6 CARRY-LOOKAHEAD ADDERS AND VARIATIONS IN FAST ADDERS

Chapter Goals

Understand the carry-lookahead method and its many variations used in the design of fast adders

Study alternatives to the carry-lookahead method for designing fast adders

Chapter Highlights

Single- and multilevel carry lookahead
Various designs for log-time adders
Relating the carry determination problem to parallel prefix computation
Implementing fast adders in VLSI

CARRY-LOOKAHEAD ADDERS AND VARIATIONS IN FAST ADDERS: TOPICS

Topics in This Chapter
6.1 Unrolling the Carry Recurrence
6.2 Carry-Lookahead Adder Design
6.4 Carry Determination as Prefix Computation
6.5 Alternative Parallel Prefix Networks
7.3 Carry-Select Adders
7.4 Analysis of Carry Propagation
7.6 Modular Two-Operand Adders

6.1 UNROLLING THE CARRY RECURRENCE

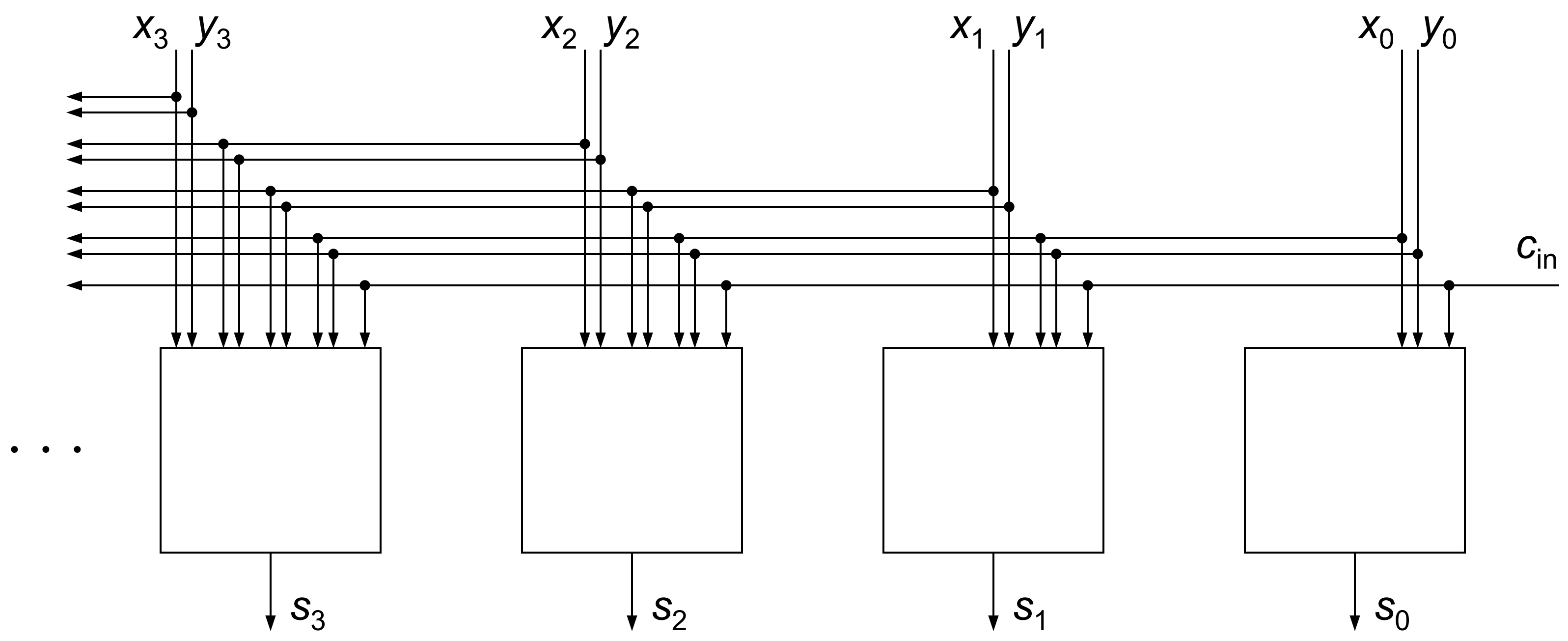
Recall the *generate*, *propagate*, *annihilate* (*absorb*), and *transfer* signals:

Signal	Binary
g_i	$x_i y_i$
p_i	$x_i \oplus y_i$
a_i	$x_i' y_i' = (x_i \vee y_i)'$
t_i	$x_i \vee y_i$
s_i	$x_i \oplus y_i \oplus c_i$

The carry recurrence can be unrolled to obtain each carry signal directly from inputs, rather than through propagation

$$\begin{aligned} c_i &= g_{i-1} \vee c_{i-1} p_{i-1} \\ &= g_{i-1} \vee (g_{i-2} \vee c_{i-2} p_{i-2}) p_{i-1} \\ &= g_{i-1} \vee g_{i-2} p_{i-1} \vee c_{i-2} p_{i-2} p_{i-1} \\ &= g_{i-1} \vee g_{i-2} p_{i-1} \vee g_{i-3} p_{i-2} p_{i-1} \vee c_{i-3} p_{i-3} p_{i-2} p_{i-1} \\ &= g_{i-1} \vee g_{i-2} p_{i-1} \vee g_{i-3} p_{i-2} p_{i-1} \vee g_{i-4} p_{i-3} p_{i-2} p_{i-1} \vee c_{i-4} p_{i-4} p_{i-3} p_{i-2} p_{i-1} \\ &= \ldots \end{aligned}$$

FULL CARRY LOOKAHEAD



Theoretically, it is possible to derive each sum digit directly from the inputs that affect it

Carry-lookahead adder design is simply a way of reducing the complexity of this ideal, but impractical, arrangement by hardware sharing among the various lookahead circuits

FOUR-BIT CARRY-LOOKAHEAD ADDER

Complexity reduced by deriving the carry-out indirectly

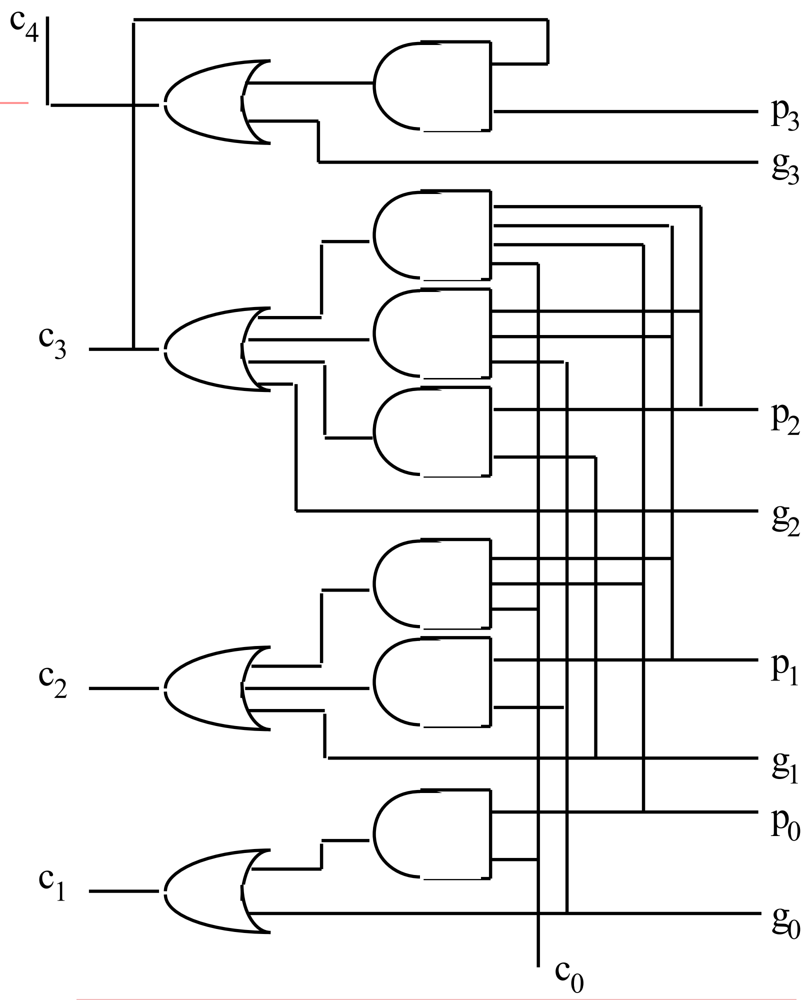


Fig. 6.1 Four-bit carry network with full lookahead.

