Capítulo 2-Projeto Lógico Combinacional I

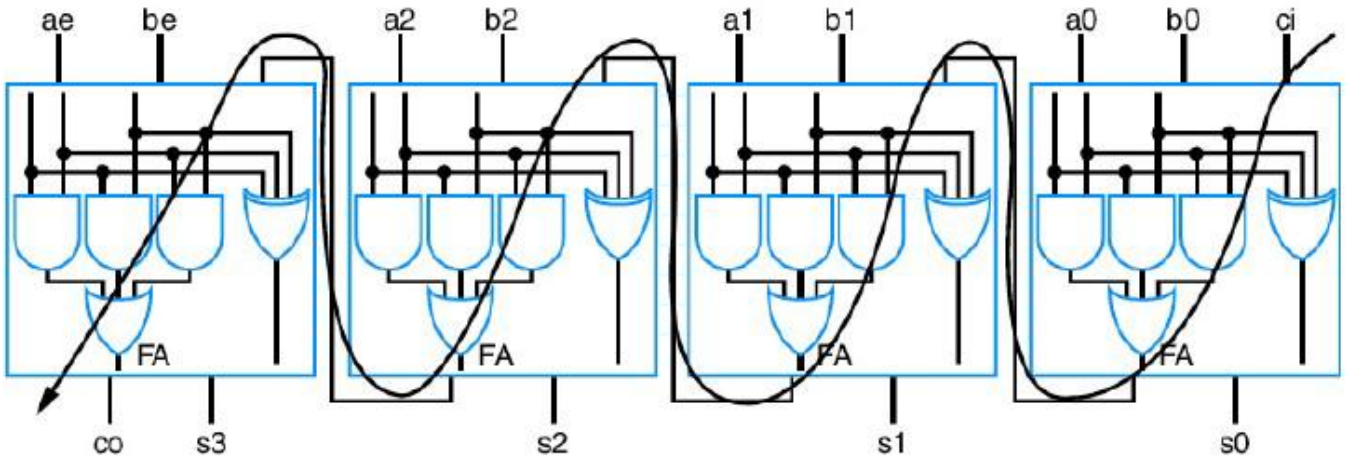
Profa. Eliete Caldeira

Tradeoffs de Componentes de Blocos Operacionais

- Anteriormente:
  - versões mais básicas e fáceis de compreender desses componentes
- Agora:
  - métodos para construir versões mais rápidas ou menores de alguns deles

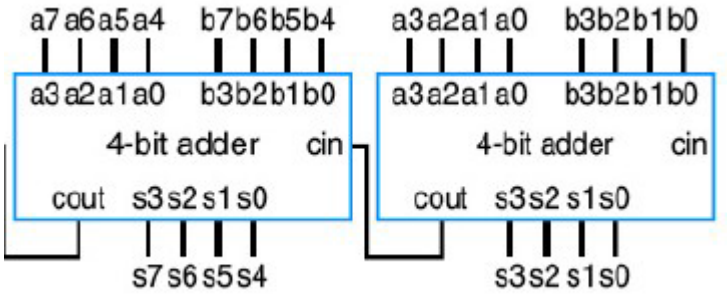
Somador

- Somador com propagação do bit de transporte de “vai um” requer que:
  - os bits de “vai um” se propaguem através de todos os somadores completos antes que todas as saídas fiquem corretas.
- O caminho mais longo através do circuito é conhecido como caminho crítico do circuito.
- Grande atraso



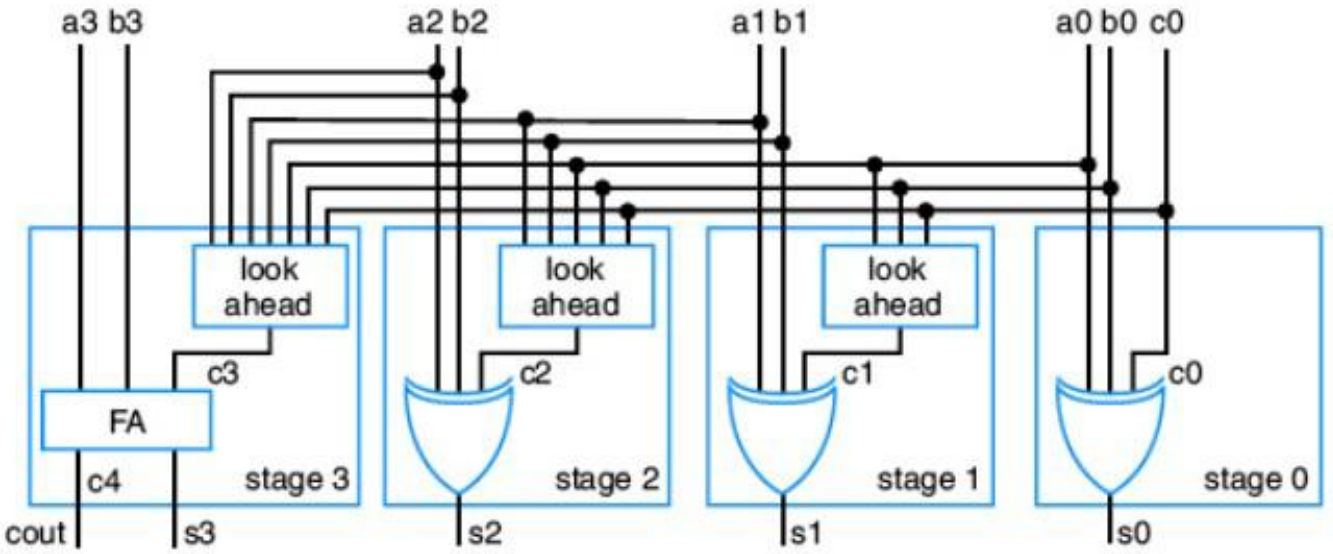
Somador: como melhorar?

- **Somador com lógica de dois níveis**
  - Somador projetado com o uso de uma lógica de dois níveis tem um atraso de apenas duas portas
- A construção de um somador de N bits usando dois níveis de lógica resulta em circuitos excessivamente grandes à medida que N cresce acima de oito ou mais bits
  - Um somador com 4 bits – cerca de 100 portas. Assim, podemos utilizar 2 somadores de 4 bits para gerar um de 8 (apenas 4 atrasos usando lógica de 2 níveis)
  - Quanto maior o número de entradas na porta, maior o atraso que ela apresenta



Somador: como melhorar?

