



Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
Curso de Graduação em Ciências da Computação



Sistemas Digitais

INE 5406

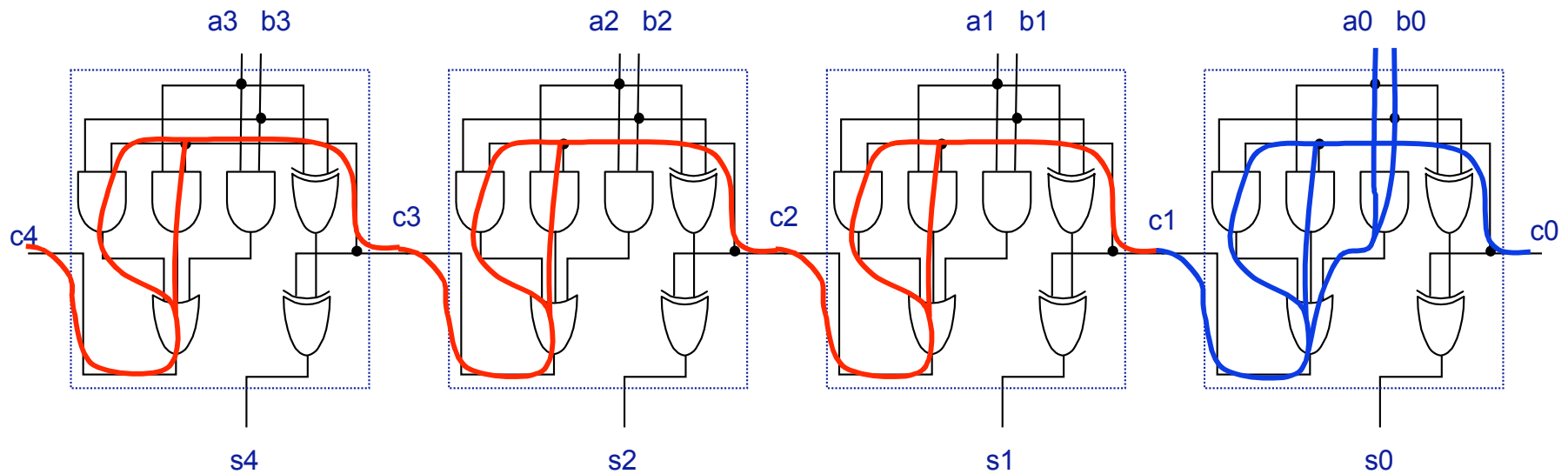
Suplemento à Aula 1-T
Arquiteturas de Somadores Rápidos.

Prof. José Luís Güntzel
guntzel@inf.ufsc.br

www.inf.ufsc.br/~guntzel/ine5406/ine5406.html

Arquiteturas de Somadores Rápidos

▶ Somador Paralelo Tipo *Ripple Carry*



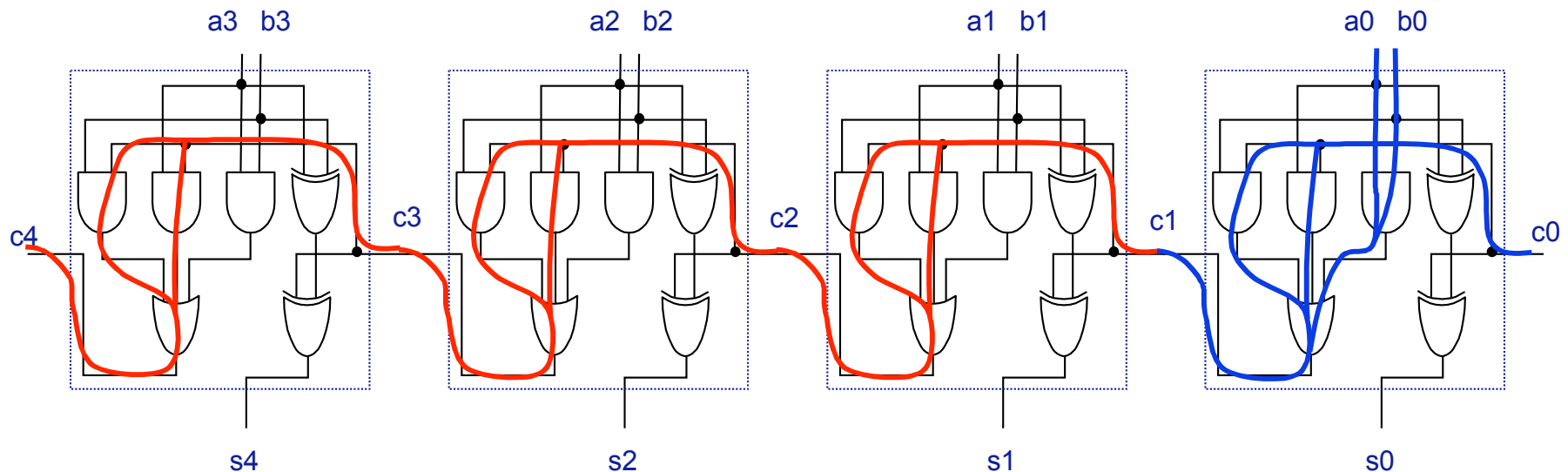
Suponha que no tempo $t=0$ um par de valores é aplicado às entradas (A,B):

- O resultado só estará pronto quando todas as saídas tiverem estabilizado
- s_i e c_{i+1} só estabilizam após c_i estabilizar
- **Em particular, s_3 e c_4 só estabilizam depois que c_3 , c_2 e c_1 tiverem estabilizados...**