Full carry lookahead is quite practical for a 4-bit adder

$$c_1 = g_0 \vee c_0 p_0$$
$$c_2 = g_1 \vee g_0 p_1 \vee c_0 p_0 p_1$$
$$c_3 = g_2 \vee g_1 p_2 \vee g_0 p_1 p_2 \vee c_0 p_0 p_1 p_2$$
$$c_4 = g_3 \vee g_2 p_3 \vee g_1 p_2 p_3 \vee g_0 p_1 p_2 p_3 \vee c_0 p_0 p_1 p_2 p_3$$

CARRY LOOKAHEAD BEYOND 4 BITS

Consider a 32-bit adder

No circuit sharing:
Repeated computations

$c_1 = g_0 \vee c_0 p_0$

$c_2 = g_1 \vee g_0 p_1 \vee c_0 p_0 p_1$

$c_3 = g_2 \vee g_1 p_2 \vee g_0 p_1 p_2 \vee c_0 p_0 p_1 p_2$

.
.
.

32-input AND

$c_{31} = g_{30} \vee g_{29} p_{30} \vee g_{28} p_{29} p_{30} \vee g_{27} p_{28} p_{29} p_{30} \vee \ . \ . \ . \ \vee c_0 p_0 p_1 p_2 p_3 \ldots p_{29} p_{30}$

. . .

32-input OR

High fan-ins necessitate tree-structured circuits

ONE SOLUTION TO THE FAN-IN PROBLEM

Multilevel lookahead

Example: 16-bit addition

Radix-16 (four digits)

Two-level carry lookahead (four 4-bit blocks)

Either way, the carries c_4, c_8, and c_{12} are determined first

c_{16}	c_{15}	c_{14}	c_{13}	c_{12}	c_{11}	c_{10}	c_9	c_8	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
c_{out}				?				?				?				c_{in}

6.2 CARRY-LOOKAHEAD ADDER DESIGN

Block *generate* and *propagate* signals

$$g_{[i,i+3]} = g_{i+3} \vee g_{i+2}p_{i+3} \vee g_{i+1}p_{i+2}p_{i+3} \vee g_i\, p_{i+1}p_{i+2}p_{i+3}$$
$$p_{[i,i+3]} = p_i\, p_{i+1}p_{i+2}p_{i+3}$$

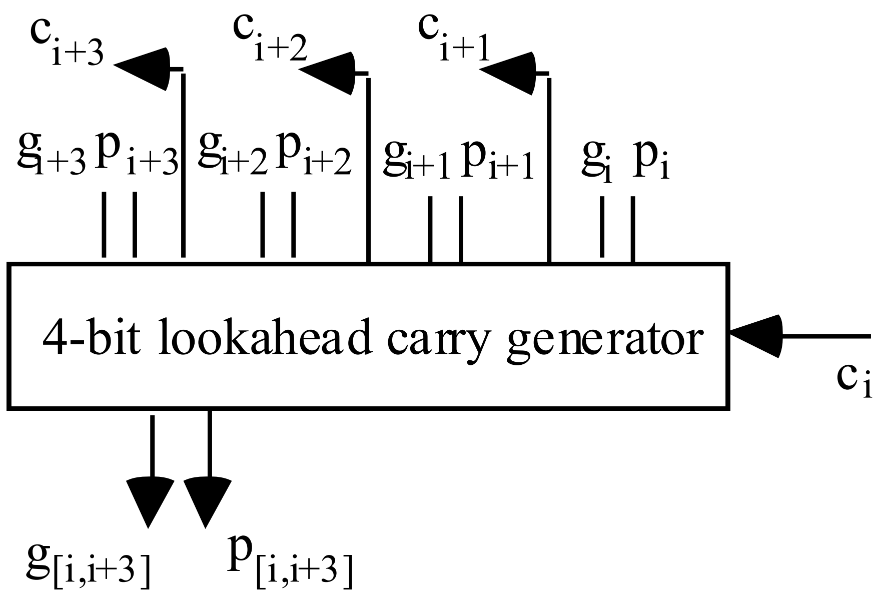


Fig. 6.2b Schematic diagram of a 4-bit lookahead carry generator.

A BUILDING BLOCK FOR CARRY-LOOKAHEAD ADDITION

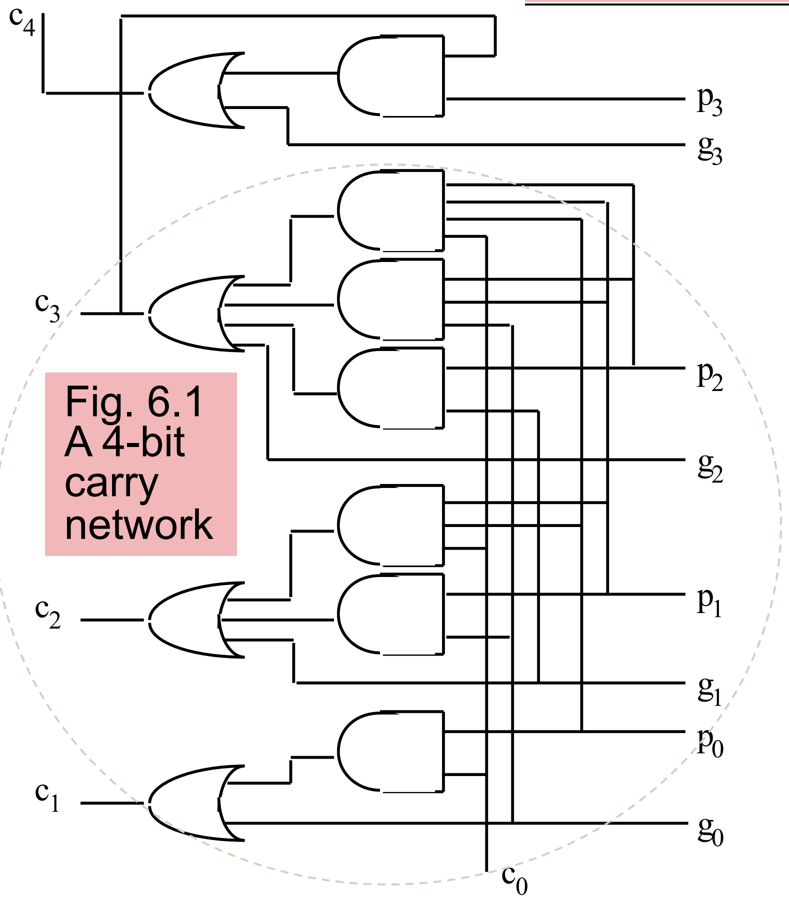
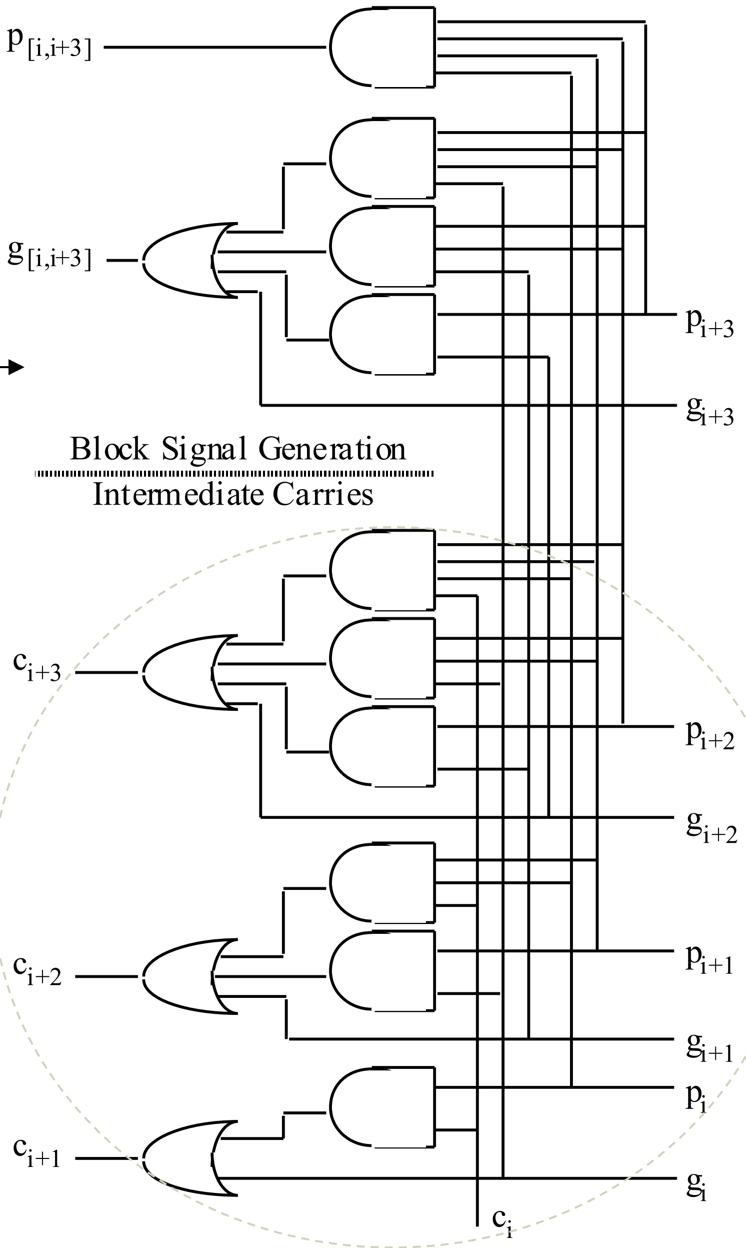