- Somador com antecipação de transporte
  - A velocidade do somador com antecipação e aumentada, mas não são usadas tantas portas como em um somador com dois níveis de lógica



Somador de transporte antecipado

- Se $s_i = a_i \oplus b_i \oplus c_i$ e $c_{i+1} = a_i b_i + a_i c_i + b_i c_i$
- Então:
  - $c_1 = a_0 b_0 + a_0 c_0 + b_0 c_0$
  - $c_2 = a_1 b_1 + a_1 c_1 + b_1 c_1$
  - $c_3 = a_2 b_2 + a_2 c_2 + b_2 c_2$
  - $c_4 = a_3 b_3 + a_3 c_3 + b_3 c_3$
- Substituindo c_1 em c_2:
  - $c_2 = a_1 b_1 + a_1(a_0 b_0 + a_0 c_0 + b_0 c_0) + b_1(a_0 b_0 + a_0 c_0 + b_0 c_0)$
  - $c_2 = a_1 b_1 + a_1 a_0 b_0 + a_1 a_0 c_0 + a_1 b_0 c_0 + b_1 a_0 b_0 + b_1 a_0 c_0 + b_1 b_0 c_0$

Somador de transporte antecipado

- Substituindo c_2 em c_3:
  - $c_3 = a_2b_2+a_2(a_1b_1+a_1a_0b_0+a_1a_0c_0+a_1b_0c_0+b_1a_0b_0+ b_1a_0c_0+b_1b_0c_0)+b_2(a_1b_1+a_1a_0b_0+a_1a_0c_0+a_1b_0c_0+ b_1a_0b_0+b_1a_0c_0 +b_1b_0c_0)$
  - $c_3 = a_2b_2+a_2a_1b_1+a_2a_1a_0b_0+a_2a_1a_0c_0+a_2a_1b_0c_0+ a_2b_1a_0b_0+a_2b_1a_0c_0+a_2b_1b_0c_0+b_2a_1b_1+b_2a_1a_0b_0+ b_2a_1a_0c_0+b_2a_1b_0c_0+b_2b_1a_0b_0+b_2b_1a_0c_0 +b_2b_1b_0c_0$
- Substituindo c_3 em c_4:
  - $c_4 = a_3b_3+a_3(a_2b_2+a_2a_1b_1+a_2a_1a_0b_0+a_2a_1a_0c_0+a_2a_1b_0c_0+ a_2b_1a_0b_0+a_2b_1a_0c_0+a_2b_1b_0c_0+b_2a_1b_1+b_2a_1a_0b_0+ b_2a_1a_0c_0+b_2a_1b_0c_0+b_2b_1a_0b_0+b_2b_1a_0c_0 +b_2b_1b_0c_0)+ b_3(a_2b_2+a_2a_1b_1+a_2a_1a_0b_0+a_2a_1a_0c_0+a_2a_1b_0c_0+a_2b_1a_0b_0+ a_2b_1a_0c_0+a_2b_1b_0c_0+b_2a_1b_1+b_2a_1a_0b_0+b_2a_1a_0c_0+b_2a_1b_0c_0+ b_2b_1a_0b_0+b_2b_1a_0c_0 +b_2b_1b_0c_0)$

Somador de transporte antecipado

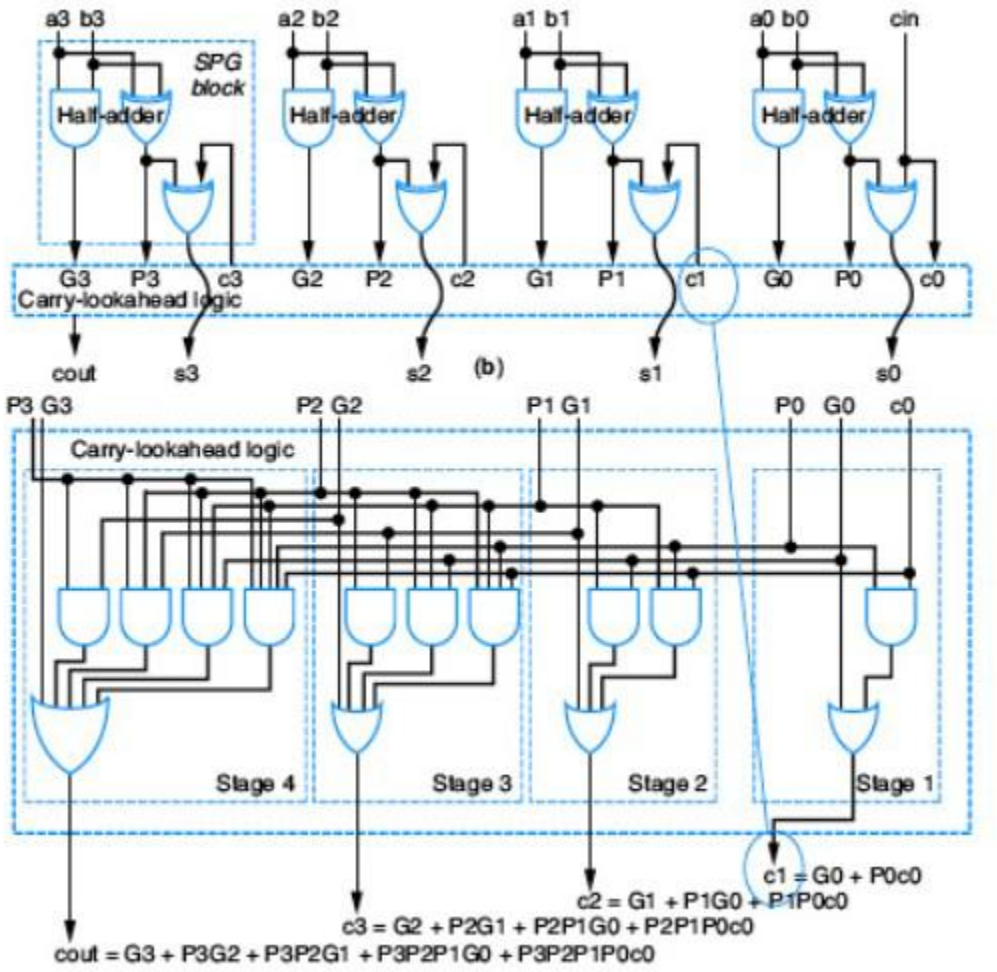
- Esquema mais eficiente de antecipação do bit de transporte

- Se a_i+b_i gera vai-um ($a_i = 1$ e $b_i = 1$), então $c_{i+1}=1$ independente de c_i
  - Se $c_i = 0$, então $a_i+b_i+c_i = 10$
  - Se $c_i = 1$, então $a_i+b_i+c_i = 11$
- Se $a_i+b_i = 1$(($a_i = 1$ e $b_i = 0$) ou ($a_i = 0$ e $b_i = 1$)), então $c_{i+1} = c_i$
  - Se $c_i = 0$, então $a_i+b_i+c_i = 01$
  - Se $c_i = 1$, então $a_i+b_i+c_i = 10$
- Ou seja,
  - $c_{i+1} = a_i b_i + (a_i \oplus b_i)\, c_i$