Arquiteturas de Somadores Rápidos

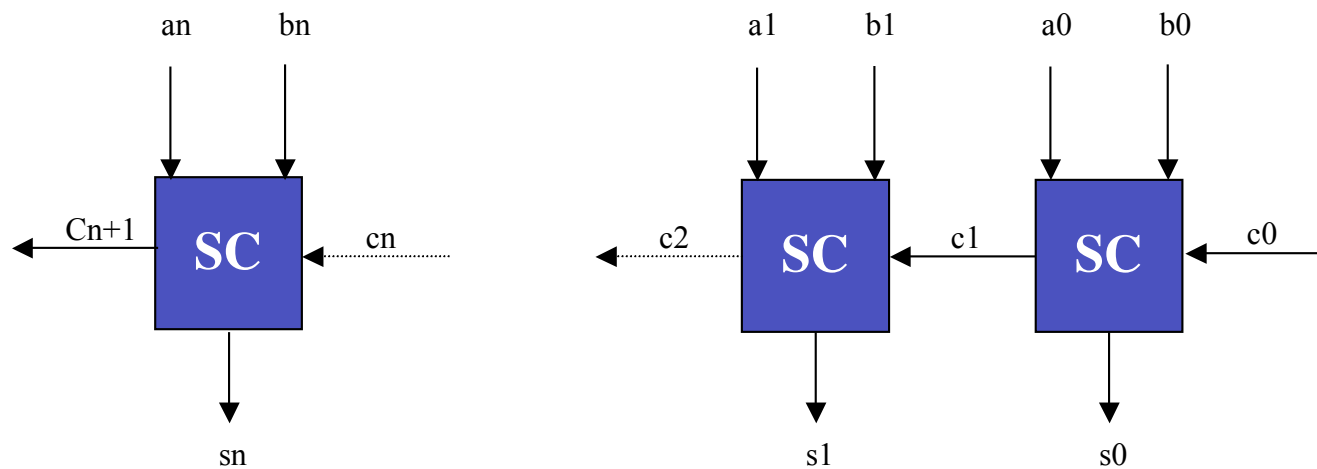
► Analisando a Propagação do *Carry*

Os caminhos críticos em somadores *Ripple Carry* passam pela cadeia de propagação do *carry*



Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*



Pergunta: será que não é possível alterar a arquitetura do somador, de modo a “quebrar” ou reduzir tal interdependência?

A resposta é ...

SIM!!!

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

Há três diferentes casos na propagação do *carry* que são mutuamente exclusivos:

a_i	b_i	c_i	k_i	c_{i+1}	s_i
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	1	1

Caso 1 (sinal *Kill*)

Quando as entradas A e B do somador local são iguais a zero, independentemente do *carry* de entrada, o *carry* de saída será igual a zero e, portanto, o estágio “matará” a propagação do *carry* do estágio anterior.

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

a_i	b_i	c_i	g_i	c_{i+1}	s_i
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Caso 2 (sinal *Generate*)

Quando as entradas A e B do somador local são iguais a um, independentemente do *carry* de entrada, o *carry* de saída será igual a um e, portanto, haverá geração de *carry* neste estágio.

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

a_i	b_i	c_i	p_i	c_{i+1}	s_i
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	1	0
1	1	1	0	1	1

Caso 3 (sinal *Propagate*)

Quando uma das entradas, A ou B, do somador local for igual a um e a outra for igual a zero, o *carry* de saída será igual ao *carry* de entrada e, portanto, o *carry* será propagado.

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

Resumindo os três casos do *carry*, temos:

Caso	a_i	b_i	c_{i+1}
Kill (k_i)	0	0	0
Propagate (p_i)	0	1	c_i
	1	0	c_i
Generate (g_i)	1	1	1

Agora, a idéia é:

- Encontrar uma equação para cada um dos três sinais (k_i , p_i e g_i)
- Achar as equações de s_i e c_{i+1} em função de k_i , p_i e g_i

Mas qual é a vantagem desta “reestruturação lógica”?

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

Caso	a_i	b_i	c_{i+1}
Kill (k_i)	0	0	0
Propagate (p_i)	0	1	c_i
	1	0	c_i
Generate (g_i)	1	1	1

Equações das funções k_i , p_i e g_i são:

$$k_i = \overline{a_i} \cdot \overline{b_i} = \overline{a_i + b_i}$$

$$p_i = a_i \oplus b_i$$

$$g_i = a_i \cdot b_i$$

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

Caso	a_i	b_i	c_{i+1}
Kill (k_i)	0	0	0
Propagate (p_i)	0	1	c_i
	1	0	c_i
Generate (g_i)	1	1	1

Então, usando soma de produtos, o *carry out* do estágio i pode ser definido como:

$$c_{i+1} = g_i + p_i \cdot c_i$$

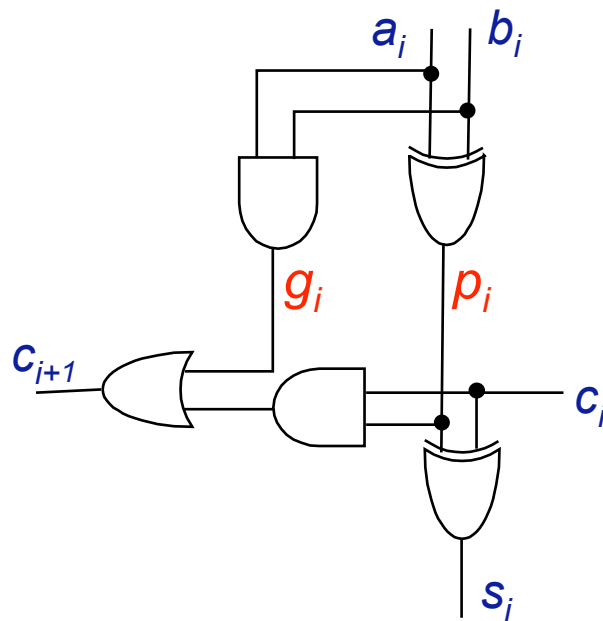
E a saída “soma” do estágio i pode ser expressa (em função de p_i e c_i) por:

$$s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$$

Arquiteturas de Somadores Rápidos

► Analisando a Propagação do *Carry*

As fórmulas para s_i e c_{i+1} geram o mesmo circuito já estudado, com base em dois meio-somadores.