Fig. 6.1 A 4-bit carry network

Fig. 6.2a A 4-bit lookahead carry generator



A Two-Level Carry-Lookahead Adder

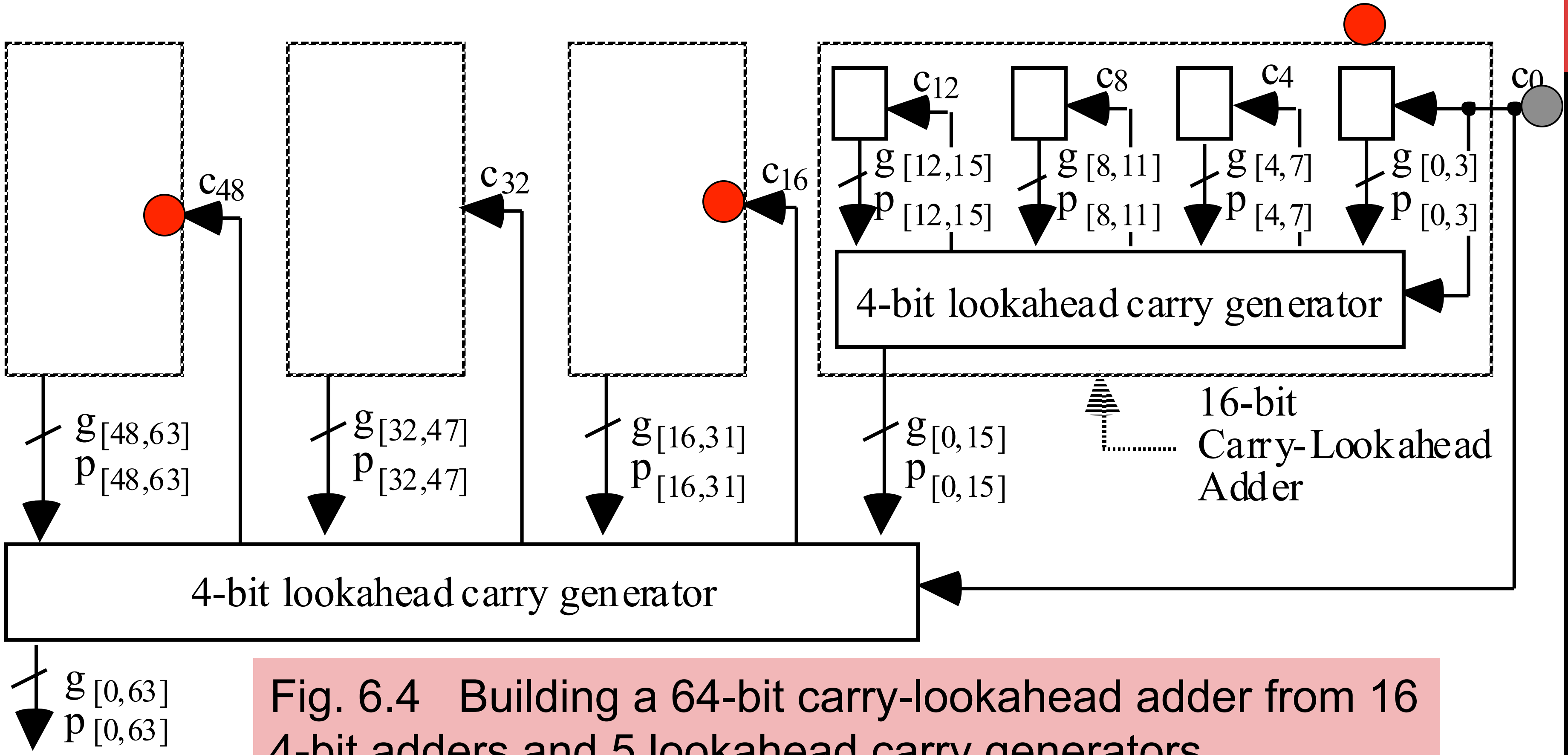


Fig. 6.4 Building a 64-bit carry-lookahead adder from 16 4-bit adders and 5 lookahead carry generators.

Carry-out: $c_{out} = g_{[0,k-1]} \vee c_0\, p_{[0,k-1]} = x_{k-1}y_{k-1} \vee s_{k-1}'\,(x_{k-1} \vee y_{k-1})$

6.4 CARRY DETERMINATION AS PREFIX COMPUTATION

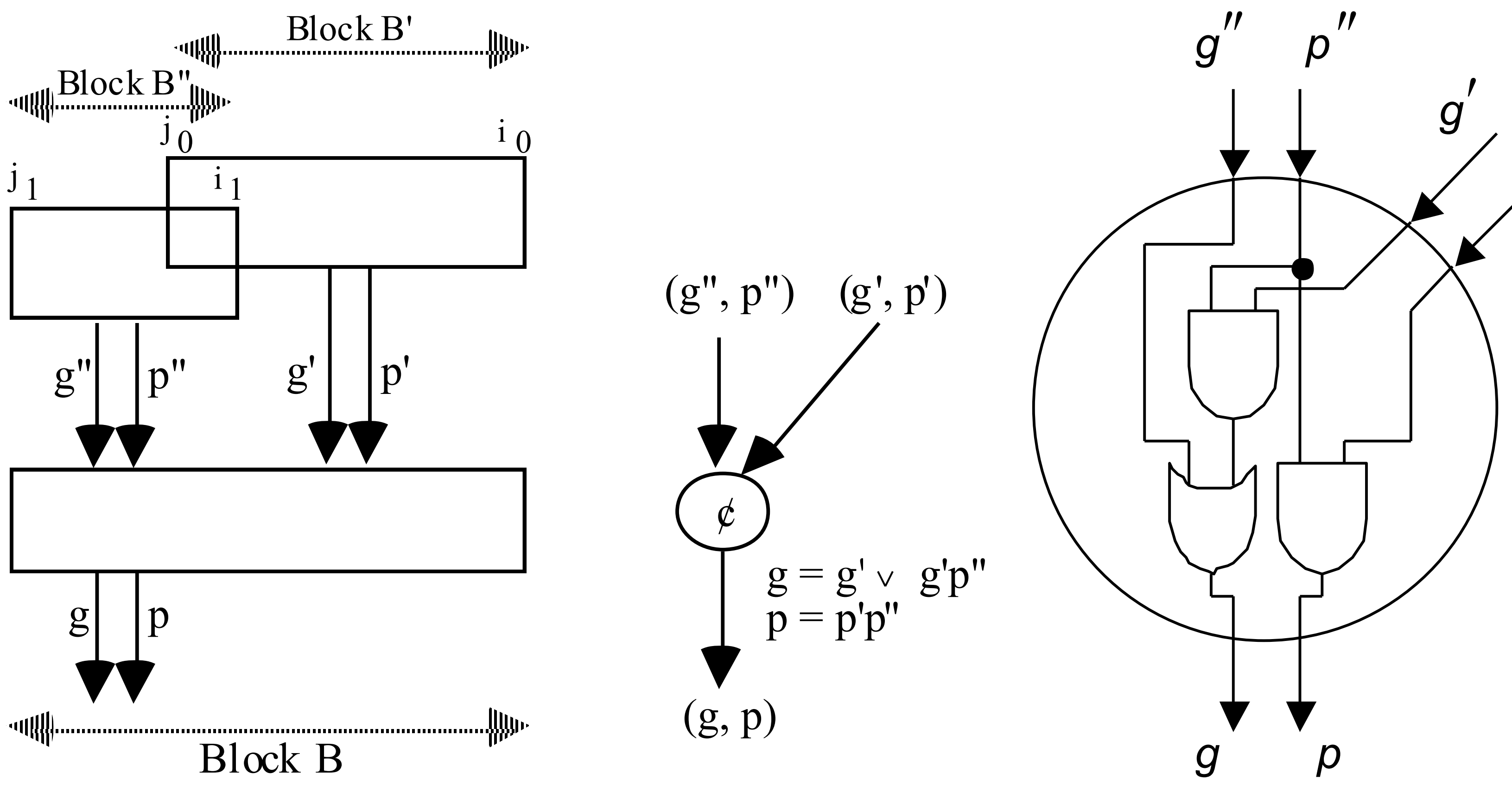


Fig. 6.5 Combining of *g* and *p* signals of two (contiguous or overlapping) blocks B' and B" of arbitrary widths into the *g* and *p* signals for block B.

