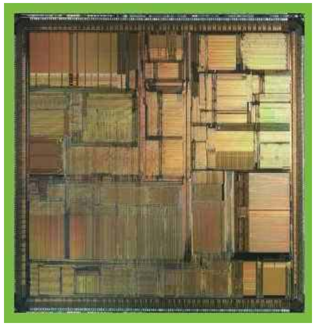
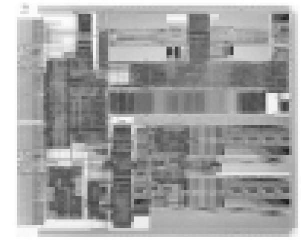
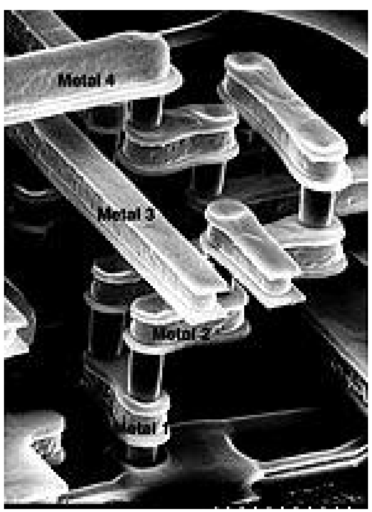




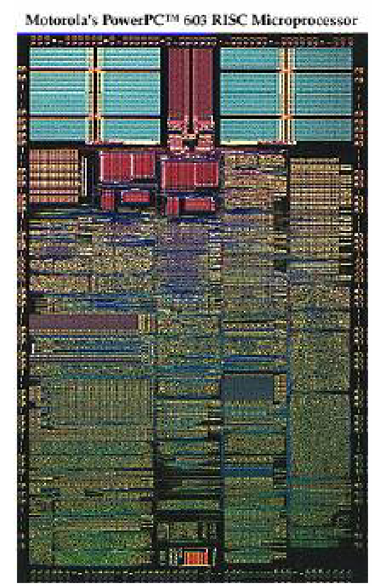
VLSI Arithmetic

Lecture 9:
Multipliers



Prof. Vojin G. Oklobdzija
University of California

http://www.ece.ucdavis.edu/acsel



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

lap

Multiplication Algorithm*

Notation for our discussion of multiplication algorithms:

a	Multiplicand	$a_{k-1}a_{k-2} \cdots a_1a_0$
x	Multiplier	$x_{k-1}x_{k-2} \cdots x_1x_0$
p	Product ($a \times x$)	$p_{2k-1}p_{2k-2} \cdots p_1p_0$

Initially, we assume unsigned operands

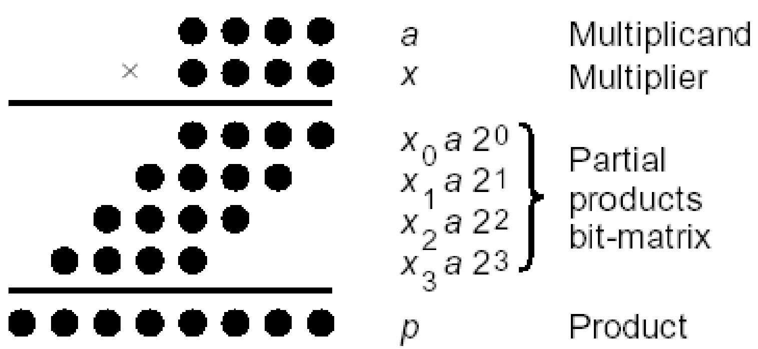


Fig. 9.1 Multiplication of two 4-bit unsigned binary numbers in dot notation.