$$p_i = a_i \oplus b_i$$

$$g_i = a_i \cdot b_i$$

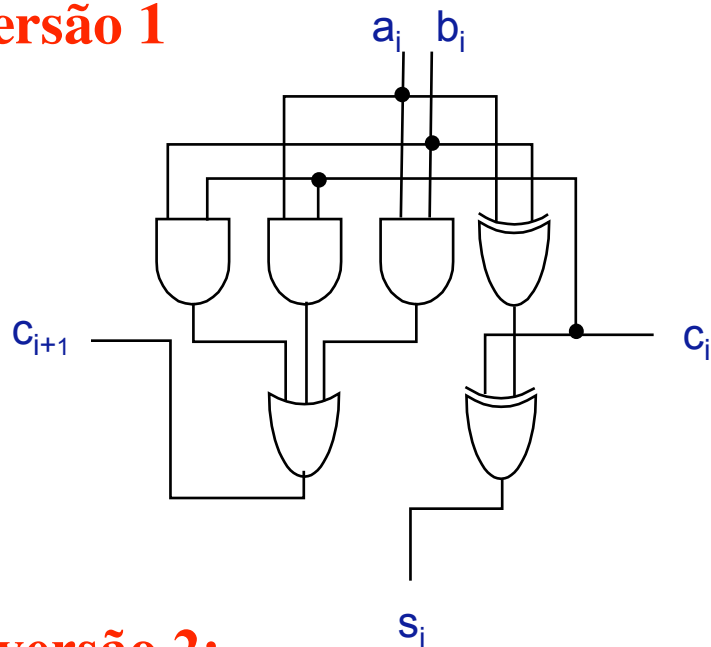
$$c_{i+1} = g_i + p_i \cdot c_i$$

$$s_i = p_i \oplus c_i$$

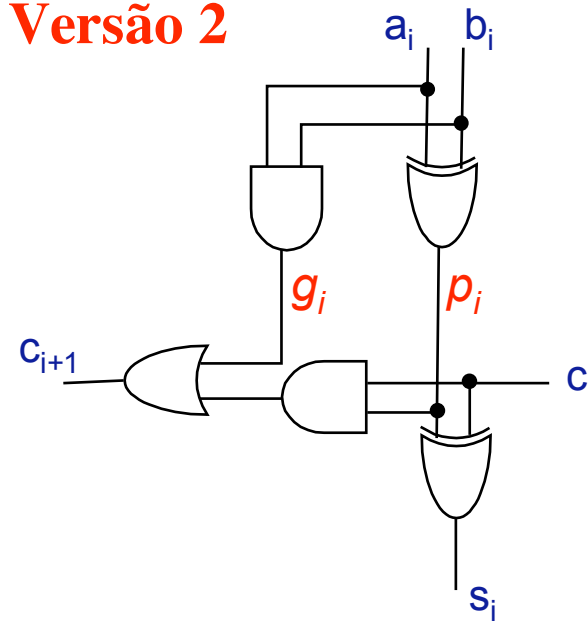
Arquiteturas de Somadores Rápidos

▶ Comparando o Desempenho das Duas Versões

Versão 1



Versão 2



Na versão 2:

- Se $g_i=1$ não é preciso esperar pelo valor de C_i para determinar o valor de C_{i+1}
- Se $p_i=1$, então já se sabe *a priori* que $C_{i+1} = C_i$

Arquiteturas de Somadores Rápidos

▶ Somadores Rápidos

Para reduzir-se o atraso na propagação do *carry* as seguintes abordagens podem ser usadas:

1. Reduzir o atraso na geração do *carry* (aplicada nos somadores **Manchester**);
2. Diminuir o atraso da cadeia de propagação do *carry* (aplicada nos somadores *Carry-Lookahead*, *Carry-Select*, *Carry-Skip*, etc.);
3. Mudar o sistema de representação numérica (não será tratado na disciplina).



Estas soluções investem em desempenho, mas resultam em acréscimo de recursos (número de transistores utilizados).

Arquiteturas de Somadores Rápidos

► Somadores Manchester *Chain*

Princípio: usar um circuito mais rápido para propagar a cadeia de *carry*;