FORMULATING THE PREFIX COMPUTATION PROBLEM

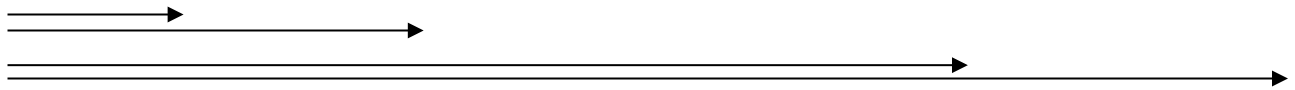
The problem of carry determination can be formulated as:

$$\begin{array}{lllll}
\text{Given} & (g_0, p_0) & (g_1, p_1) \quad . \; . \; . & (g_{k-2}, p_{k-2}) & (g_{k-1}, p_{k-1}) \\
\text{Find} & (g_{[0,0]}, p_{[0,0]}) & (g_{[0,1]}, p_{[0,1]}) \; . \; . \; . & (g_{[0,k-2]}, p_{[0,k-2]}) & (g_{[0,k-1]}, p_{[0,k-1]}) \\
 & \downarrow & \downarrow & \downarrow & \downarrow \\
 & c_1 & c_2 \quad . \; . \; . & c_{k-1} & c_k
\end{array}$$

Carry-in can be viewed as an extra (-1) position: $(g_{-1}, p_{-1}) = (c_{\text{in}}, 0)$

The desired pairs are found by evaluating all prefixes of

$$(g_0, p_0) \;¢\; (g_1, p_1) \;¢\; . \; . \; . \;¢\; (g_{k-2}, p_{k-2}) \;¢\; (g_{k-1}, p_{k-1})$$



The carry operator ¢ is associative, but not commutative

$$[(g_1, p_1) \;¢\; (g_2, p_2)] \;¢\; (g_3, p_3) = (g_1, p_1) \;¢\; [(g_2, p_2) \;¢\; (g_3, p_3)]$$

Prefix sums analogy:

$$\begin{array}{llllll}
\text{Given} & x_0 & x_1 & x_2 & . \; . \; . & x_{k-1} \\
\text{Find} & x_0 & x_0+x_1 & x_0+x_1+x_2 & . \; . \; . & x_0+x_1+\ldots+x_{k-1}
\end{array}$$

6.5 ALTERNATIVE PARALLEL PREFIX NETWORKS

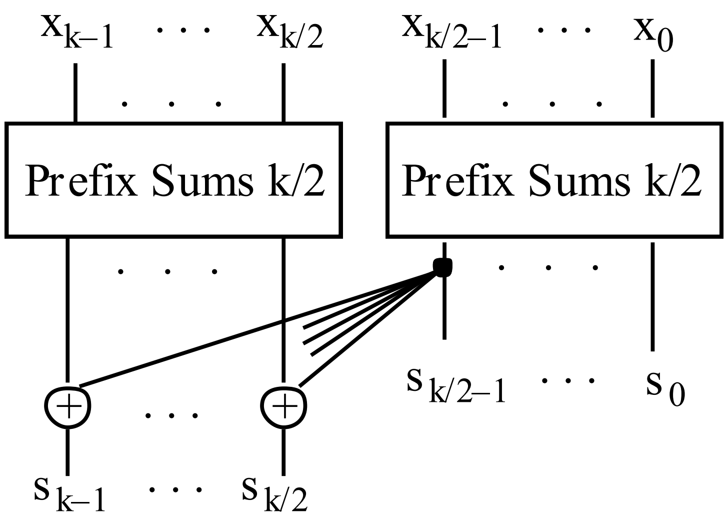
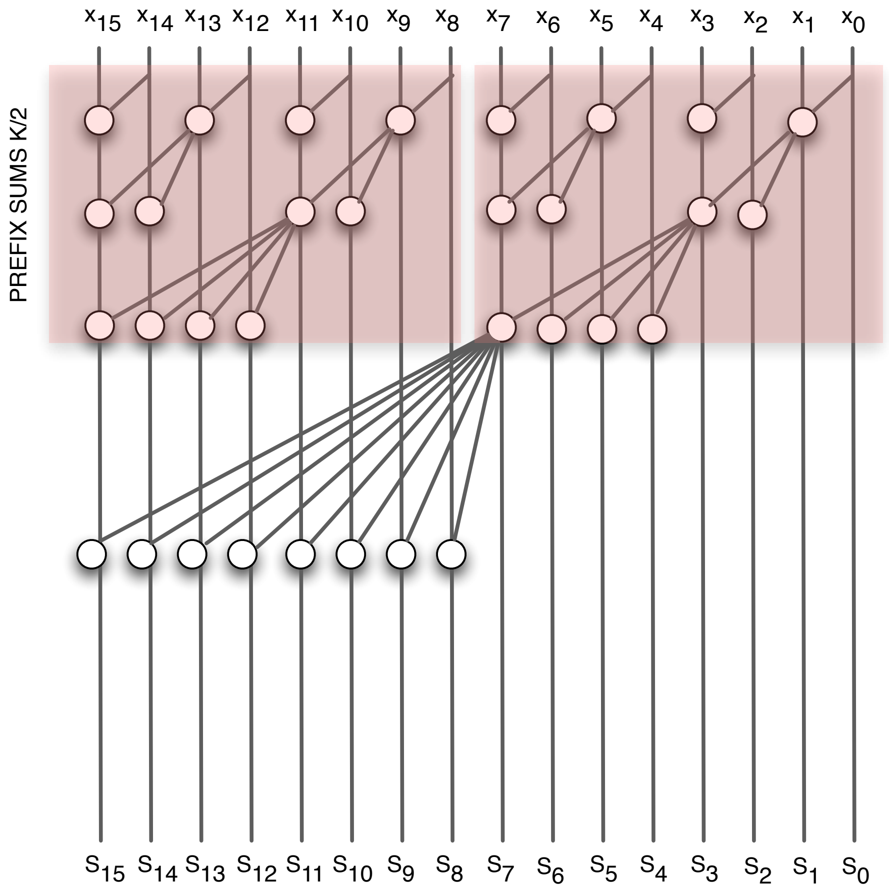


Fig. 6.7 Ladner-Fischer parallel prefix sums network built of two *k*/2-input networks and *k*/2 adders.

Delay recurrence $D(k) = D(k/2) + 1 = \log_2 k$

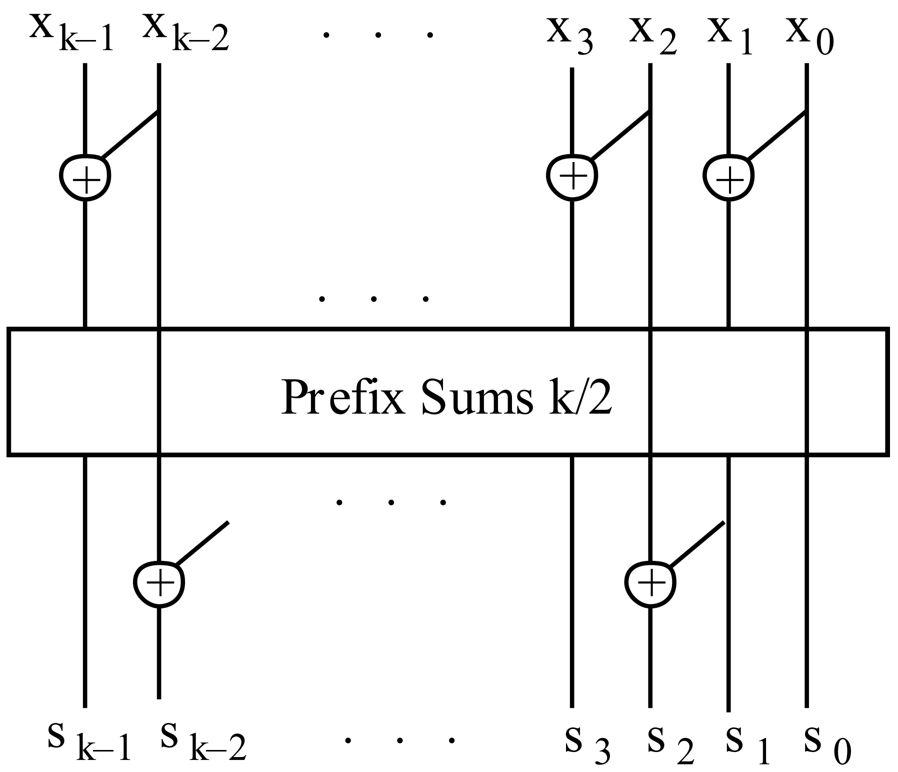
Cost recurrence $C(k) = 2C(k/2) + k/2 = (k/2) \log_2 k$



x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
PREFIX SUMS K/2
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