**from Parhami*

Multiplication Algorithm*

Multiplication with right shifts: top-to-bottom accumulation

$p^{(j+1)} = (p^{(j)} + x_j\, a\, 2^k)\, 2^{-1}$ with $p^{(0)} = 0$ and

|——add——|

|—shift right—|

$p^{(k)} = p = ax + p^{(0)} 2^{-k}$

Multiplication with left shifts: bottom-to-top accumulation

$p^{(j+1)} = 2p^{(j)} + x_{k-j-1} a$ with $p^{(0)} = 0$ and

|shift|

|——add——|

$p^{(k)} = p = ax + p^{(0)} 2^{k}$

**from Parhami*

Multiplication Algorithm*

Right-shift algorithm

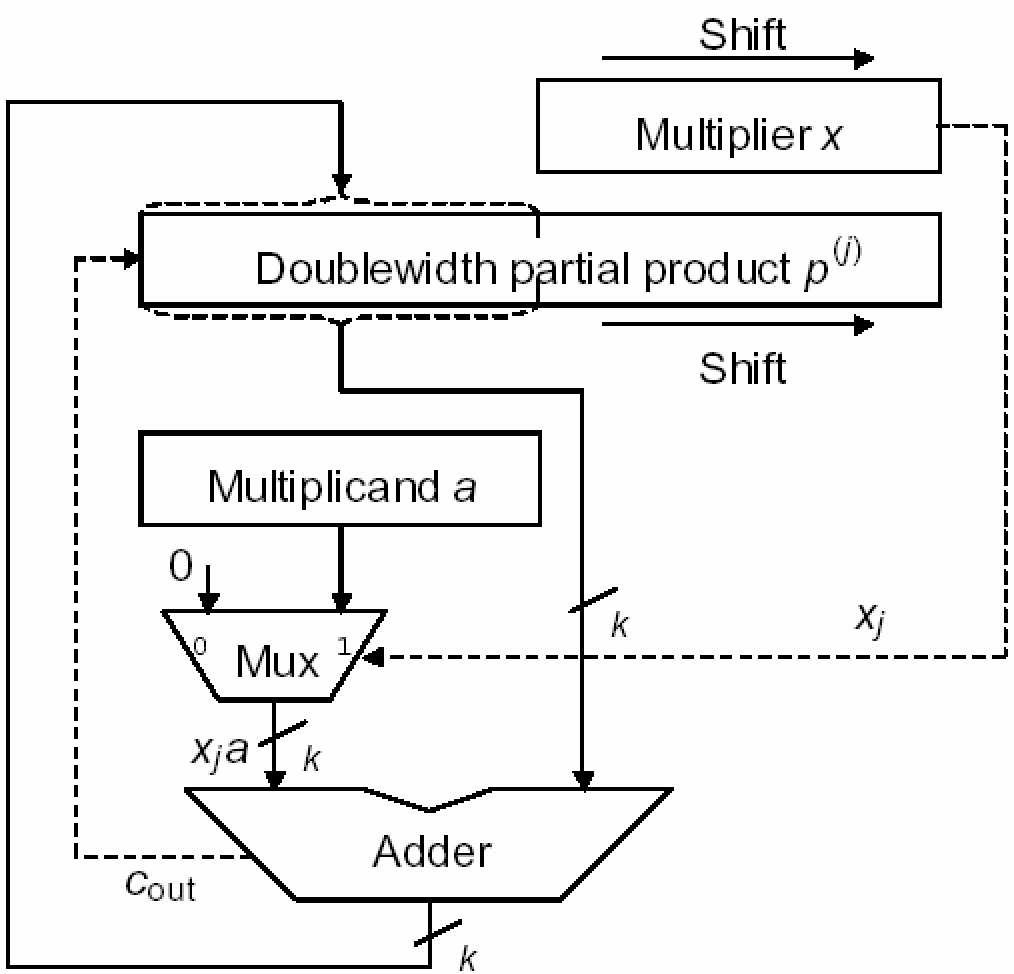
a	1 0 1 0
x	1 0 1 1
$p^{(0)}$	0 0 0 0
$+x_0a$	1 0 1 0
$2p^{(1)}$	0 1 0 1 0
$p^{(1)}$	0 1 0 1 0
$+x_1a$	1 0 1 0
$2p^{(2)}$	0 1 1 1 1 0
$p^{(2)}$	0 1 1 1 1 0
$+x_2a$	0 0 0 0
$2p^{(3)}$	0 0 1 1 1 1 0
$p^{(3)}$	0 0 1 1 1 1 0
$+x_3a$	1 0 1 0
$2p^{(4)}$	0 1 1 0 1 1 1 0
$p^{(4)}$	0 1 1 0 1 1 1 0

Left-shift algorithm

a	1 0 1 0
x	1 0 1 1
$p^{(0)}$	0 0 0 0
$2p^{(0)}$	0 0 0 0 0
$+x_3a$	1 0 1 0
$p^{(1)}$	0 1 0 1 0
$2p^{(1)}$	0 1 0 1 0 0
$+x_2a$	0 0 0 0
$p^{(2)}$	0 1 0 1 0 0
$2p^{(2)}$	0 1 0 1 0 0 0
$+x_1a$	1 0 1 0
$p^{(3)}$	0 1 1 0 0 1 0
$2p^{(3)}$	0 1 1 0 0 1 0 0
$+x_0a$	1 0 1 0
$p^{(4)}$	0 1 1 0 1 1 1 0