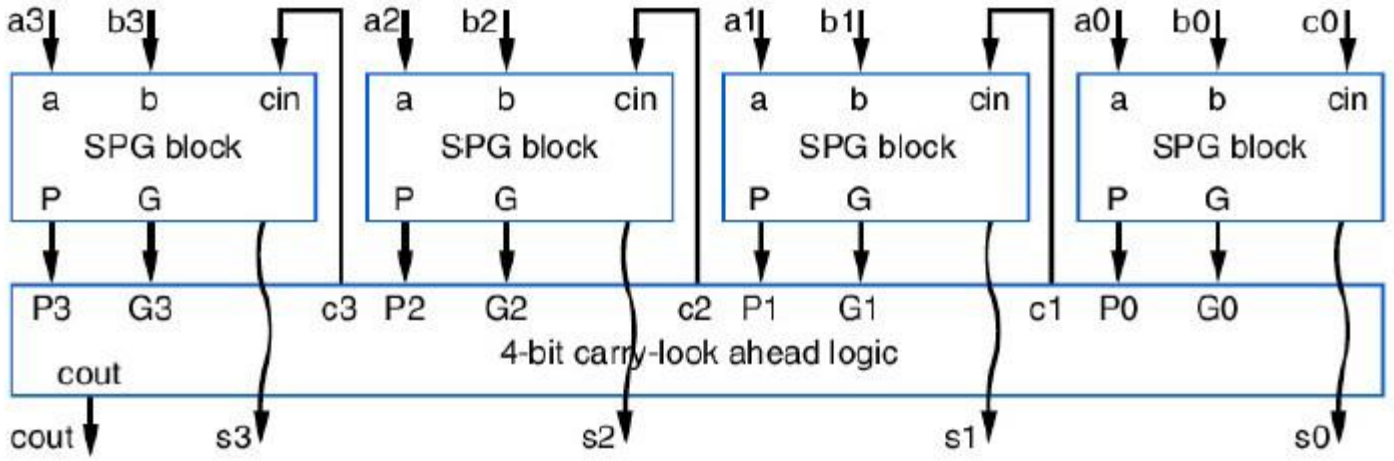
Somador de transporte antecipado

- Sejam definidos:
  - gerar como $g_i = a_i b_i$
  - e propagar como $p_i = a_i \oplus b_i$
- Então
  - $c_1 = g_0 + p_0 c_0$
  - $c_2 = g_1 + p_1 c_1 = g_1 + p_1(g_0 + p_0 c_0) = g_1 + p_1 g_0 + p_1 p_0 c_0$
  - $c_3 = g_2 + p_2 c_2 = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 c_0) =$
    $g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$
  - $c_4 = g_3 + p_3 c_3 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) =$
    $g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$
- Além disto $s_i = p_i \oplus c_i$

Somador de transporte antecipado



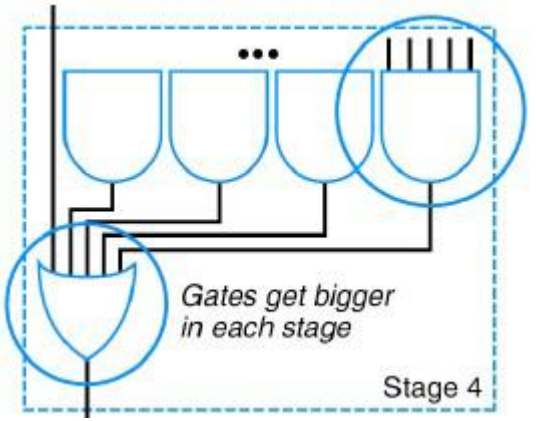
Somador de transporte antecipado



- Cada bloco SPG tem 3 portas para gerar g_i (and), p_i (xor) e s_i (xor), total de 12 portas para 4 bits
- O bloco de antecipação de transporte tem 2+3+4+5 = 14 portas para 4 bits
- Total de 26 portas com 4 níveis de tempo de atraso
- Para 8 bits 8x3+(2+3+4+5+6+7+8+9) = 68 com o atraso de 4 portas lógicas

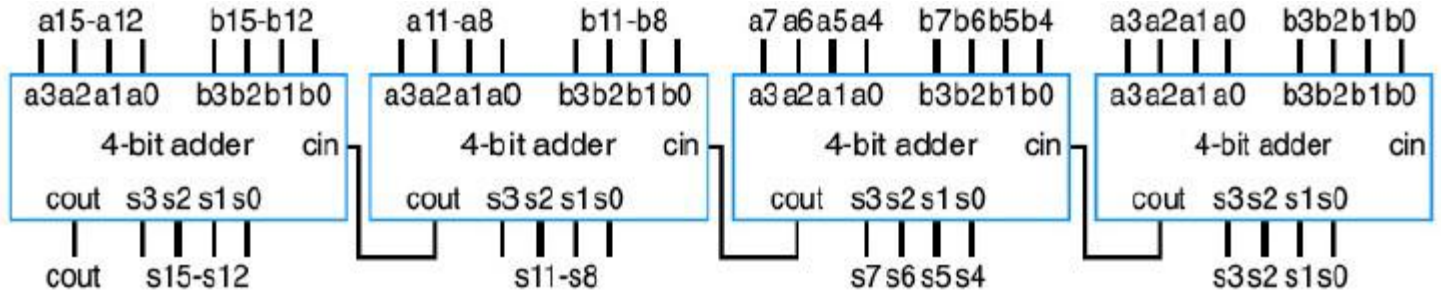
Somador de transporte antecipado

- Problemas: à medida que o número de bits do somador aumenta, as portas do circuito de antecipação de transporte têm cada vez mais entradas.
- Portas com muitas entradas gastam muitos transistores, ocupam maior área e têm tempo de atraso maior
- Além disto, portas com muitas entradas são construídas de árvores de portas menores.



Somadores hierárquicos com antecipação de bit de transporte

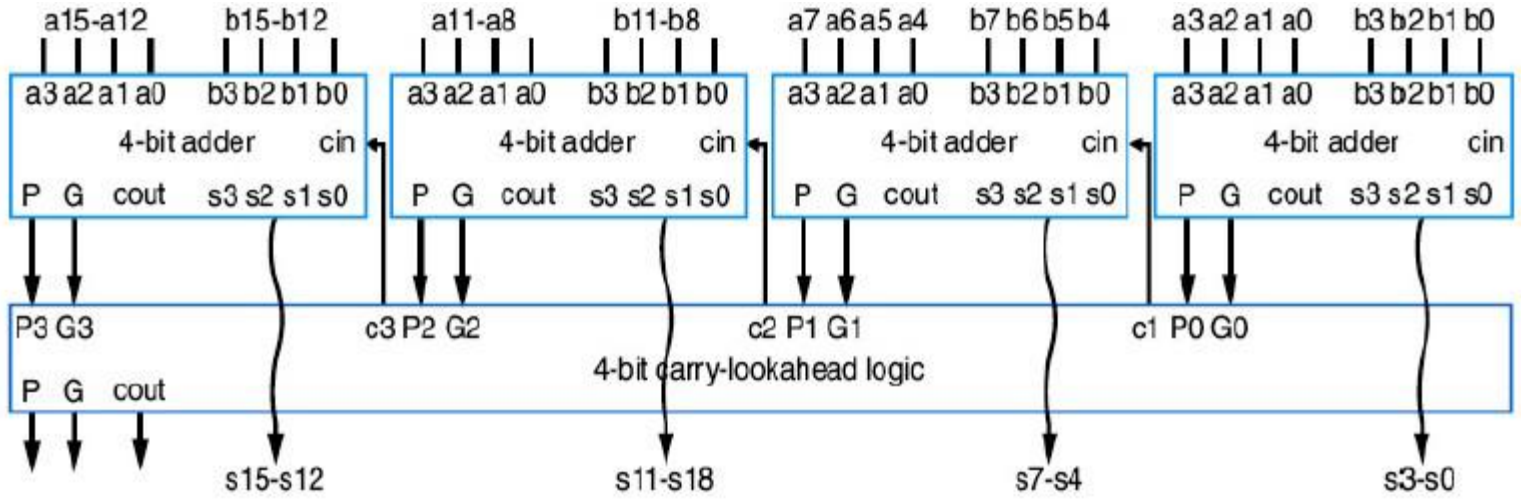
- Podemos construir somadores a partir de somadores menores
- A primeira forma é conectar para propagar transporte
  - 16 atrasos de porta versus 32 atrasos no caso do somadores de propagação