LADNER-FISHER
DELAY=4
COST=32
COST x DELAY=128

THE BRENT-KUNG RECURSIVE CONSTRUCTION

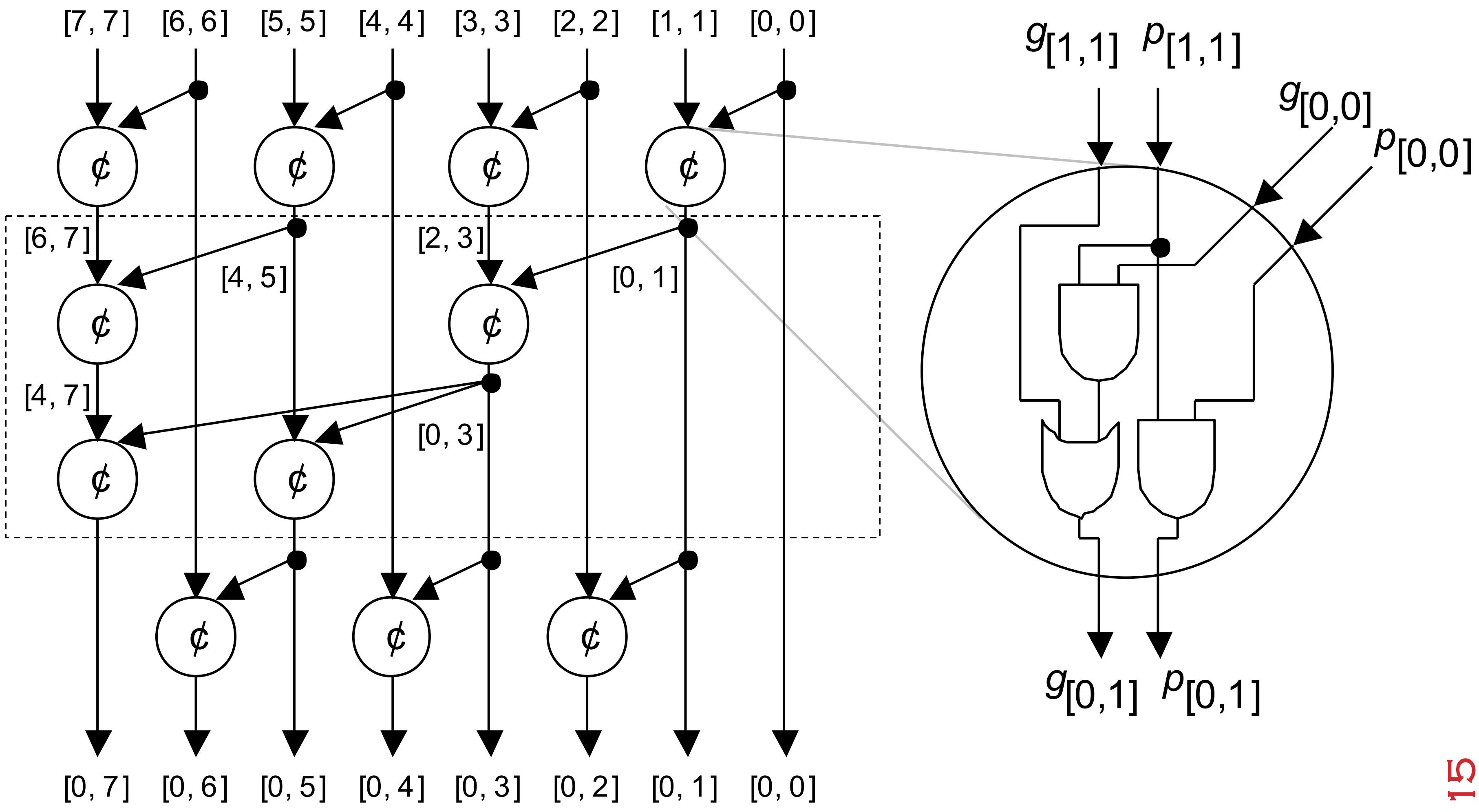


x_{k-1} x_{k-2} . . . x_3 x_2 x_1 x_0
Prefix Sums k/2
s_{k-1} s_{k-2} . . . s_3 s_2 s_1 s_0

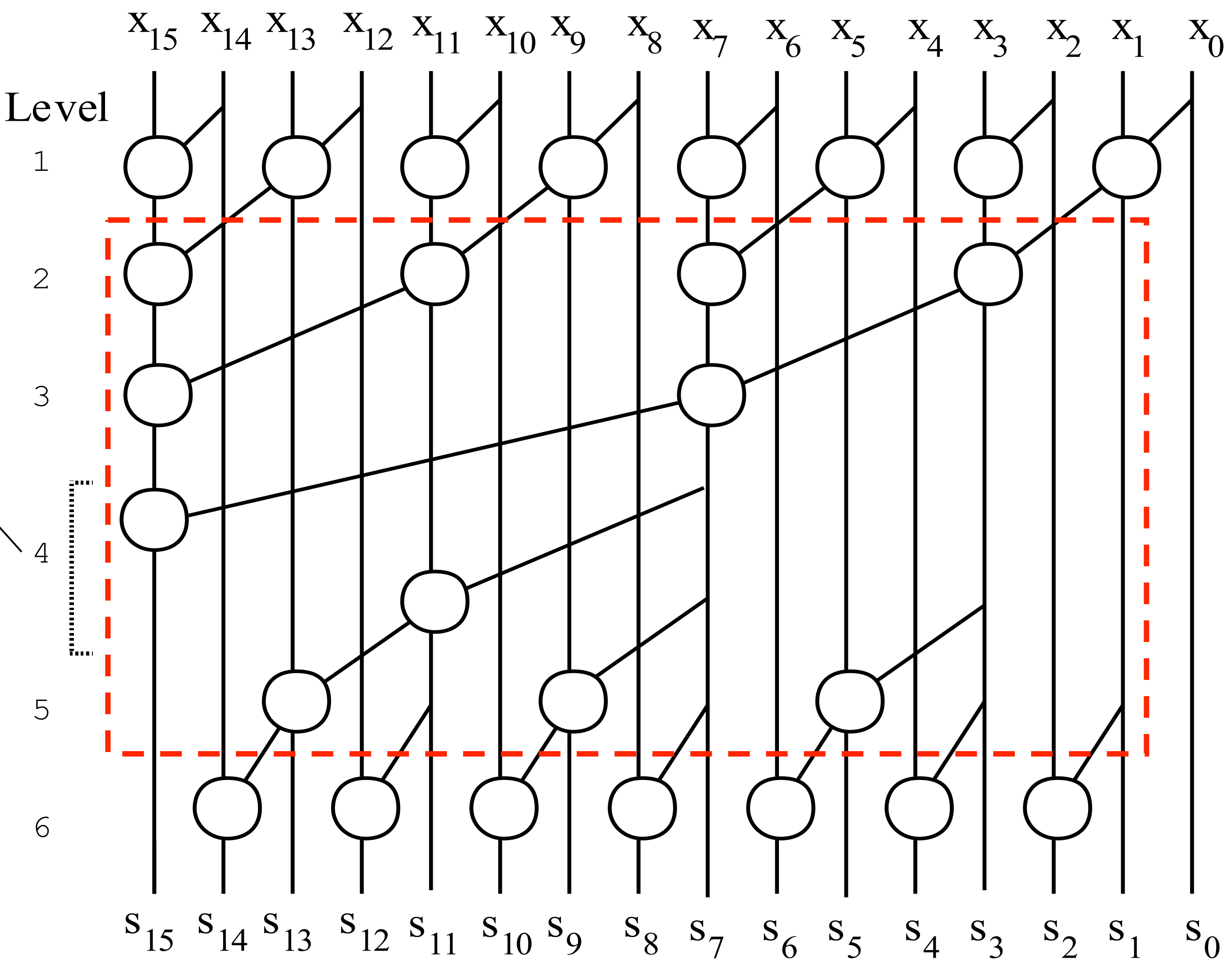
Fig. 6.8 Parallel prefix sums network built of one *k*/2-input network and *k* – 1 adders.