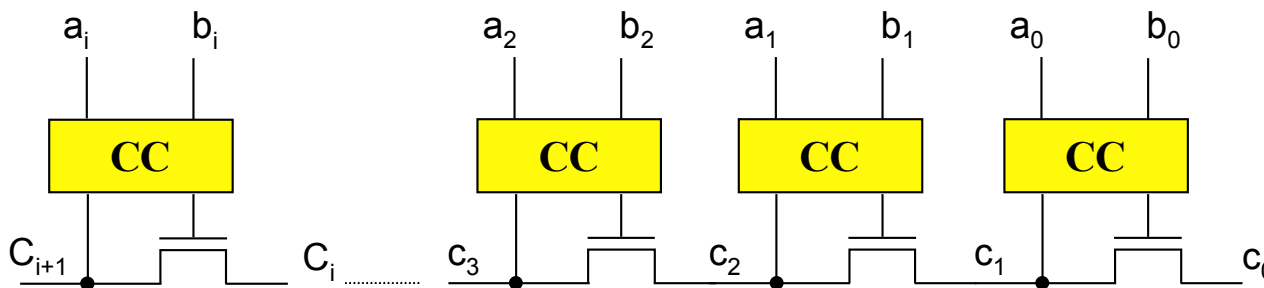
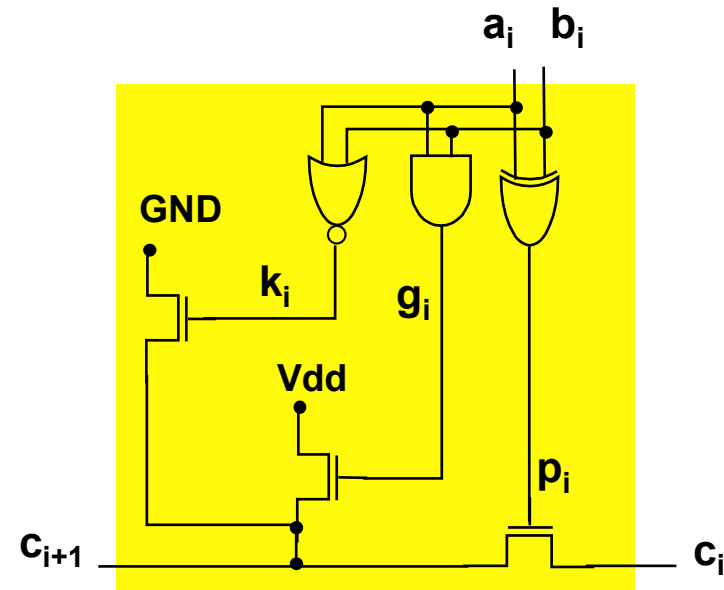
- Este circuito mais rápido pode ser constituído de *Transmission Gates* ou transistores de passagem;
- Como as situações de *kill*, *propagate* e *generate* são mutuamente exclusivas, pode-se construir um conjunto de chaves com a seguinte lógica:
 - Se $k_i = 1$, então $c_{i+1} = 0$
 - Se $g_i = 1$, então $c_{i+1} = 1$
 - Se $p_i = 1$, então $c_{i+1} = c_i$

Arquiteturas de Somadores Rápidos

► Somadores Manchester *Chain*

Uma Implementação com transistores de passagem

Controle da Cadeia de Propagação do *Carry* (CC)

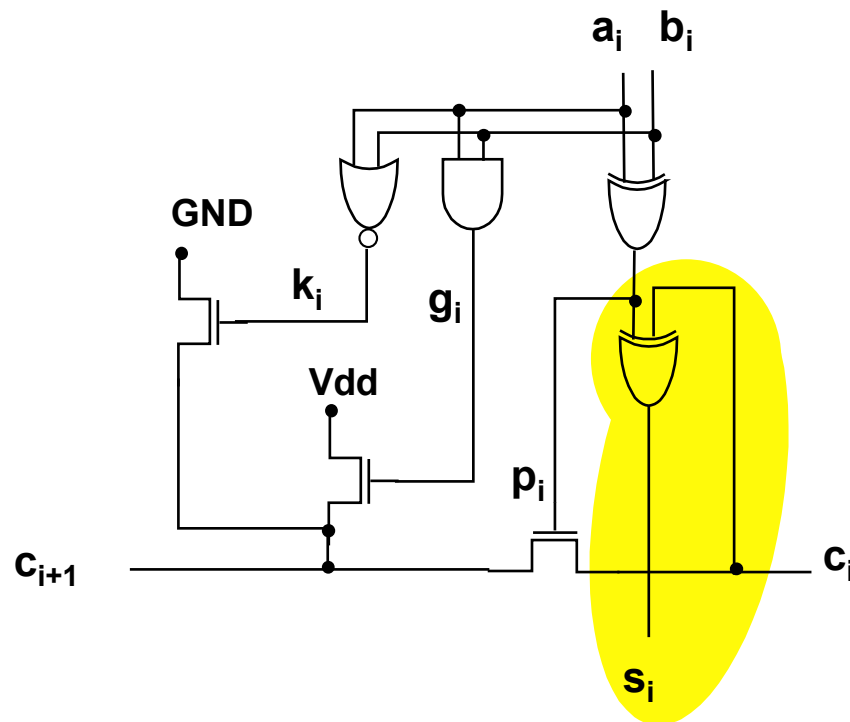


Cadeia de propagação do *Carry*

Arquiteturas de Somadores Rápidos

► Somadores Manchester Chain

E como seria o somador *Manchester Chain* completo?



Basta modificar o controle da cadeia de propagação do *carry* para que ele calcule também a saída S_i

Arquiteturas de Somadores Rápidos

▶ Somadores *Carry Lookahead*

Princípio: paralelizar o cálculo dos *carries*

- No extremo, todos os *carries* podem ser computados ao mesmo tempo.
- Para o somador *Carry-Lookahead*, pode-se considerar que não existe mais a exclusividade mútua entre os sinais *generate* e *propagate*.
- Então, a função *propagate* pode ser simplificada para um simples “OU” entre as duas entradas, pois se o nível de soma gera *carry out* ($g_i = 1$), não importa o valor de *propagate*.

Arquiteturas de Somadores Rápidos

▶ Somadores *Carry Lookahead*

Assim, para o *carry-lookahead* temos que:

$$p_i = a_i + b_i$$

$$c_{i+1} = g_i + p_i \cdot c_i$$

Então:

$$c_1 = g_0 + p_0 \cdot c_0$$

$$c_2 = g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0)$$

Expandindo, segue:

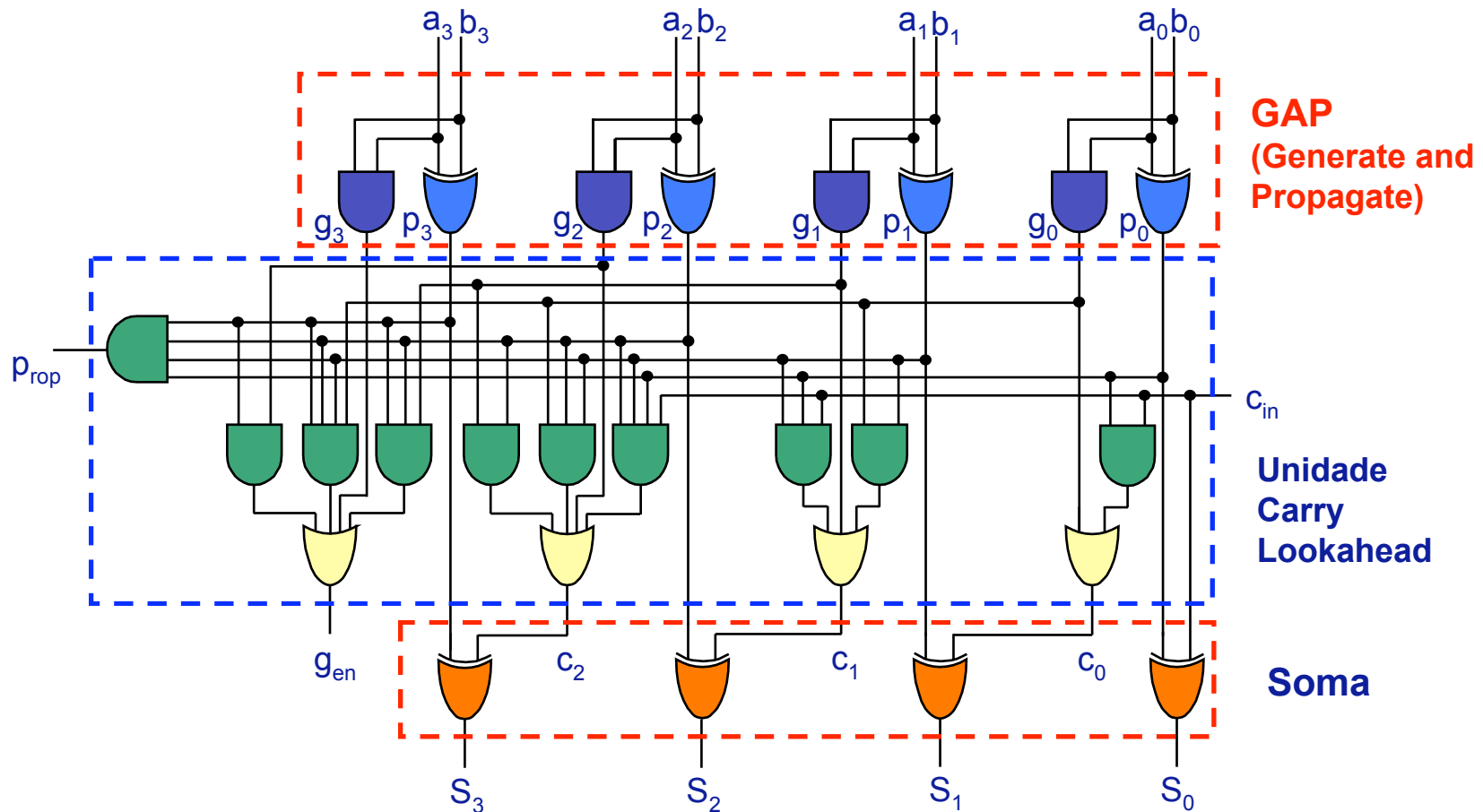
$$c_2 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0$$

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

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

Arquiteturas de Somadores Rápidos

► Somadores *Carry Lookahead*



Arquiteturas de Somadores Rápidos

▶ Somadores *Carry Lookahead*

- Problema com os CLGs: a complexidade da equação do *carry* cresce muito rapidamente!!!
- Para somadores com entradas usando muitos bits, a propagação do *carry* com cadeia *carry-lookahead* é mais lenta que um *ripple carry*.
- Tipicamente, o *carry-lookahead* não é usado para somadores com entradas maiores que 4 bits.

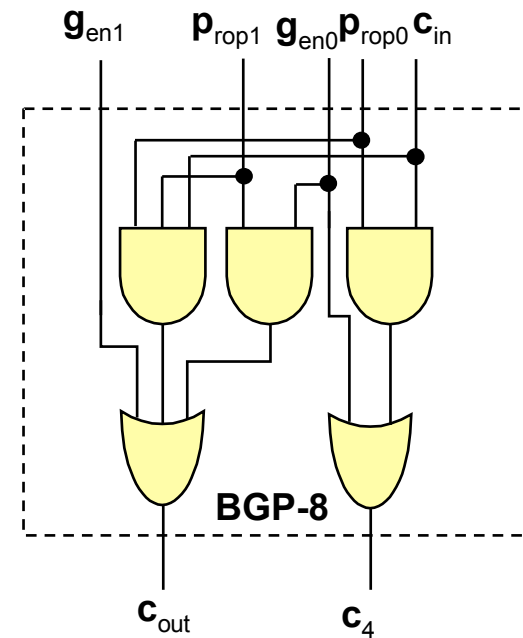
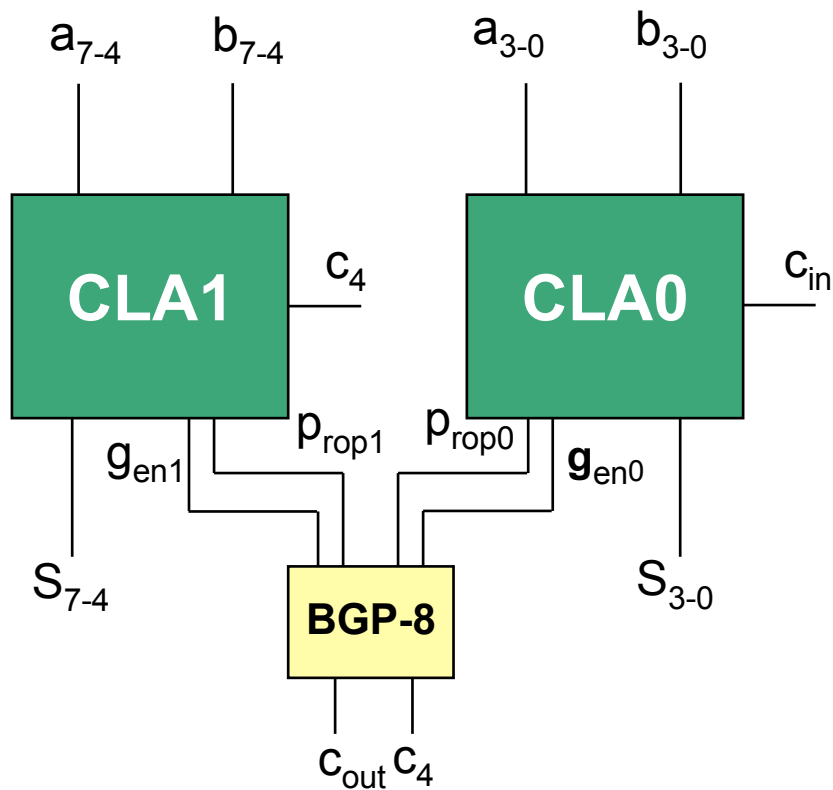
Arquiteturas de Somadores Rápidos