Somadores hierárquicos com antecipação de bit de transporte

- Outra forma é conectar usando um gerador de transporte antecipado
- No módulo $P = p_3p_2p_1p_0$

e $G = g_3+p_3g_2+p_3p_2g_1+p_3p_2p_1g_0$



Somadores hierárquicos com antecipação de bit de transporte

- Para 32 bits:

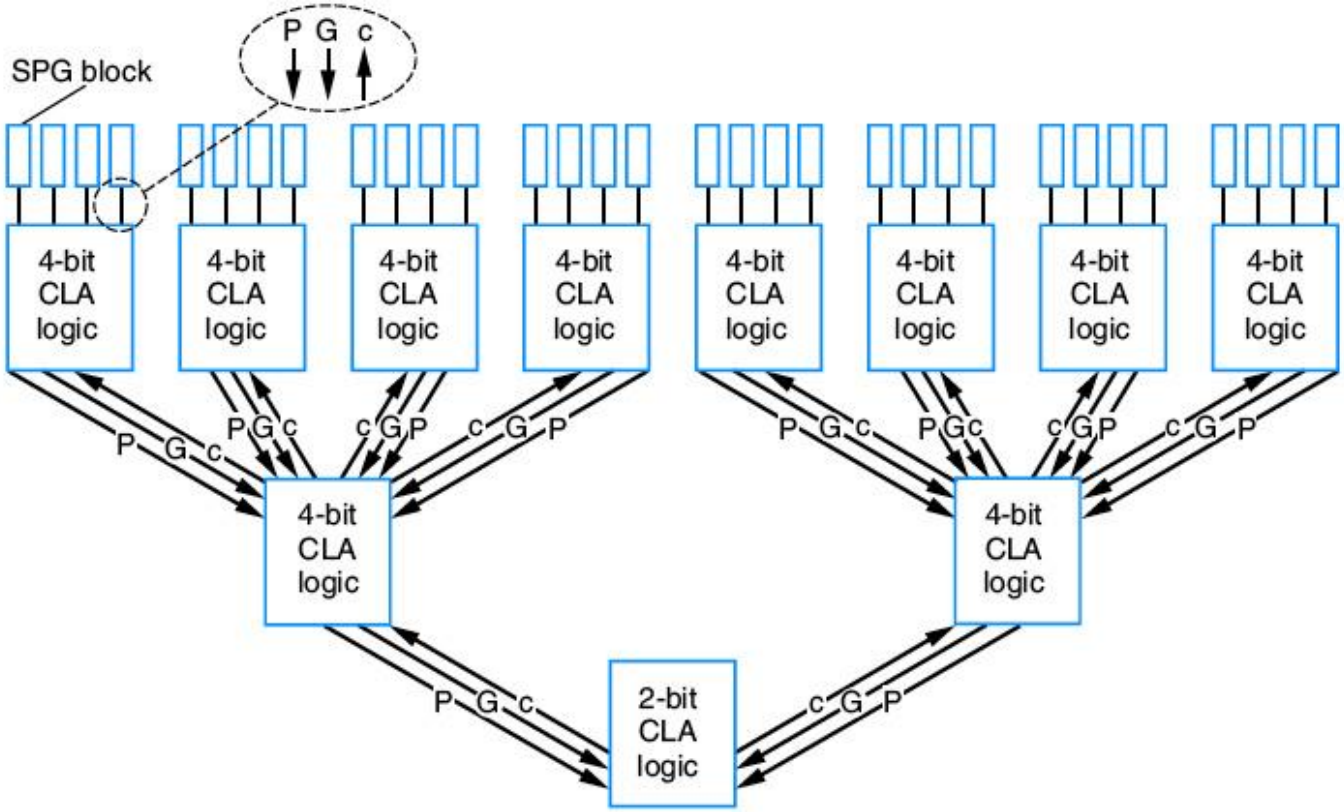


Figure 6.63 View of multilevel carry-lookahead, showing tree structure, which enables fast addition with reasonable numbers and sizes of gates. Each level adds only two gate-delays.

Somador com seleção de bit de transporte

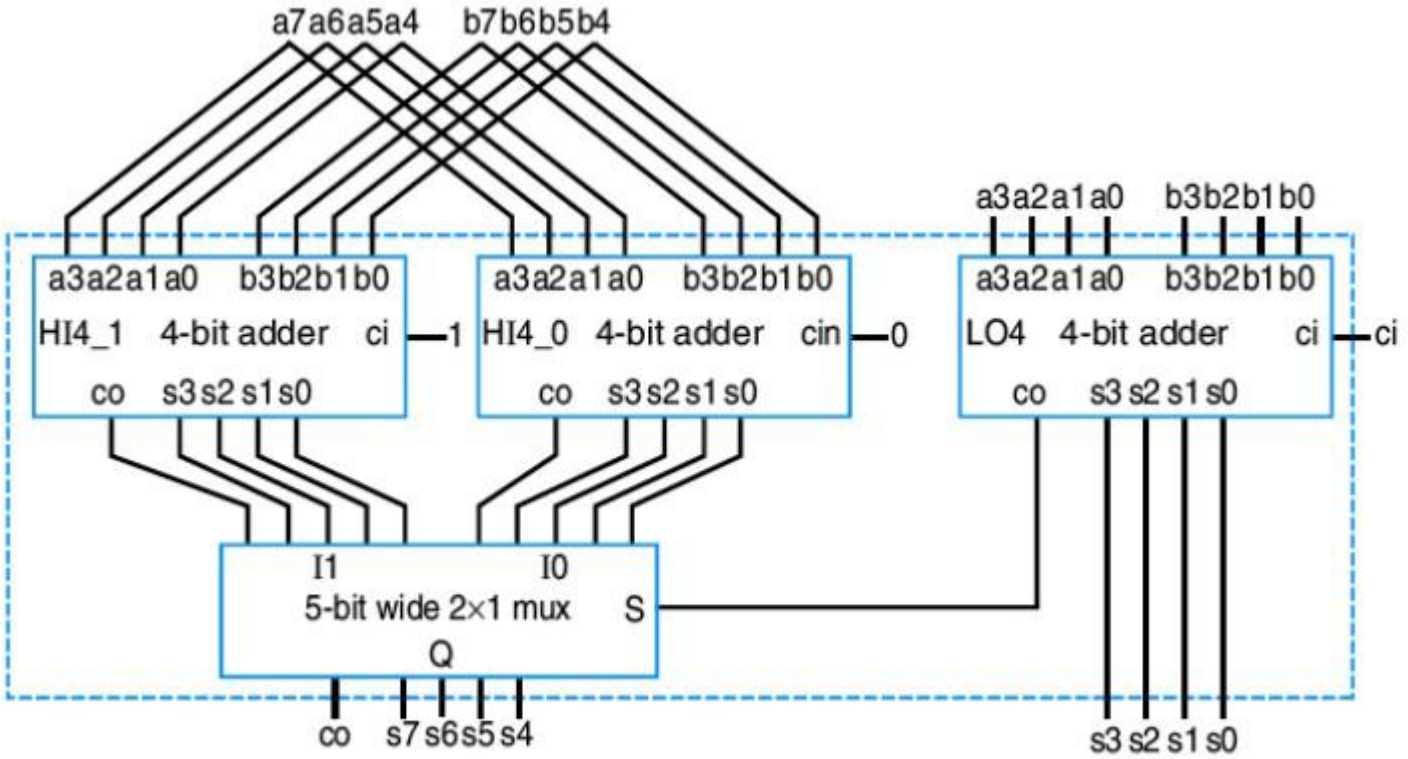


Figure 6.64 8-bit carry-select adder implemented using three 4-bit adders.

