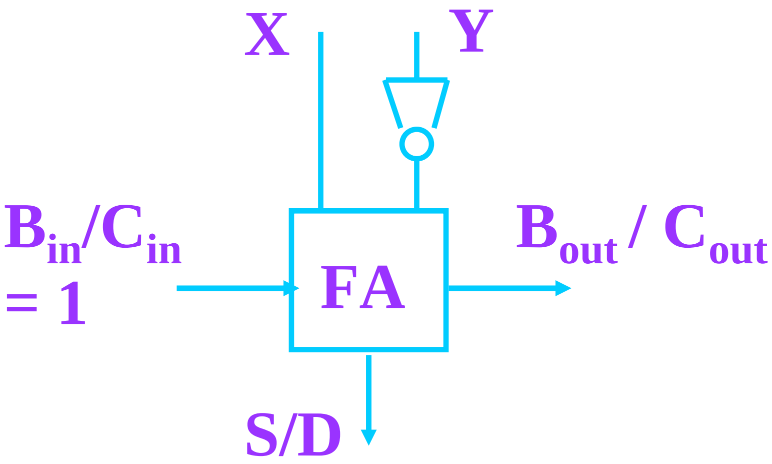
Subtrator Completo

$D = X \oplus Y \oplus B_{in}$

$B_{out} = \overline{X}Y + \overline{X}B_{in} + YB_{in} = \overline{X}Y + B_{in}\,(\overline{X}+Y)$

Pode-se fazer um subtrator usando-se um FA (Full Adder) com:

- **entrada Y invertida**
- **$C_{in} = 1$**

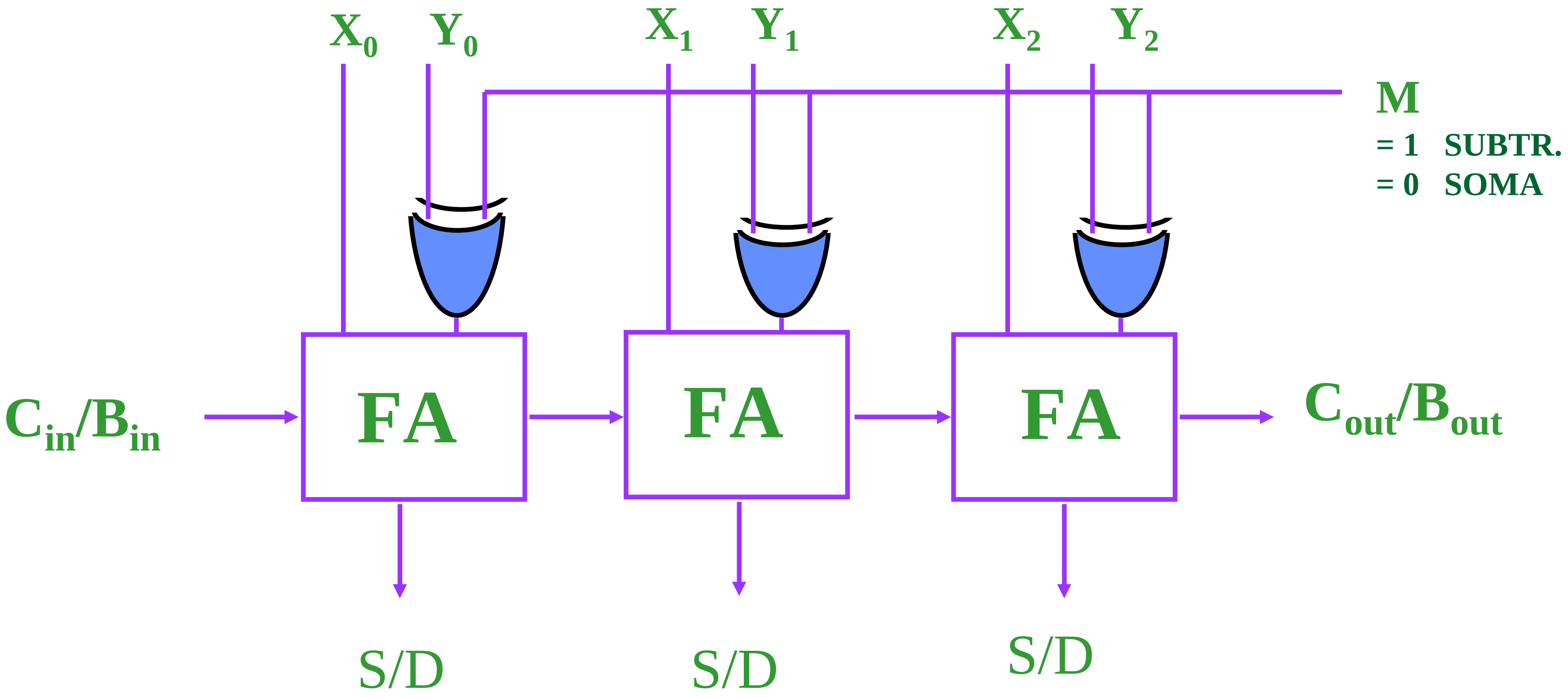


Isto corresponde a

$$X + \overline{Y} + 1 = X - Y$$

2´s de Y

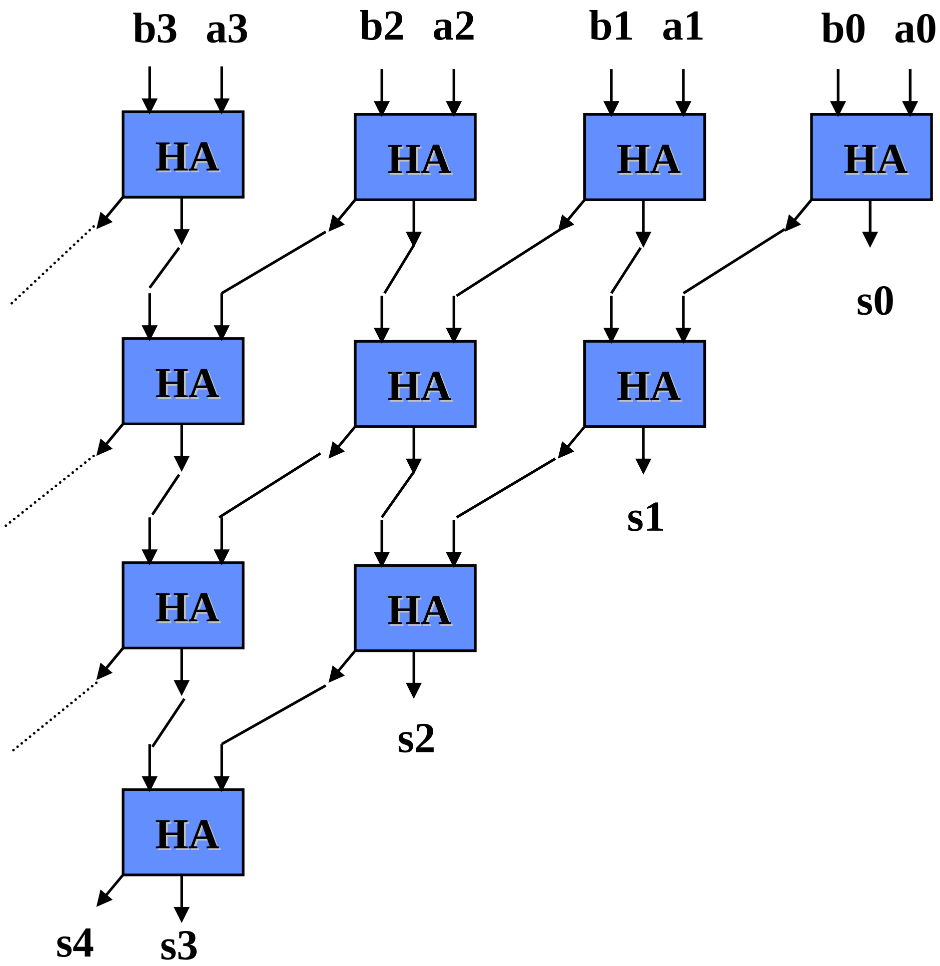
Somador / Subtrator



6. Somador usando apenas Meio-Somadores (HAs)

(A + B = S)