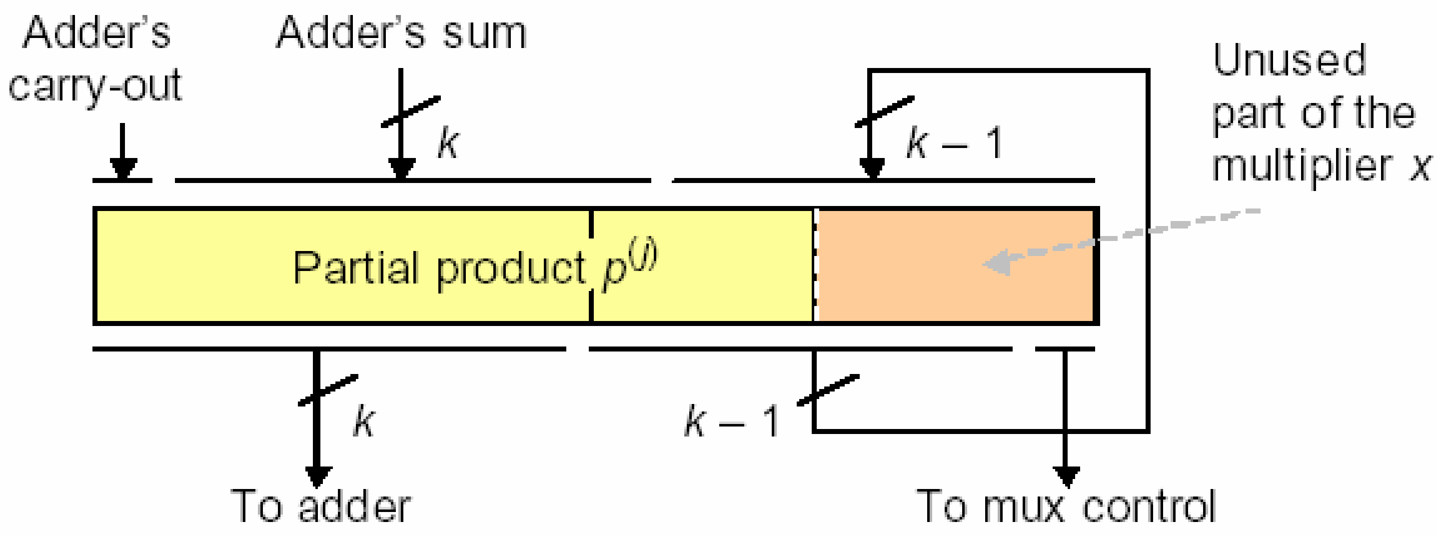
**from Parhami*

Basic Hardware Multipliers



Hardware realization of the sequential multiplication algorithm with additions and right shifts. **from Parhami*

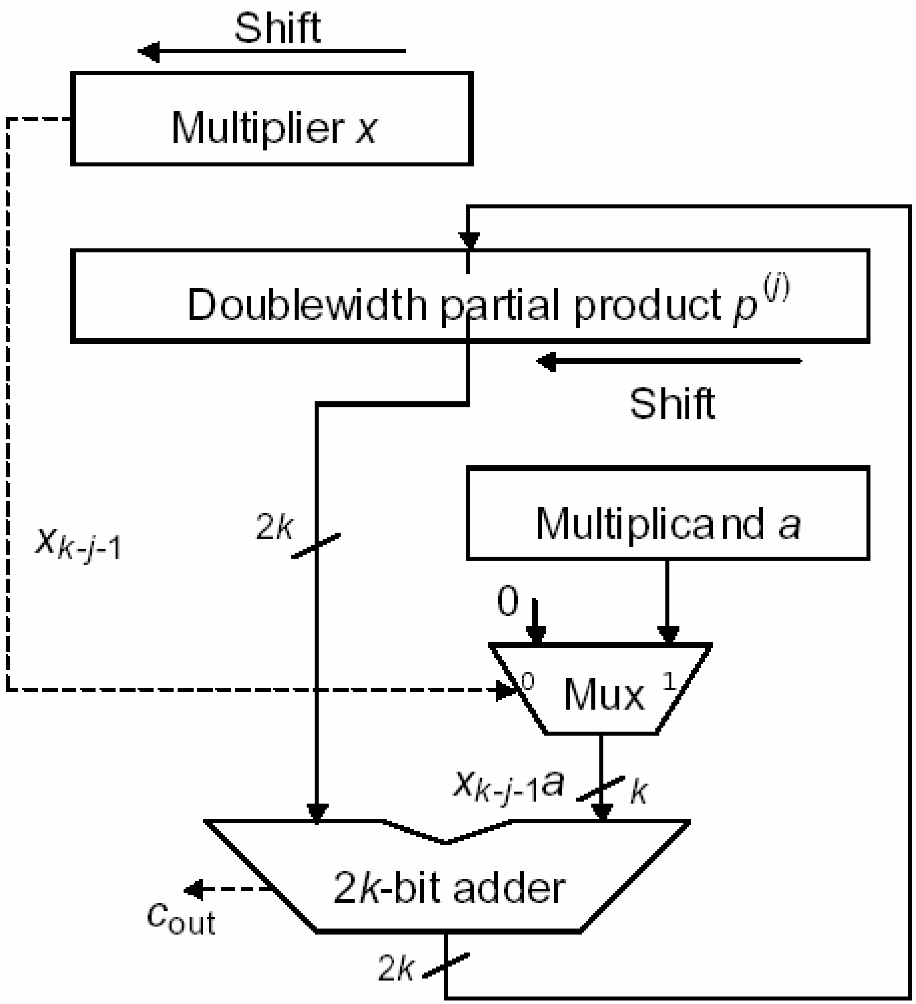
Multiplication*



Combining the loading and shifting of the double-width register holding the partial product and the partially used multiplier.