Tradeoff de somadores

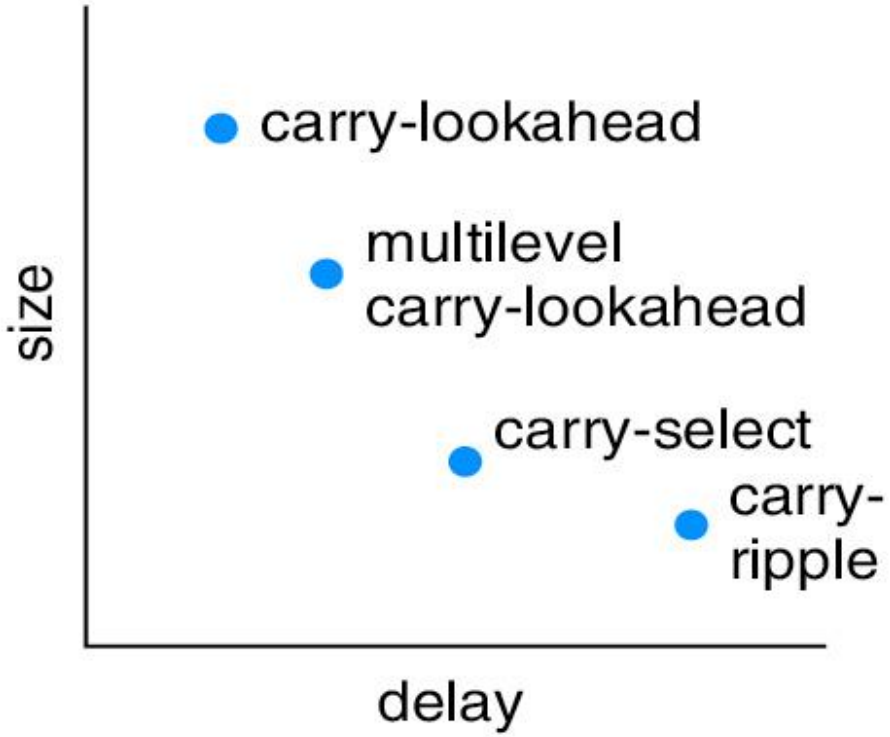


Figure 6.65 Adder tradeoffs.

Unidade Lógica e Aritmética

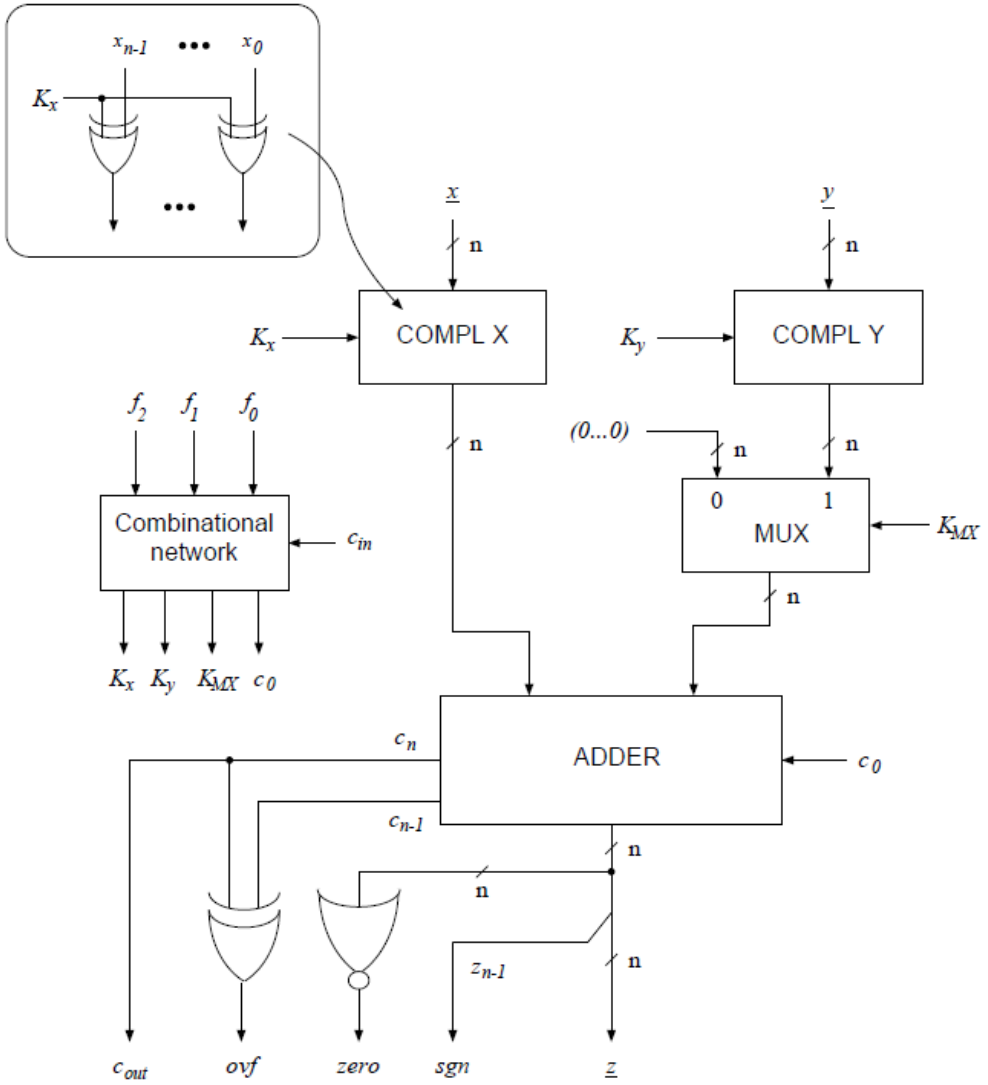
- ULA ou ALU (*Arithmetic Logic Unit*) em complemento de 2
- Entradas:
  - $x = (x_{n-1}, \ldots, x_0)$; $x_j \in \{0,1\}$
  - $y = (y_{n-1}, \ldots, y_0)$; $y_j \in \{0,1\}$
  - $c_{in} \in \{0,1\}$
  - $F = (f_2, f_1, f_0)$
- Saídas:
  - $z = (z_{n-1}, \ldots, z_0)$; $z_j \in \{0,1\}$
  - c_{in}, sgn, zero, ovf $\in \{0,1\}$
- A tabela mostra as funções

F	Operation		
001	ADD	add	$z = x + y$
011	SUB	subtract	$z = x - y$
101	ADDC	add with carry	$z = x + y + c_{in}$
110	CS	change sign	$z = -x$
010	INC	increment	$z = x + 1$

$sgn = 1$ **if** $z < 0$, 0 **otherwise** (the sign)
$zero = 1$ **if** $z = 0$, 0 **otherwise**
$ovf = 1$ **if** z overflows, 0 **otherwise**

Unidade Lógica e Aritmética

- Como deve ser a tabela-verdade do circuito que gera K_x, K_y, K_{mx} e c_0 a partir de F e de c_{in}?