A	⇒	a3	a2	a1	a0
B	⇒	b3	b2	b1	b0
	s4(Cout)	s3	s2	s1	s0



7. Somador com Carry Look-Ahead (vai-um antecipado)

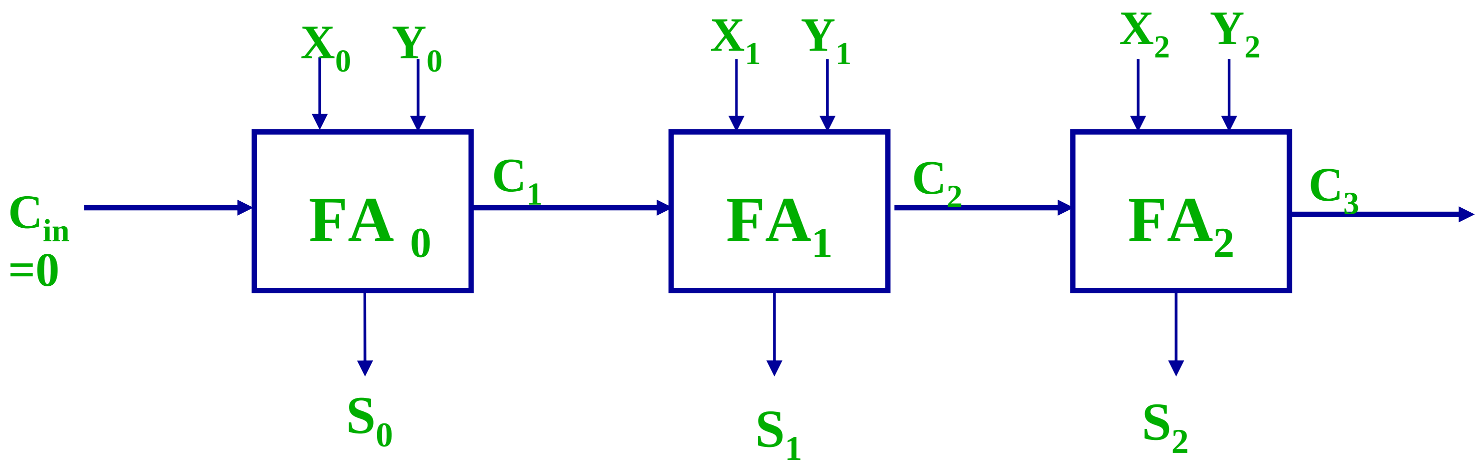
Problema com Somador "*Ripple Carry*" e com o somador usando HAs:

_ tempo de propagação do último carry-out (último 'vai-um')

p.ex.

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

- Existe um carry em cada estágio
- Bits de carry e soma do último estágio só estão disponíveis após os tempos de propagação dos estágios anteriores



Alternativa 1

- Calcular cada S_i diretamente em função de X_i, Y_i, X_{i-1}, Y_{i-1}, ...
  - construir tabela-verdade
  - implementar circuito com lógica de 2 níveis

p.ex. soma com 2 estágios

X_0	Y_0	X_1	Y_1	Cin	S_1
0	0	0	0	0	0
0	0	0	1	0	1
0	0	0	0	1	1
0	0	0	1	1	0
0	1	1	0	0	1
:	:	:	:	:	:

Vantagem: Tempo de propagação só de 2 portas

Desvantagem: Equações muito grandes quando N é grande

Exige muitas portas, com muitas entradas

Alternativa 2

Um estágio causa carry se

a) gerar um carry, pois $X_i = 1$ e $Y_i = 1$

ou

b) propagar um carry vindo do estágio anterior

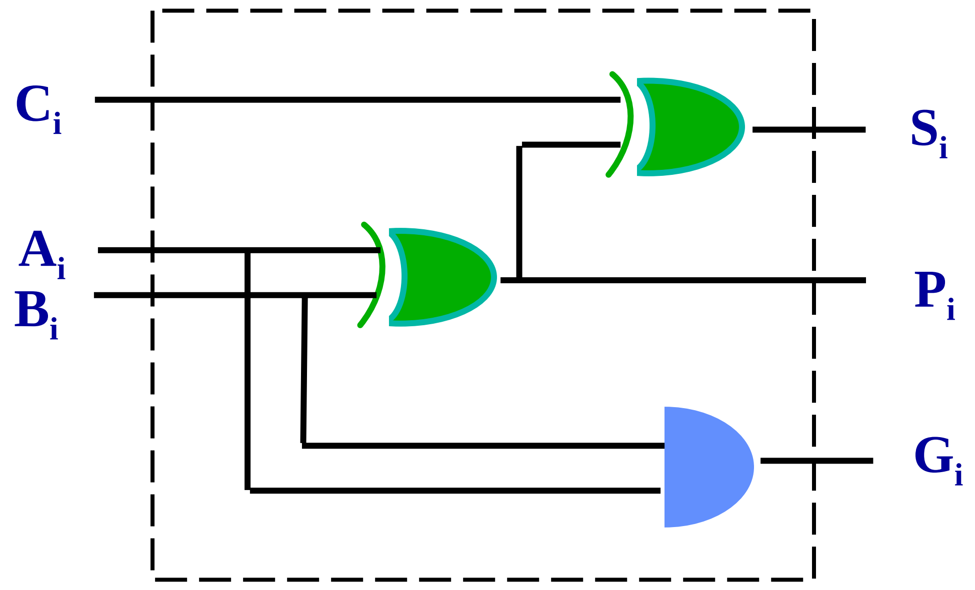
$C_i = 1$ e ($X_i = 1$ ou $Y_i = 1$)