Delay recurrence	$D(k) = D(k/2) + 2 = 2 \log_2 k - 1$ (–2 really)
Cost recurrence	$C(k) = C(k/2) + k - 1 = 2k - 2 - \log_2 k$

BRENT-KUNG CARRY NETWORK (8-BIT ADDER)



BRENT-KUNG CARRY NETWORK (16-BIT ADDER)



x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
Level
1
2
3
4
5
6
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Reason for latency being $2 \log_2 k - 2$

Fig. 6.9 Brent-Kung parallel prefix graph for 16 inputs.

KOGGE-STONE CARRY NETWORK (16-BIT ADDER)

Cost formula

$C(k) = (k - 1)$
$\quad + (k - 2)$
$\quad + (k - 4) + \ldots$
$\quad + (k - k/2)$
$= k \log_2 k - k + 1$

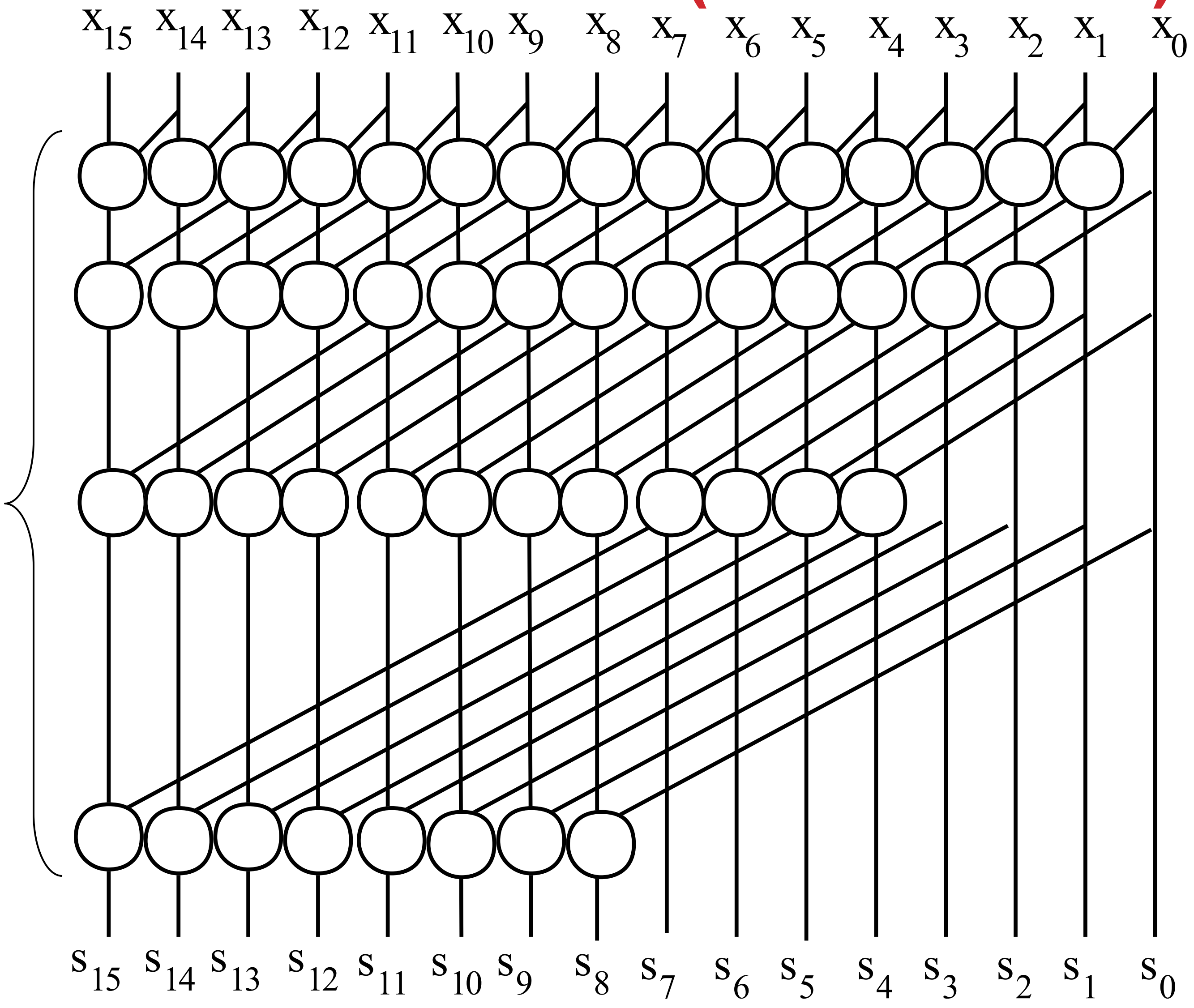
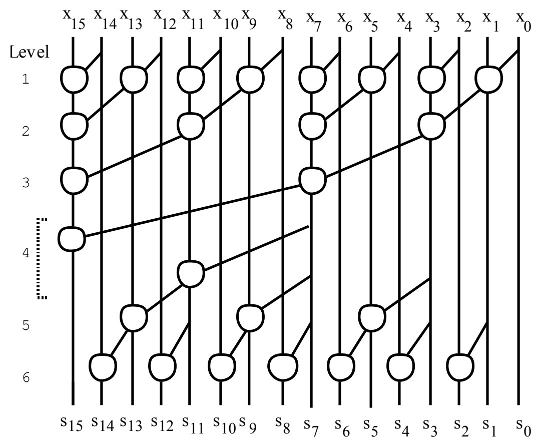
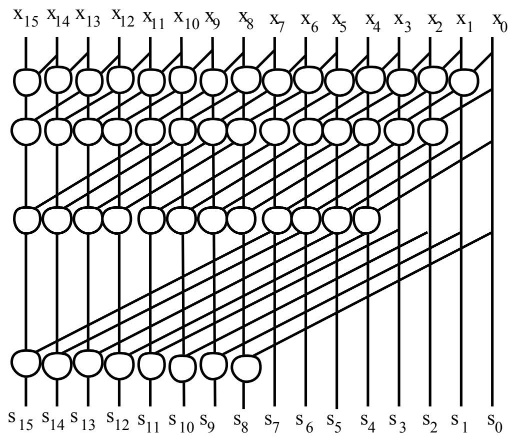

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
$\log_2 k$ levels (minimum possible)
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Fig. 6.10 Kogge-Stone parallel prefix graph for 16 inputs.

HYBRID B-K/K-S (HAN CARLSON) CARRY NETWORK (16-BIT)

Brent-Kung:
6 levels
26 cells


x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
Level
1
2
3
4
5
6
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0


x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Kogge-Stone:
4 levels
49 cells