▶ Somadores *Carry Lookahead*

- **Solução:** aplicar o *lookahead* em grupos de somadores com, no máximo, 4 bits.
- Funções auxiliares P e G são necessárias e indicam, respectivamente:
 - Se $P = 1$, então o *carry* é propagado pelo grupo.
 - Se $G = 1$, então o grupo gera *carry out*.

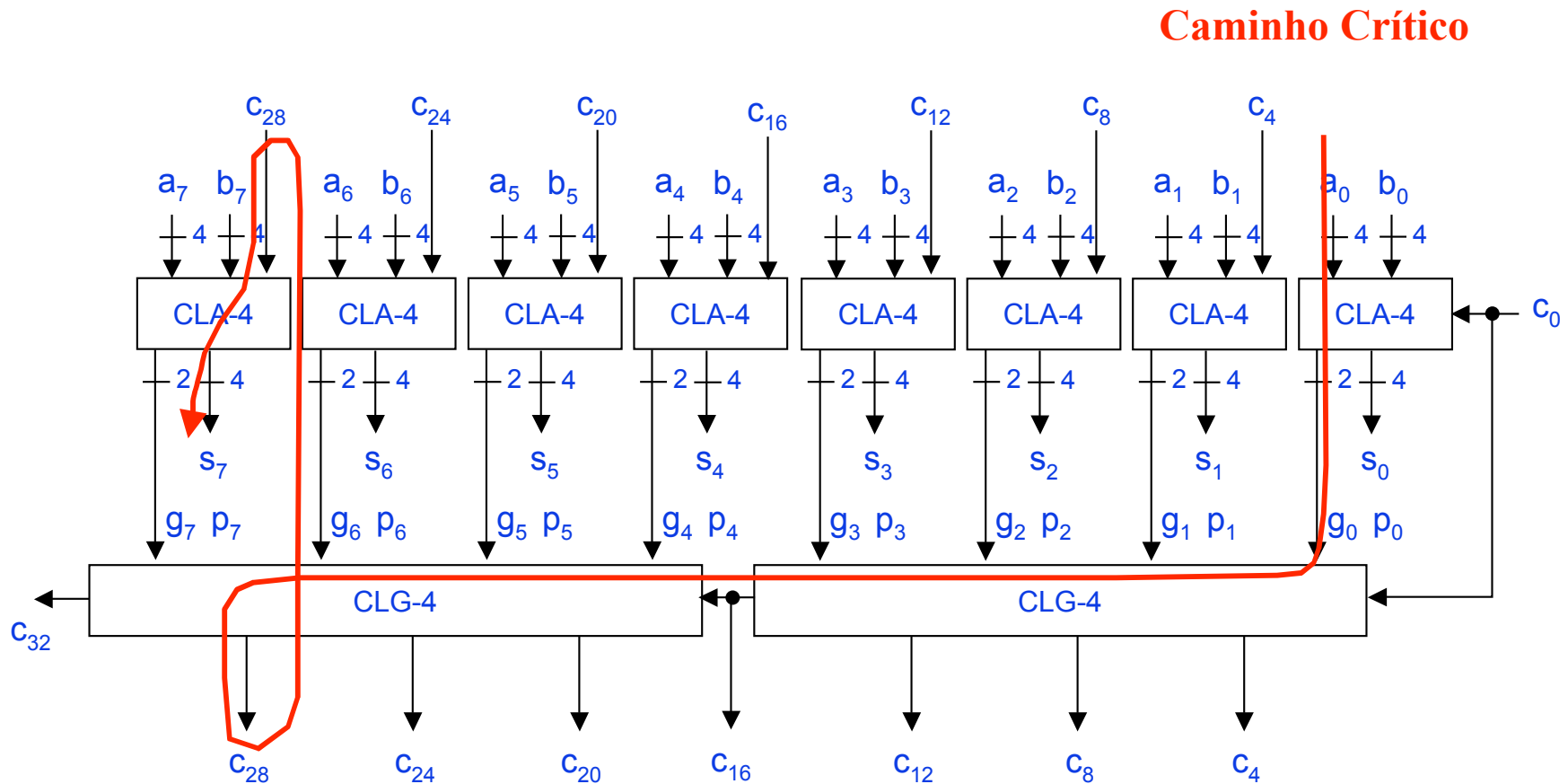
Arquiteturas de Somadores Rápidos

▶ Somadores *Carry Lookahead*



Arquiteturas de Somadores Rápidos

► Somadores *Carry Lookahead*



Arquiteturas de Somadores Rápidos

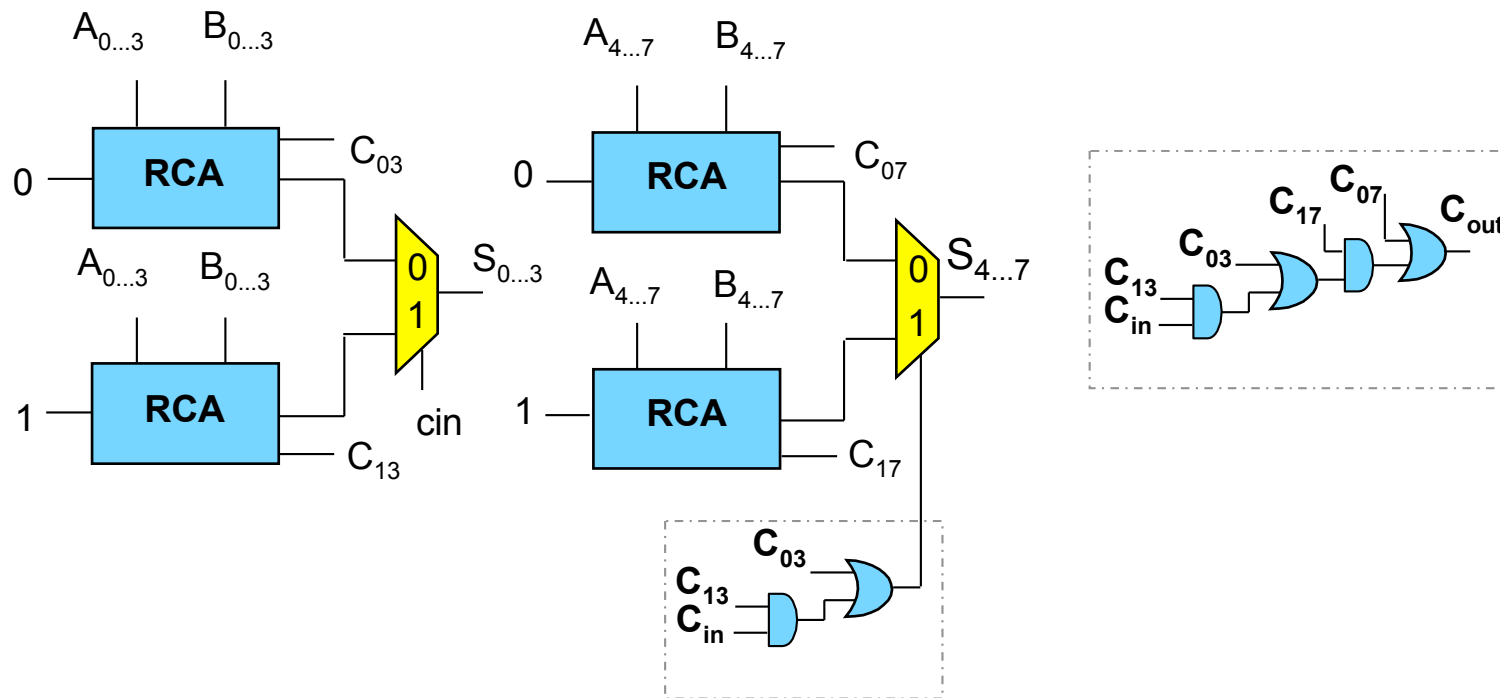
▶ Somadores *Carry Select*

Princípios:

- dividir a adição em seções de 4 ou 8 bits
 - realizar a adição de cada seção simultaneamente para os dois casos possíveis (*carry in=0* e *carry in=1*)