Unidade Lógica e Aritmética

- Geração dos sinais e controle

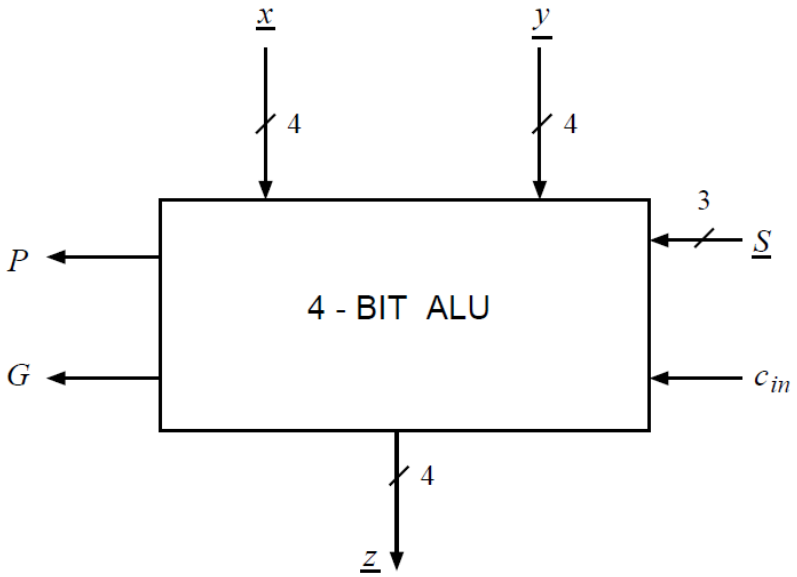
$$a_i = \begin{cases} b_i & \textbf{if} \quad K = 0 \\ b_i' & \textbf{if} \quad K = 1 \end{cases}$$

Operation	Op-code		Control Signals		
	$f_2 f_1 f_0$	$\underline{z}$	K_x	K_y	K_{MX}
ADD	001	$\mathrm{ADD}(\underline{x}, \underline{y}, 0)$	0	0	1
SUB	011	$\mathrm{ADD}(\underline{x}, \underline{y'}, 1)$	0	1	1
ADDC	101	$\mathrm{ADD}(\underline{x}, \underline{y}, c_{\mathrm{in}})$	0	0	1
CS	110	$\mathrm{ADD}(\underline{x'}, \underline{0}, 1)$	1	d.c.	0
INC	010	$\mathrm{ADD}(\underline{x}, \underline{0}, 1)$	0	d.c.	0

$$\begin{aligned} K_x &= f_2 f_1 \\ K_y &= f_1 \\ K_{MX} &= f_0 \\ c_0 &= f_1 + f_2 f_0 c_{\mathrm{in}} \end{aligned}$$

Redes ALU

- Considere o módulo ALU onde P e G são sinais de propagar e gerar e $c_{out} = G+P.c_{in}$



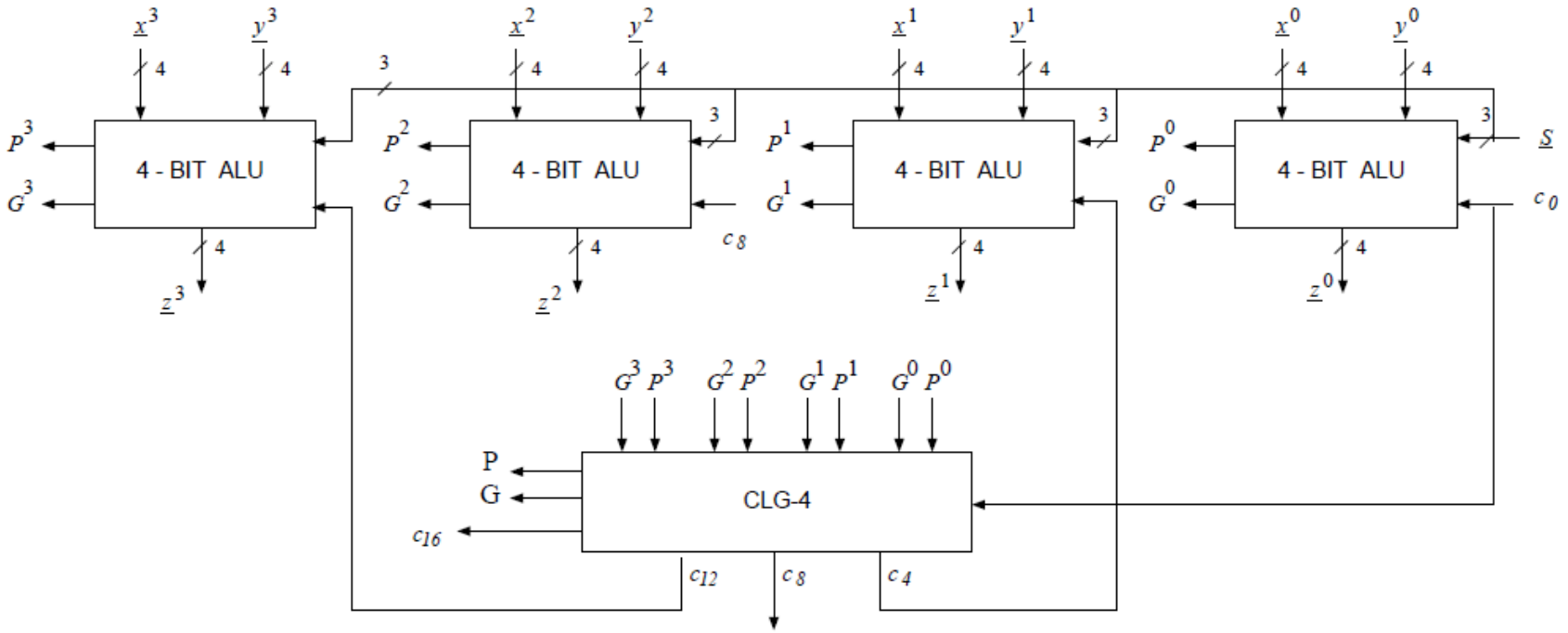
Control (S)	Function
ZERO	$z = 0$
ADD	$z = (x + y + c_{\text{in}}) \bmod 16$
SUB	$z = (x + y' + c_{\text{in}}) \bmod 16$
EXSUB	$z = (x' + y + c_{\text{in}}) \bmod 16$
AND	$\underline{z} = \underline{x} \cdot \underline{y}$
OR	$\underline{z} = \underline{x} + \underline{y}$
XOR	$\underline{z} = \underline{x} \oplus \underline{y}$
ONE	$\underline{z} = 1111$