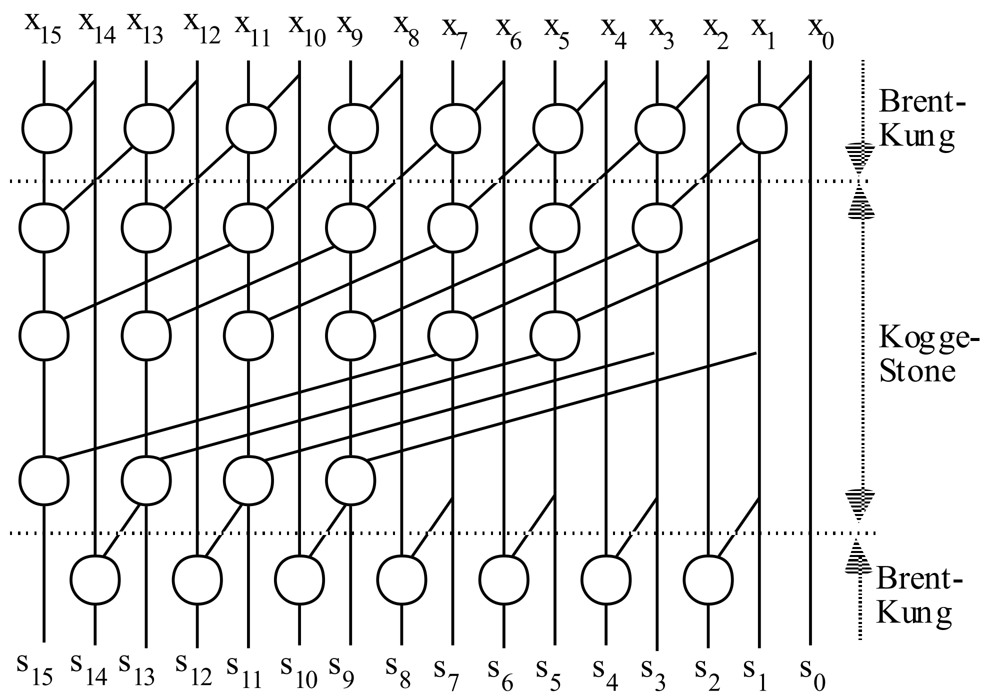

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
Brent-Kung
Kogge-Stone
Brent-Kung
s_{15} s_{14} s_{13} s_{12} s_{11} s_{10} s_9 s_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Fig. 6.11
A Hybrid Brent-Kung/ Kogge-Stone parallel prefix graph for 16 inputs.

Hybrid:
5 levels
32 cells

SPEED-COST TRADEOFFS IN CARRY NETWORKS

Method	Delay	Cost
Ladner-Fischer	$\log_2 k$	$(k/2)\log_2 k$
Kogge-Stone	$\log_2 k$	$k\log_2 k - k + 1$
Brent-Kung	$2\log_2 k - 2$	$2k - 2 - \log_2 k$
Han Carlson	$\text{Log}_2 k\ +1$	$(k/2)\log_2 k$

k= 16-bits

Method	Delay	Cost
Ladner-Fischer	4	32
Kogge-Stone	4	*49*
Brent-Kung	6	*26*
Han Carlson	5	32

PROBLEMAS

Problema 6.1 Obtenha os gráficos para valores pares $n \in [4, 32]$ dos somadores Ladner-Fisher (LF), Brent-Kung (BK), Kogge-Stone (KS), e Han-Carlson (HC) para:

a) Área.

b) Atraso.

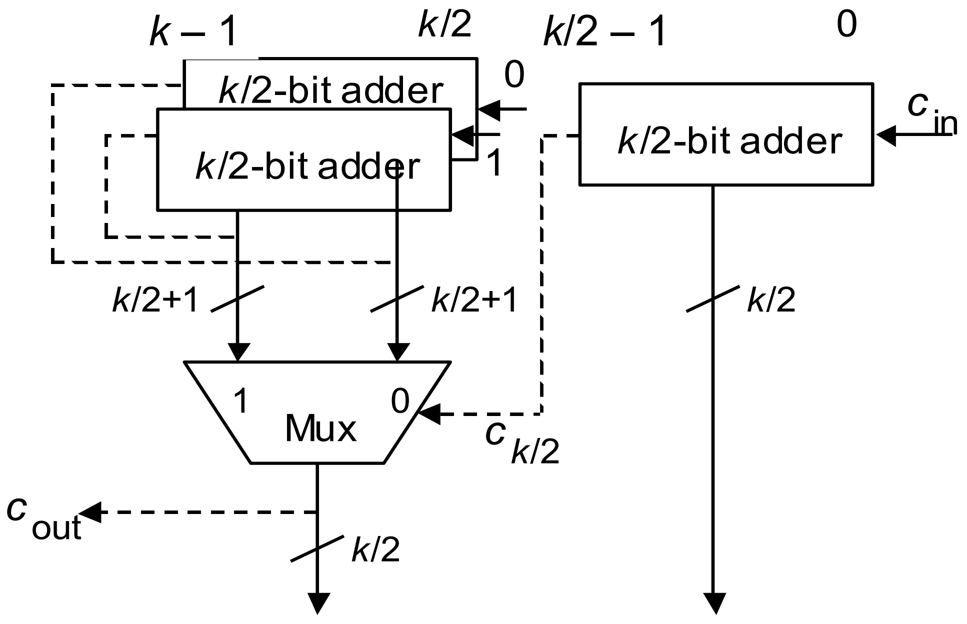
c) Produto área atraso (AT)

d) Produto área atraso quadrado (AT^2).

e) Fan-out.

Indique qual é a melhor opção para as diferentes figuras de mérito apresentadas acima.

7.3 Carry-Select Adders



High $k/2$ bits   Low $k/2$ bits

Fig. 7.9 Carry-select adder for k-bit numbers built from three $k/2$-bit adders.

$$C_{\text{select-add}}(k) = 3C_{\text{add}}(k/2) + k/2 + 1$$

$$T_{\text{select-add}}(k) = T_{\text{add}}(k/2) + 1$$

MULTILEVEL CARRY-SELECT ADDERS

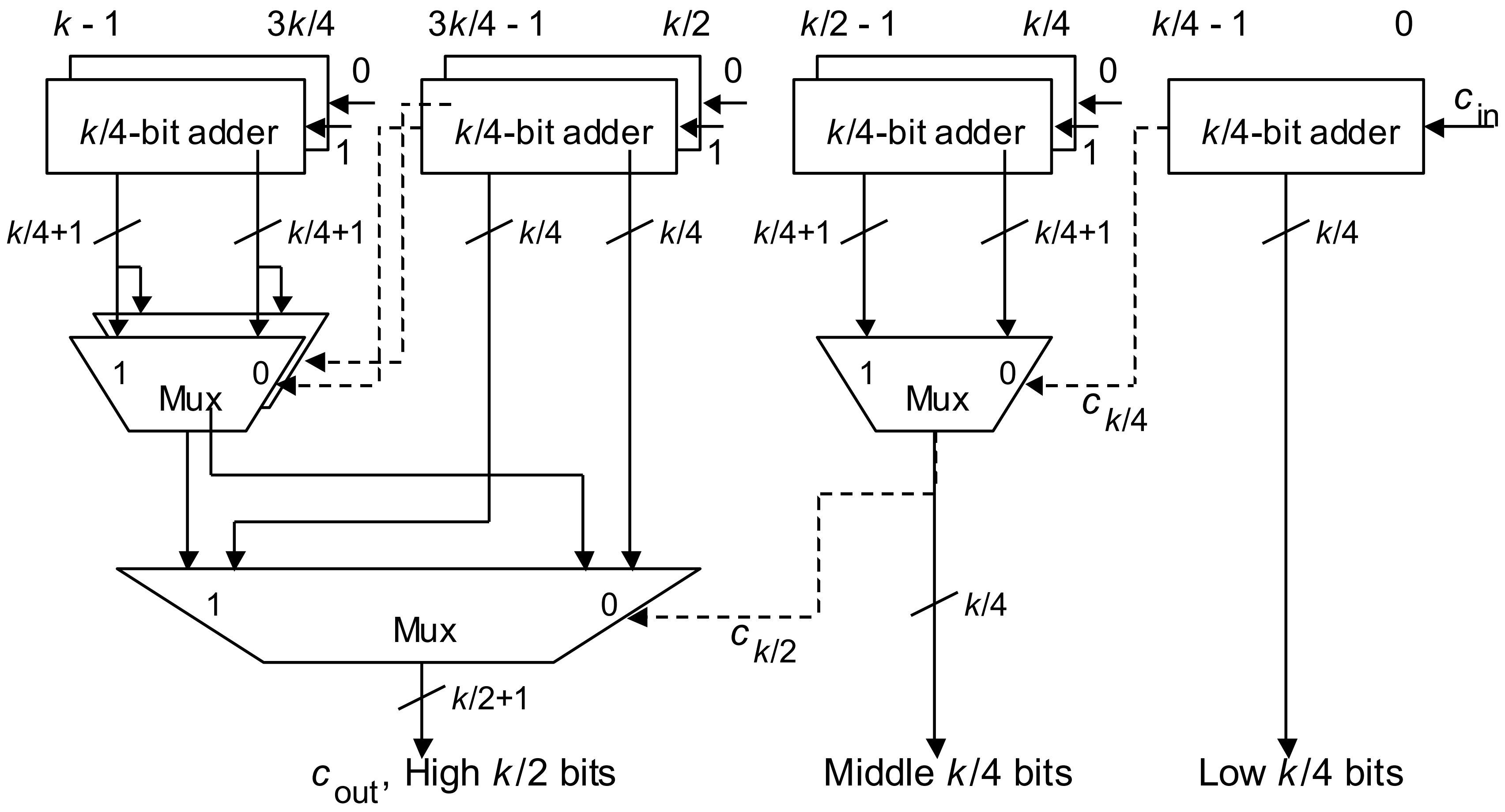


Fig. 7.10 Two-level carry-select adder built of $k/4$-bit adders.