*from Parhami

Multiplication*



Hardware realization of the sequential multiplication algorithm with left shifts and additions.

**from Parhami*

Multiplication of Signed Numbers

a		1	0	1	1	0					
x		0	1	0	1	1					
$p^{(0)}$		0	0	0	0	0					
$+x_0a$		1	0	1	1	0					
$2p^{(1)}$	1	1	0	1	1	0					
$p^{(1)}$		1	1	0	1	1	0				
$+x_1a$		1	0	1	1	0					
$2p^{(2)}$	1	1	0	0	0	1	0				
$p^{(2)}$		1	1	0	0	0	1	0			
$+x_2a$		0	0	0	0	0					
$2p^{(3)}$	1	1	1	0	0	0	1	0			
$p^{(3)}$		1	1	1	0	0	0	1	0		
$+x_3a$		1	0	1	1	0					
$2p^{(4)}$	1	1	0	0	1	0	0	1	0		
$p^{(4)}$		1	1	0	0	1	0	0	1	0	
$+x_4a$		0	0	0	0	0					
$2p^{(5)}$	1	1	1	0	0	1	0	0	1	0	
$p^{(5)}$		1	1	1	0	0	1	0	0	1	0

*from Parhami

Sequential multiplication of 2's-complement numbers with right shifts (positive multiplier).

a	1 0 1 1 0	
x	1 0 1 0 1	
$p^{(0)}$	0 0 0 0 0	
$+x_0a$	1 0 1 1 0	
$2p^{(1)}$	1 1 0 1 1 0	
$p^{(1)}$	1 1 0 1 1	0
$+x_1a$	0 0 0 0 0	
$2p^{(2)}$	1 1 1 0 1 1	0
$p^{(2)}$	1 1 1 0 1	1 0
$+x_2a$	1 0 1 1 0	
$2p^{(3)}$	1 1 0 0 1 1	1 0
$p^{(3)}$	1 1 0 0 1	1 1 0
$+x_3a$	0 0 0 0 0	
$2p^{(4)}$	1 1 1 0 0 1	1 1 0
$p^{(4)}$	1 1 1 0 0	1 1 1 0
$+(-x_4a)$	0 1 0 1 0	
$2p^{(5)}$	0 0 0 1 1 0	1 1 1 0
$p^{(5)}$	0 0 0 1 1	0 1 1 1 0

*from Parhami

Sequential multiplication of 2's-complement numbers with right shifts (negative multiplier).

Multiplier Recoding*

**from Parhami*

Table 9.1 Radix-2 Booth's recoding

x_i	x_{i-1}	y_i	Explanation
0	0	0	No string of 1s in sight
0	1	1	End of string of 1s in x
1	0	-1	Beginning of string of 1s in x
1	1	0	Continuation of string of 1s in x

Example

	1	0	0	1	1	1	0	1	1	0	1	0	1	1	1	0	Operand x
(1)	-1	0	1	0	0	-1	1	0	-1	1	-1	1	0	0	-1	0	Recoded version y

a	1 0 1 1 0	
x	1 0 1 0 1	Multiplier
y	$\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$	Booth-recoded
$p^{(0)}$	0 0 0 0 0	
$+y_0a$	0 1 0 1 0	
$2p^{(1)}$	0 0 1 0 1 0	
$p^{(1)}$	0 0 1 0 1 0	
$+y_1a$	1 0 1 1 0	
$2p^{(2)}$	1 1 1 0 1 1 0	
$p^{(2)}$	1 1 1 0 1 1 0	
$+y_2a$	0 1 0 1 0	
$2p^{(3)}$	0 0 0 1 1 1 1 0	
$p^{(3)}$	0 0 0 1 1 1 1 0	
$+y_3a$	1 0 1 1 0	
$2p^{(4)}$	1 1 1 0 0 1 1 1 0	
$p^{(4)}$	1 1 1 0 0 1 1 1 0	
$+y_4a$	0 1 0 1 0	
$2p^{(5)}$	0 0 0 1 1 0 1 1 1 0	
$p^{(5)}$	0 0 0 1 1 0 1 1 1 0	

*from Parhami

Sequential multiplication of 2's-complement numbers with right shifts using Booth's recoding.