a' denotes the integer represented by vector $\underline{a}'$

$\cdot$, $+$, and $\oplus$ are applied to the corresponding bits

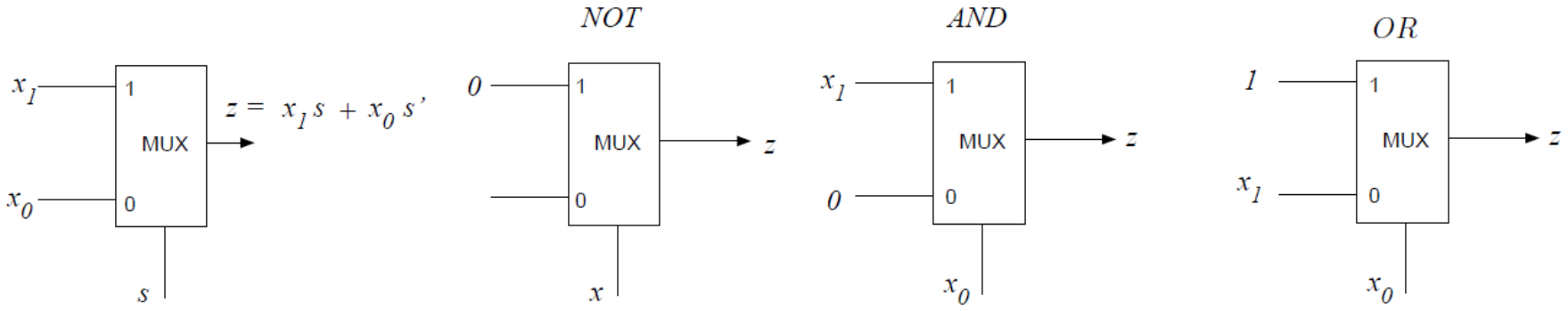
Redes ALU

- Rede *carry skip* de ALU's



Implementação de circuitos lógicos usando MUX 2:1

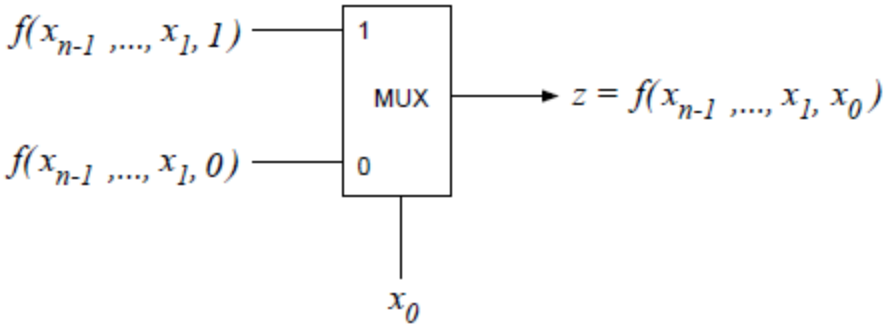
- Um MUX 2:1 é um bloco universal, ou seja, com ele é possível fazer as operações NOT, AND e OR



Implementação de circuitos lógicos usando MUX 2:1

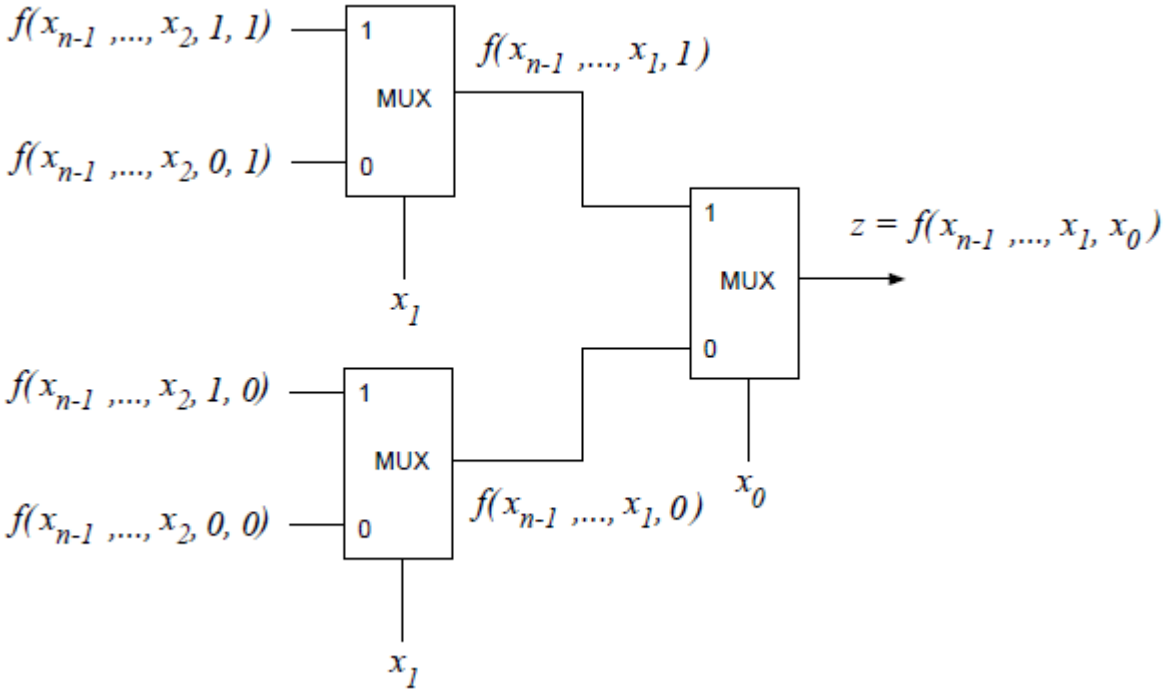
- Decomposição de Shannon

$$z = f(x_{n-1}, x_{n-2}, \ldots, x_0) = f(x_{n-1}, x_{n-2}, \ldots, 1) \cdot x_0 + f(x_{n-1}, x_{n-2}, \ldots, 0) \cdot x_0'$$