mas não ambos, pois então recai-se no caso a

$$G_i = X_i \cdot Y_i$$

$$P_i = X_i \oplus Y_i$$

Unidade Somadora



- Expandindo as Equações para Geração de Carry :
  - P/ Carry Look-ahead de 1, 2 e 3 estágios

$$C_1 = G_0 + P_0\, C_0$$
$$C_2 = G_1 + P_1\, C_1 = G_1 + P_1\, (G_0 + P_0 C_0)$$
$$C_3 = G_2 + P_2\, C_2 = G_2 + P_2\, (G_1 + P_1\, (G_0 + P_0 C_0))$$
$$= G_2 + P_2\, G_1 + P_2\, P_1\, G_0 + P_2\, P_1\, P_0\, C_0$$

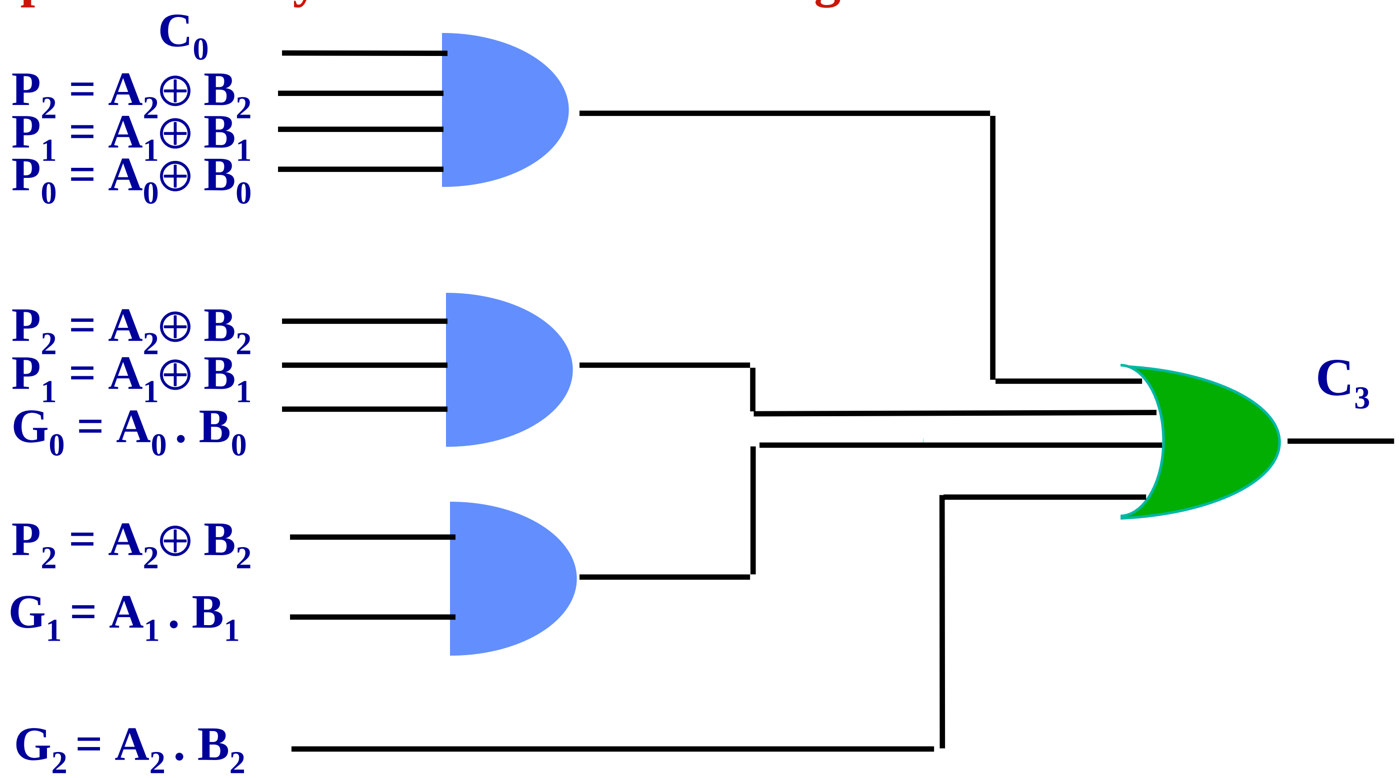
$$C_{out} = \underbrace{XY}_{G} + C_{in}\, \underbrace{(X \oplus Y)}_{P}$$