-
- Em cada seção de adição são usados dois somadores (*ripple carry* ou *carry lookahead*) idênticos e um multiplexador
 - O multiplexador seleciona um dos dois resultados, utilizando como controle o *carry out* da seção anterior

Arquiteturas de Somadores Rápidos

► Somadores *Carry Select* com 8 bits (usando somadores RCA...)



Arquiteturas de Somadores Rápidos

▶ Somadores *Carry Select*

Equações utilizadas pelos MUX de cada seção de soma

$$B1 = C_{in}$$

$$B2 = C03 \text{ or } (C13 \text{ and } C_{IN})$$

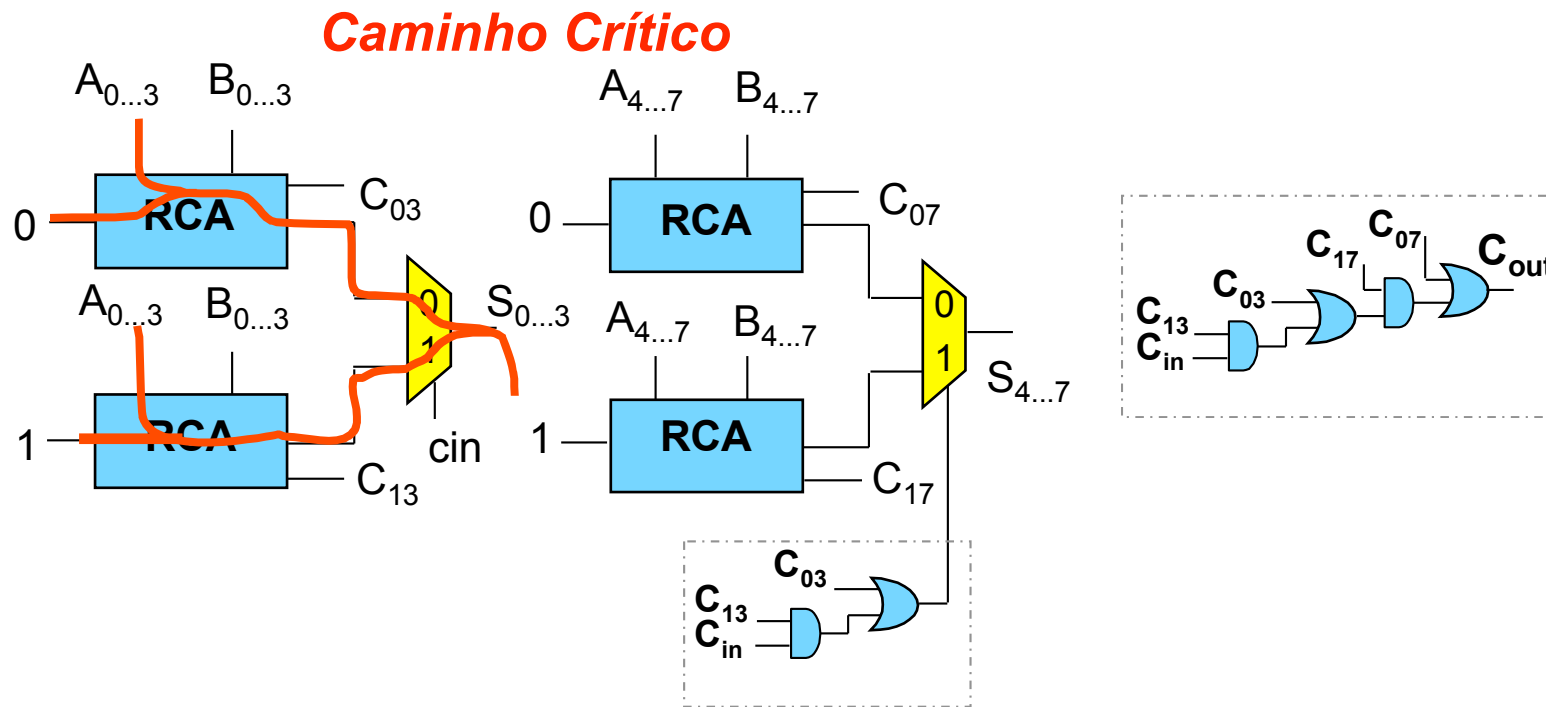
$$B3 = C07 \text{ or } (C17 \text{ and } (C03 \text{ or } (C13 \text{ and } C_{IN})))$$