PROBLEMAS

Problema 6.2 Faça uma optimização para um *Carry-Select-Adder* de 64-bits dividido em 4 somas e assumindo que a área e o atraso do somador está expressado como $n \times A_{adder}$, $n \times T_{adder}$, respectivamente, sendo n o número de bits. Considere a área e atraso do multiplexador 2:1 como uma unidade $(2/3) \times A_{adder}$, $(2/3) \times T_{adder}$.

7.4 ANALYSIS OF CARRY PROPAGATION

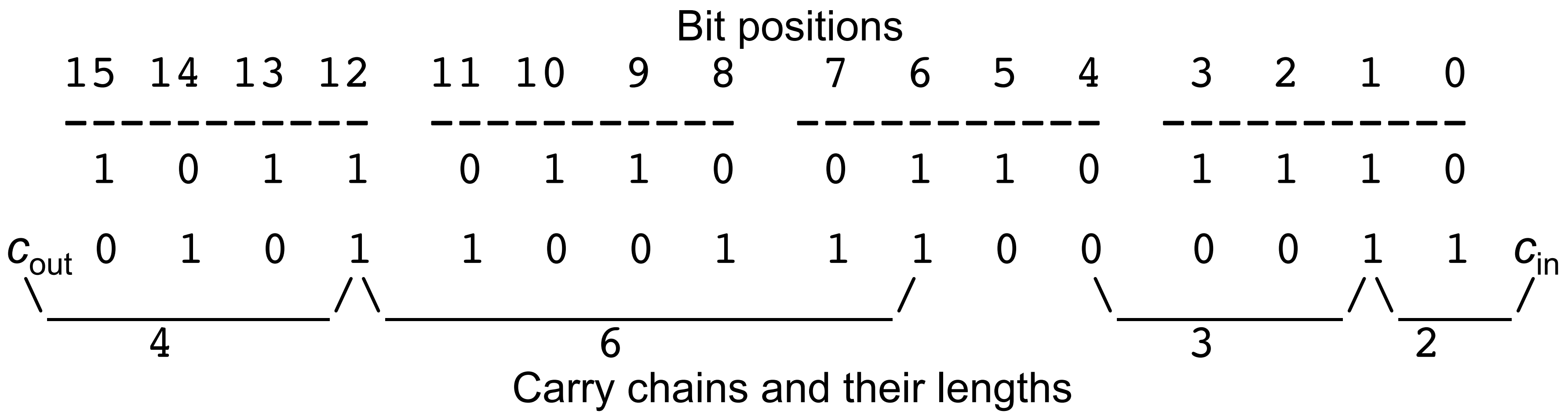


Fig. 7.11 Example addition and its carry propagation chains.

Given binary numbers with random bits, for each position *i* we have

Probability of carry generation	=	¼	(both 1s)
Probability of carry annihilation	=	¼	(both 0s)
Probability of carry propagation	=	½	(different)

PROBLEMAS

Problema 6.3. Uma entrada fixa A = 31727_{10} deve ser somada com entradas variáveis B e C, todas de 16-bits. Ditas entradas variáveis só podem ter os seguintes valores:

- B = {2638_{10}, 31439_{10}, 14923_{10}}.
- C = {3041_{10}, 15343_{10}, 3192_{10}}.

Qual das somas pode ser implementada com um atraso menor?

Problema 6.4. Dois vetores de 12 bits A e B precisam de ser somados, onde A é sempre múltiplo $100_{(10)}$ e B múltiplo de $48_{(10)}$. Considerando os tempos de atraso do problema 6.2, obtenha a soma dos vectores com um atraso máximo de $5 \times T_{adder}$.

7.6 MODULAR TWO-OPERAND ADDERS

mod-2^k: Ignore carry out of position $k - 1$

mod-$(2^k - 1)$: Use End-Around Carry (EAC)

mod-$(2^k + 1)$: Use Inverted End-Around Carry (IEAC)

Number	Std. binary
0	0 0 . . . 0 0 0
1	0 0 . . . 0 0 1
2	0 0 . . . 0 1 0
.	.
.	.
.	.
2^k-1	0 1 . . . 1 1 1
2^k	1 0 . . . 0 0 0

GENERAL MODULAR ADDERS

($x + y$) mod m

if $x + y \geq m$
then $x + y - m$
else $x + y$

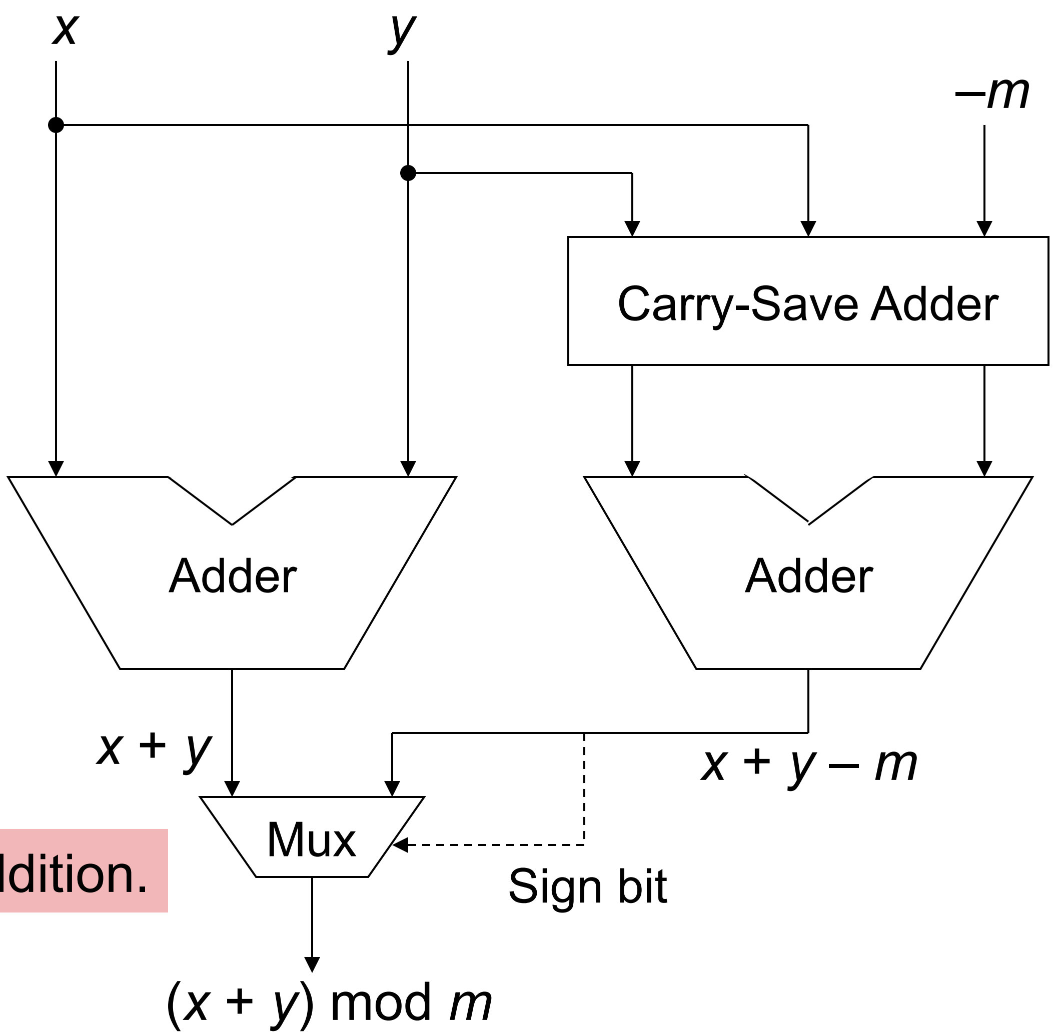


Fig. 7.15 Fast modular addition.

PROBLEMAS

Problema 6.5. Implemente os seguintes somadores modulares:

a) $|A+B|_{29}$.

b) $||A+B|_{27}+C|_{29}$.

c) $||A+B|_{11}+C|_{13}$.

d) $|A+B|_{59}$

e) $|A+B|_{15}$