ou seja

$C_3 = 1$ se

- for gerado carry no estágio 2 (G_2) , ou
- for propagado carry pelo estágio 2, gerado no estágio 1 ($P_2\, G_1$), ou
- for propagado carry pelos estágios 1 e 2, gerado no estágio 0 ($P_2\, P_1 G_0$), ou
- for propagado o carry Co (entrada) pelos estágios 0, 1 e 2 ($P_2\, P_1 P_0 C_0$)

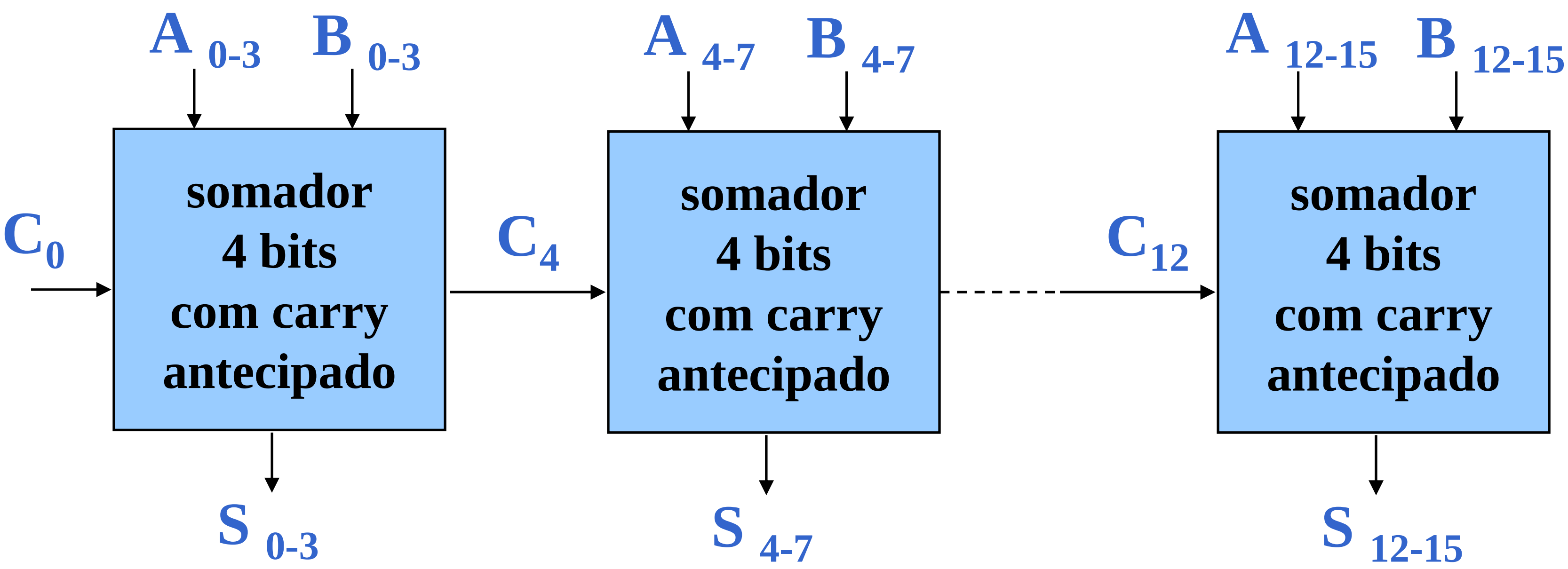
Rede Lógica para o Carry Look-ahead de 3 estágios



- <u>Analisando o tempo de propagação:</u>
  - C_i em cada estágio tem tempo de propagação de 3 portas
    - C_i em cada estágio não depende de C_{i-1}
    - C_i é calculado em função de A_i, B_i, A_{i-1}, B_{i-1},… e C_{i-3}
  - S_i em cada estágio tem tempo de propagação de 4 portas
- Número de Entradas nas portas AND ou OR (ou NAND ou NOR):
  - Para N-estágios de Look-Ahead ----> Portas de N+1 Entradas ! (atraso)

Solução intermediária (Com Carry Look-ahead de 4 estágios)

- por exemplo supondo um somador de 16 bits



- dentro de cada somador de 4 bits as equações não crescem demais
  - gasto moderado de portas e entradas
- tempo de propagação = 4 x tempo de um somador com carry antecipado