Multiplication by Constants

Aspects of multiplication by integer constants:
Produce efficient code using as few registers as possible
Find the best code by a time/space-efficient algorithm

Use binary expansion
Example: multiply R_1 by 113 = $(1110001)_{two}$

$R_2 \leftarrow R_1$ shift-left 1
$R_3 \leftarrow R_2 + R_1$
$R_6 \leftarrow R_3$ shift-left 1
$R_7 \leftarrow R_6 + R_1$
$R_{112} \leftarrow R_7$ shift-left 4
$R_{113} \leftarrow R_{112} + R_1$

Only two registers are required; R_1 and another

Shorter sequence using shift-and-add instructions

$R_3 \leftarrow R_1$ shift-left 1 + R_1
$R_7 \leftarrow R_3$ shift-left 1 + R_1
$R_{113} \leftarrow R_7$ shift-left 4 + R_1

**from Parhami*

Multiplication by Constants

Use of subtraction (Booth's recoding) may help

Example:

multiply R_1 by 113 = $(1110001)_{two}$ = $(100^{-}10001)_{two}$

$$R_8 \leftarrow R_1 \text{ shift-left } 3$$
$$R_7 \leftarrow R_8 - R_1$$
$$R_{112} \leftarrow R_7 \text{ shift-left } 4$$
$$R_{113} \leftarrow R_{112} + R_1$$

Use of factoring may help

Example: multiply R_1 by $119 = 7 \times 17 = (8 - 1) \times (16 + 1)$

$$R_8 \leftarrow R_1 \text{ shift-left } 3$$
$$R_7 \leftarrow R_8 - R_1$$
$$R_{112} \leftarrow R_7 \text{ shift-left } 4$$
$$R_{119} \leftarrow R_{112} + R_7$$

Shorter sequence using shift-and-add/subtract instructions

$$R_7 \leftarrow R_1 \text{ shift-left } 3 - R_1$$
$$R_{119} \leftarrow R_7 \text{ shift-left } 4 + R_7$$

**from Parhami*

Fast Multipliers

Viewing multiplication as a multioperand addition problem, there are but two ways to speed it up

a. Reducing the number of operands to be added: handling more than one multiplier bit at a time (high-radix multipliers, Chapter 10)

b. Adding the operands faster: parallel/pipelined multioperand addition (tree and array multipliers, Chapter 11)

from Parhami

Using Higher Radix Multiplier

10.1 Radix-4 Multiplication

Radix-r versions of multiplication recurrences

Multiplication with right shifts: top-to-bottom accumulation

$$p^{(j+1)} = (p^{(j)} + x_j\, a\, r^k)\, r^{-1} \qquad \text{with} \quad p^{(0)} = 0 \quad \text{and}$$

|——add——|

|—shift right—|

$$p^{(k)} = p = ax + p^{(0)} r^{-k}$$

Multiplication with left shifts: bottom-to-top accumulation

$$p^{(j+1)} = r\, p^{(j)} + x_{k-j-1}\, a \qquad \text{with} \quad p^{(0)} = 0 \quad \text{and}$$

|shift|

|——add——|

$$p^{(k)} = p = ax + p^{(0)} r^{k}$$

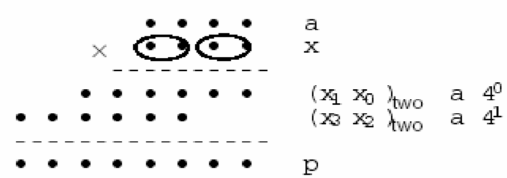


Fig. 10.1 Radix-4, or two-bit-at-a-time, multiplication in dot notation.

**from Parhami*

Using Higher Radix Multiplier

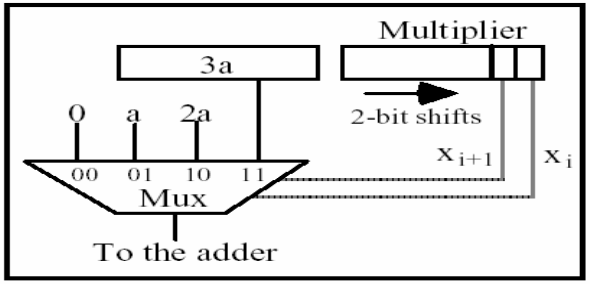


Fig. 10.2 The multiple generation part of a radix-4 multiplier with precomputation of 3*a*.

a			0	1	1	0				
$3a$	0	1	0	0	1	0				
x			1	1	1	0				
$p^{(0)}$			0	0	0	0				
$+(x_1x_0)_{two}a$	0	0	1	1	0	0				
$4p^{(1)}$	0	0	1	1	0	0				
$p^{(1)}$			0	0	1	1	0	0		
$+(x_3x_2)_{two}a$	0	1	0	0	1	0				
$4p^{(2)}$	0	1	0	1	0	1	0	0		
$p^{(2)}$			0	1	0	1	0	1	0	0

Fig. 10.3 Example of radix-4 multiplication using the 3*a* multiple.