$$B4 = C011 \text{ or } (C111 \text{ and } (C07 \text{ or } (C17 \text{ and } (C03 \text{ or } (C13 \text{ and } C_{IN}))))))$$

$$B4 = C015 \text{ or } (C115 \text{ and } (C011 \text{ or } (C111 \text{ and } (C07 \text{ or } (C17 \text{ and } (C03 \text{ or } (C13 \text{ and } C_{IN})))))))$$

Arquiteturas de Somadores Rápidos

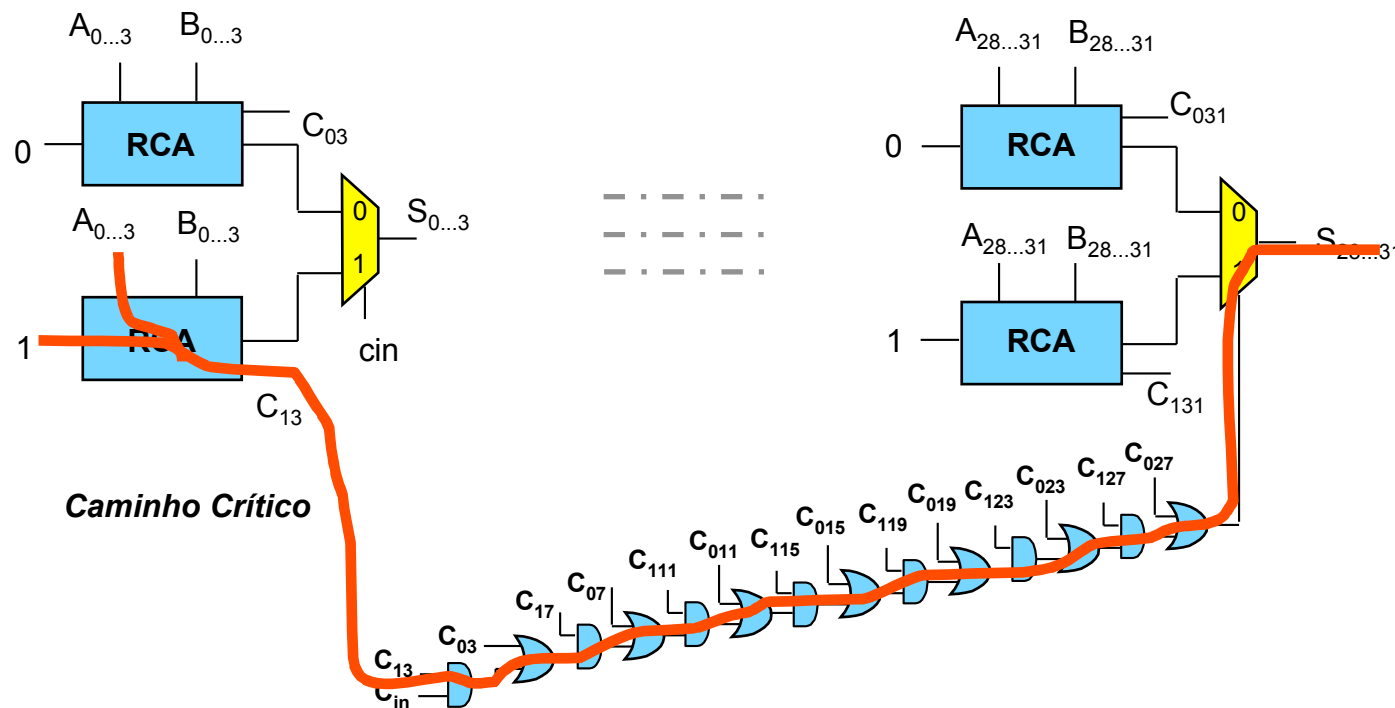
► Somadores *Carry Select* com 8 bits



Atraso crítico = Atraso RCA + Atraso MUX

Arquiteturas de Somadores Rápidos

► Somadores *Carry Select* com 32 bits



Atraso crítico = Atraso RCA + Atraso Equação + Atraso MUX