Implementação de circuitos lógicos usando MUX 2:1

- Repetindo a decomposição de Shannon pode-se obter uma árvore de MUXes



Implementação de circuitos lógicos usando MUX 2:1 – Exemplo

- Implemente usando MUX 2:1a função

$$f(x_3,x_2,x_1,x_0) = z = x_3\,(x_1 + x_2x_0)$$

- Decomponha na sequência em relação a x_0, x_1, x_2 e x_3

Implementação de circuitos lógicos usando MUX 2:1 – Exemplo

- Implemente usando MUX 2:1a função

  $$f(x_3,x_2,x_1,x_0) = z = x_3\,(x_1+x_2x_0)$$

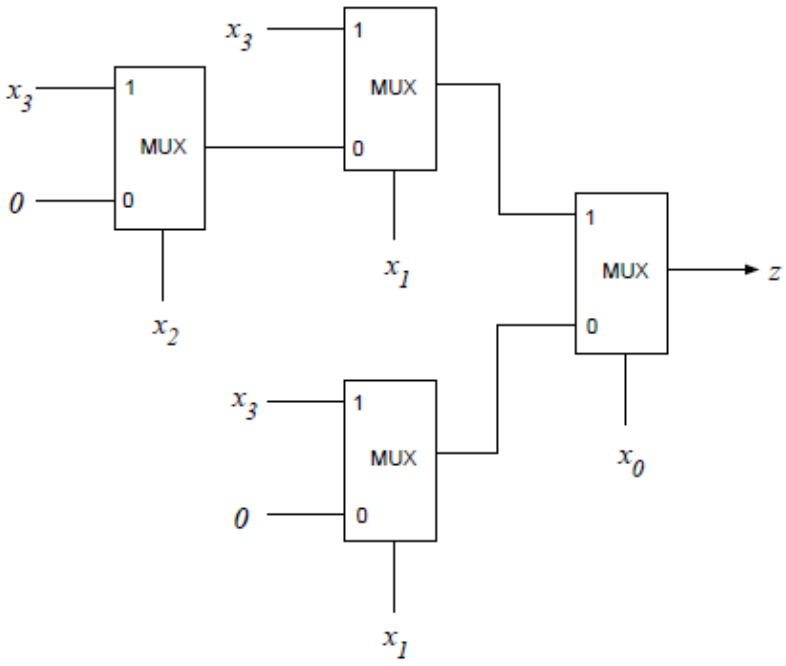
- Decomponha na sequência em relação a x_0, x_1, x_2 e x_3
  - $f(x_3,x_2,x_1,0) = x_3\,(x_1+x_2.0)= x_3\,(x_1+0)= x_3\,.x_1$
  - $f(x_3,x_2,x_1,1) = x_3\,(x_1+x_2.1)= x_3\,(x_1+x_2)$
  - $f(x_3,x_2,0,0) = x_3\,.0= 0$
  - $f(x_3,x_2,1,0) = x_3\,.1= x_3$
  - $f(x_3,x_2,0,1) = x_3\,(0+x_2)= x_3\,(x_2)= x_3.x_2$
  - $f(x_3,x_2,1,1) = x_3\,(1+x_2)= x_3\,(1) = x_3$
  - $f(x_3,0,0,1) = x_3.0 = 0$
  - $f(x_3,1,0,1) = x_3.1 = x_3$

Implementação de circuitos lógicos usando MUX 2:1 – Exemplo

- Implemente usando MUX 2:1a função

  $f(x_3,x_2,x_1,x_0) = z = x_3\,(x_1+x_2x_0)$

- Decomponha na sequência em relação a x_0, x_1, x_2 e x_3
  - $f(x_3,x_2,x_1,0) = x_3\,(x_1+x_2.0) = x_3\,(x_1+0) = x_3\,.x_1$
  - $f(x_3,x_2,x_1,1) = x_3\,(x_1+x_2.1) = x_3\,(x_1+x_2)$
  - $f(x_3,x_2,0,0) = x_3\,.0 = 0$
  - $f(x_3,x_2,1,0) = x_3\,.1 = x_3$
  - $f(x_3,x_2,0,1) = x_3\,(0+x_2) = x_3\,(x_2) = x_3.x_2$
  - $f(x_3,x_2,1,1) = x_3\,(1+x_2) = x_3\,(1) = x_3$
  - $f(x_3,0,0,1) = x_3.0 = 0$
  - $f(x_3,1,0,1) = x_3.1 = x_3$



Implementação de circuitos lógicos usando MUX 2:1

- A decomposição de Shannon não precisa ser realizada em uma ordem específica.

- Repetindo o exemplo, decompondo em outra ordem. O que muda?

Implementação de circuitos lógicos usando MUX 2:1 – Exemplo

- Implemente usando MUX 2:1a função

$$f(x_3,x_2,x_1,x_0) = z = x_3\,(x_1 + x_2x_0)$$

- Mas agora decomponha na sequência em relação a x_1, x_0 e x_2

Implementação de circuitos lógicos usando MUX 2:1 – Exemplo

- Implemente usando MUX 2:1a função

$$f(x_3,x_2,x_1,x_0) = z = x_3\,(x_1+x_2x_0)$$

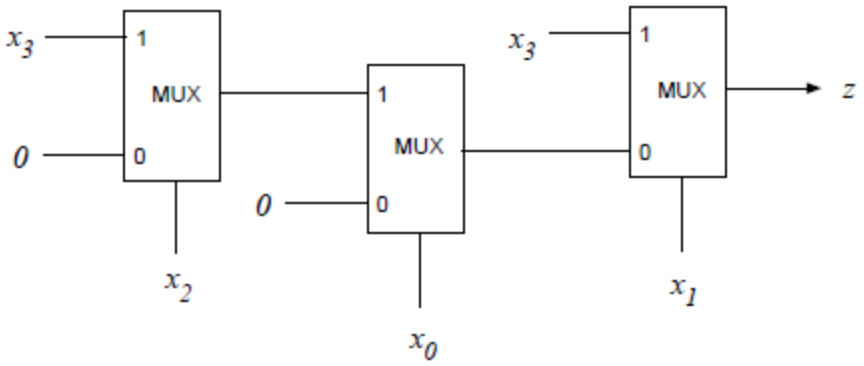
- Mas agora decomponha na sequência em relação a x_1, x_0 e x_2
  - $f(x_3,x_2,0,x_0) = x_3\,(0+x_2.x_0)= x_3\,(x_2.x_0)= x_3.x_2.x_0$
  - $f(x_3,x_2,1,x_0) = x_3\,(1+x_2.x_0)= x_3\,(1)= x_3$
  - $f(x_3,x_2,0,0) = x_3.x_2.0=0$
  - $f(x_3,x_2,0,1) = x_3.x_2.1= x_3.x_2$
  - $f(x_3,0,0,1) = x_3.0= 0$
  - $f(x_3,1,0,1) = x_3.1= x_3$

Implementação de circuitos lógicos usando MUX 2:1 – Exemplo

- Implemente usando MUX 2:1a função

  $f(x_3,x_2,x_1,x_0) = z = x_3\,(x_1+x_2x_0)$

- Mas agora decomponha na sequência em relação a x_1, x_0 e x_2
  - $f(x_3,x_2,0,x_0) = x_3\,(0+x_2.x_0) = x_3\,(x_2.x_0) = x_3.x_2.x_0$
  - $f(x_3,x_2,1,x_0) = x_3\,(1+x_2.x_0) = x_3\,(1) = x_3$
  - $f(x_3,x_2,0,0) = x_3.x_2.0 = 0$
  - $f(x_3,x_2,0,1) = x_3.x_2.1 = x_3.x_2$
  - $f(x_3,0,0,1) = x_3.0 = 0$
  - $f(x_3,1,0,1) = x_3.1 = x_3$



Implementação de circuitos lógicos usando MUX 2:1

- A implementação obtida depende da ordem em que a decomposição de Shannon é realizada
- Mas, não é possível saber qual a melhor ordem antecipadamente
- O bloco MUX 2:1 é usado em FPGAs para implementar funções combinacionais programação em campo

FIM