**from Parhami*

Higher Radix Multiplier

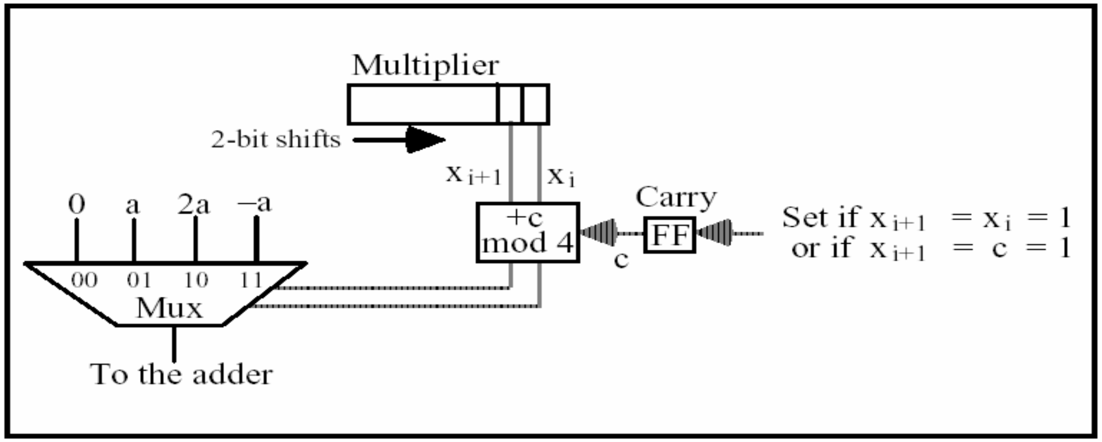


Fig. 10.4 **The multiple generation part of a radix-4 multiplier based on replacing 3*a* with 4*a* (carry into next higher radix-4 multiplier digit) and –*a*.**

x_{i+1}	x_i	c	Mux control		Set carry
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	0	1

**from Parhami*

10.2 Modified Booth's Recoding

Table 10.1 Radix-4 Booth's recoding yielding $(z_{k/2} \cdots z_1 z_0)_{four}$

x_{i+1}	x_i	x_{i-1}	y_{i+1}	y_i	$z_{i/2}$	Explanation
0	0	0	0	0	0	No string of 1s in sight
0	0	1	0	1	1	End of string of 1s
0	1	0	0	1	1	Isolated 1
0	1	1	1	0	2	End of string of 1s
1	0	0	$^{-}1$	0	$^{-}2$	Beginning of string of 1s
1	0	1	$^{-}1$	1	$^{-}1$	End a string, begin new one
1	1	0	0	$^{-}1$	$^{-}1$	Beginning of string of 1s
1	1	1	0	0	0	Continuation of string of 1s

Example: $(21\ 31\ 22\ 32)_{four}$

	1 0	0 1	1 1	0 1	1 0	1 0	1 1	1 0	Operand x
(1)	$^{-}2$	2	$^{-}1$	2	$^{-}1$	$^{-}1$	0	$^{-}2$	Recoded version z

*from Parhami

Booth's Recoding

a			0	1	1	0					
x			1	0	1	0					
z				$^{-}1$		$^{-}2$					Recoded version of x
$p^{(0)}$	0	0	0	0	0	0					
$+z_0a$	1	1	0	1	0	0					
$4p^{(1)}$	1	1	0	1	0	0					
$p^{(1)}$	1	1	1	1	0	1	0	0			
$+z_1a$	1	1	1	0	1	0					
$4p^{(2)}$	1	1	0	1	1	1	0	0			
$p^{(2)}$			1	1	0	1	1	1	0	0	

Fig. 10.5 Example radix-4 multiplication with modified Booth's recoding of the 2's-complement multiplier.

**from Parhami*

Booth's Recoding

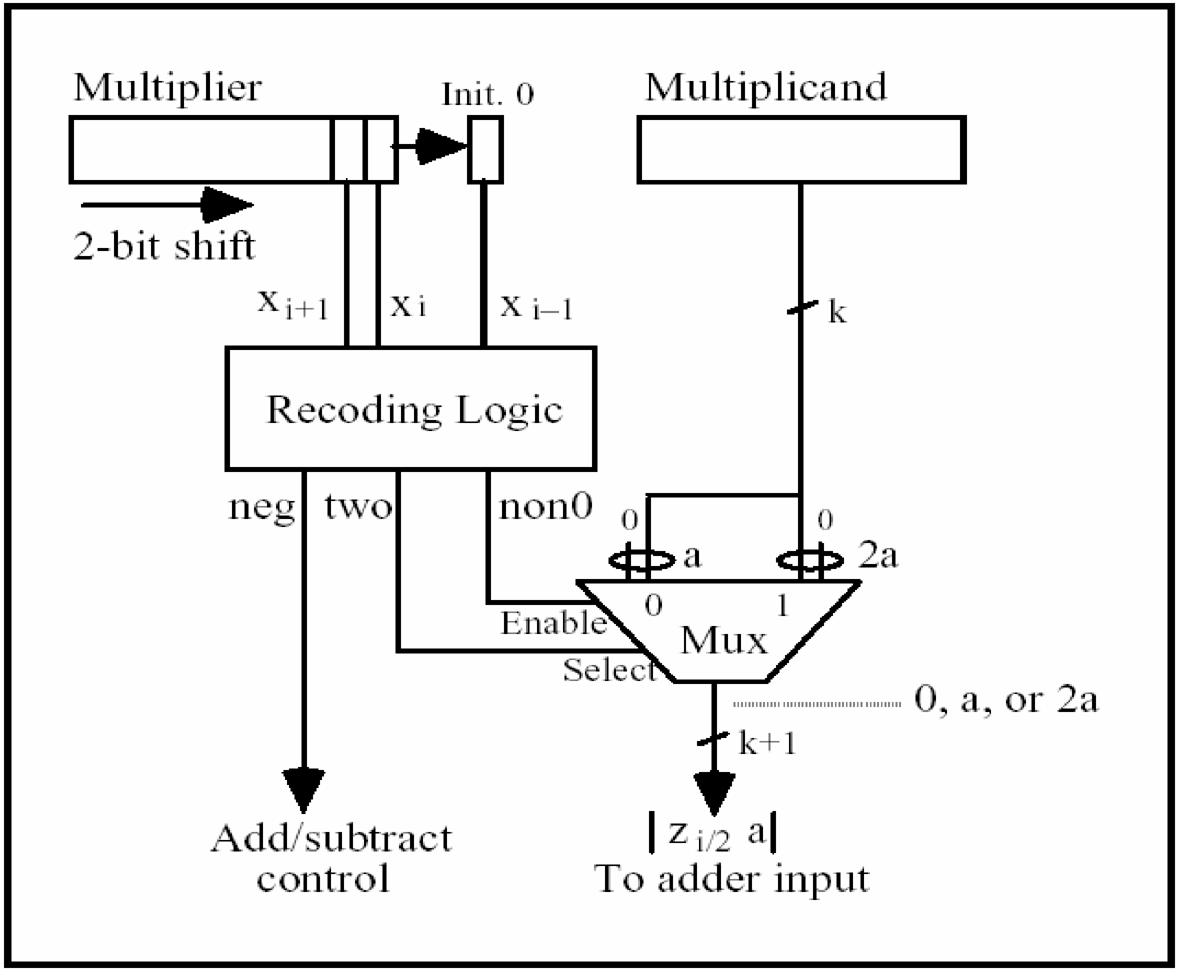
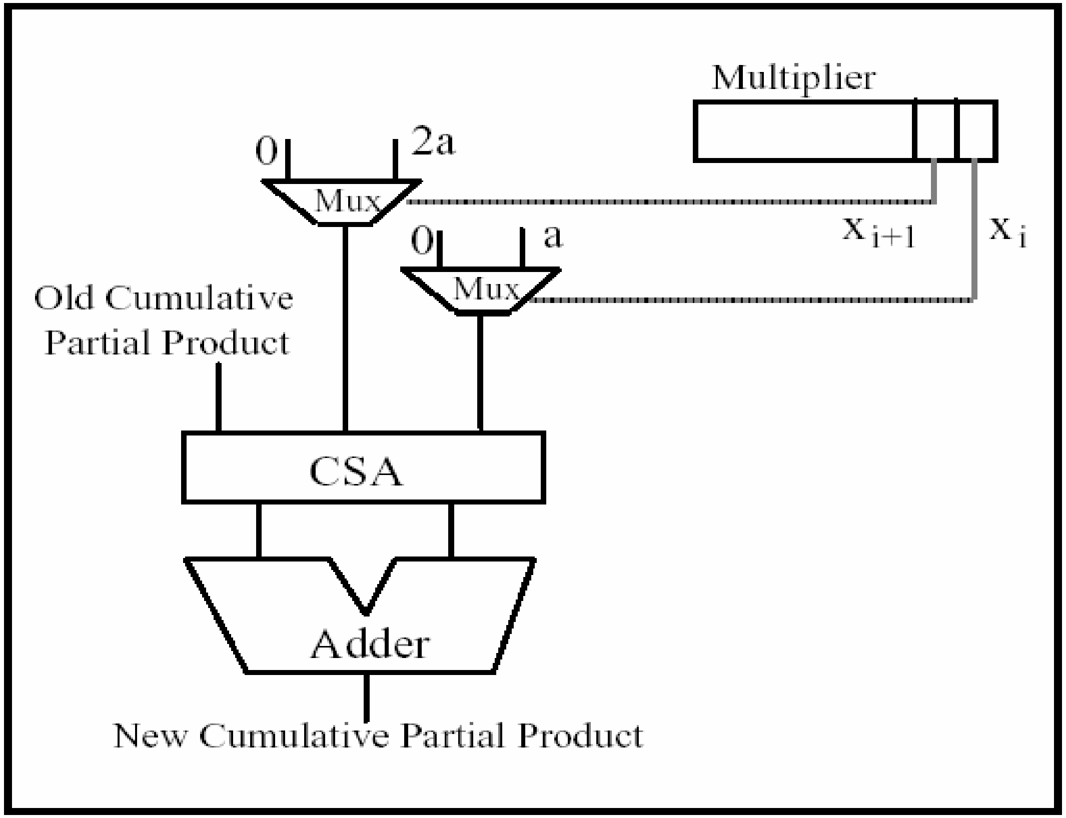


Fig. 10.6 The multiple generation part of a radix-4 multiplier based on Booth's recoding.

**from Parhami*

Booth's Recoding

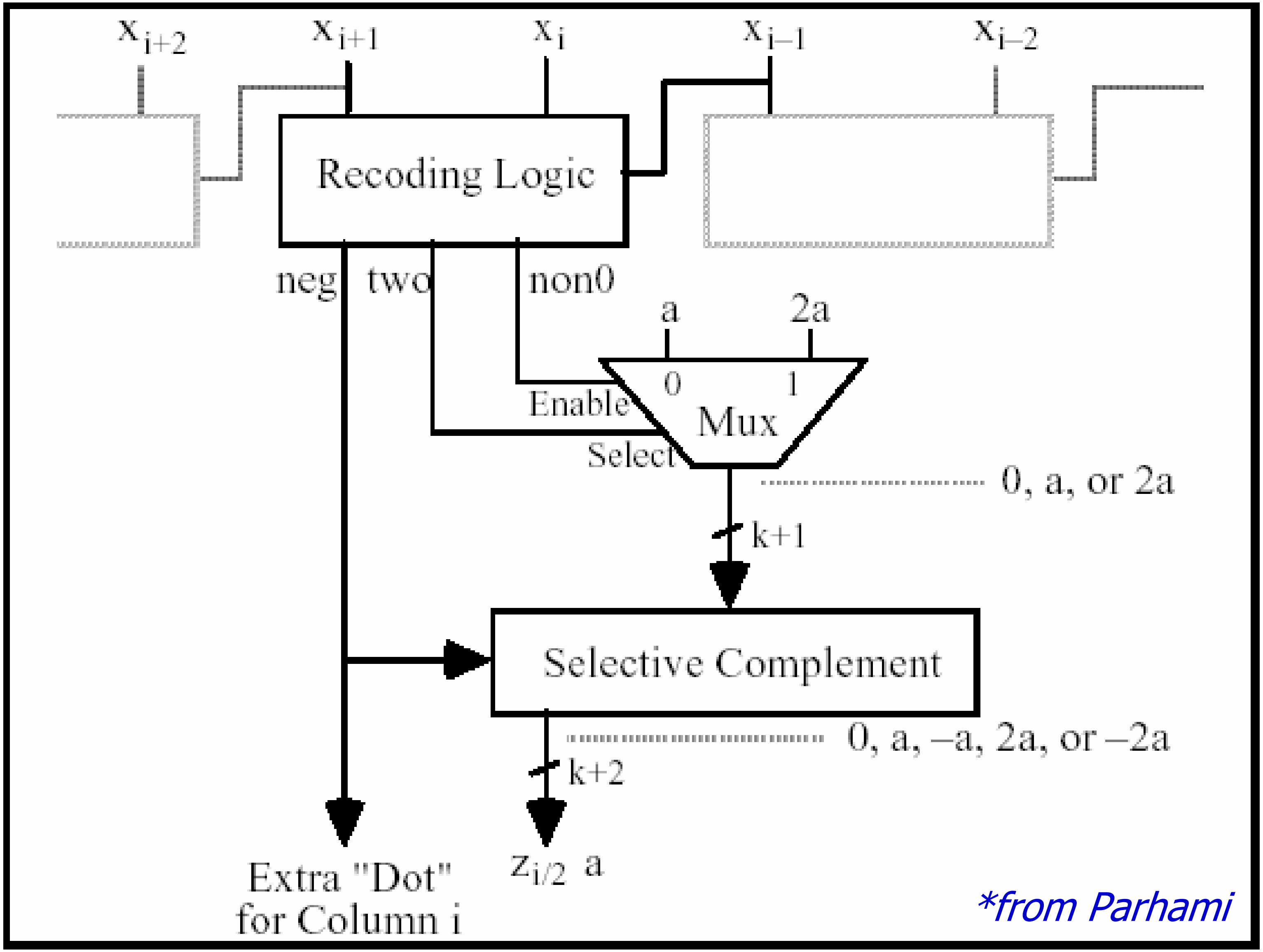
Using Carry-Save Adders



Radix-4 multiplication with a carry-save adder used to combine the cumulative partial product, $x_i a$, and $2x_{i+1}a$ into two numbers.

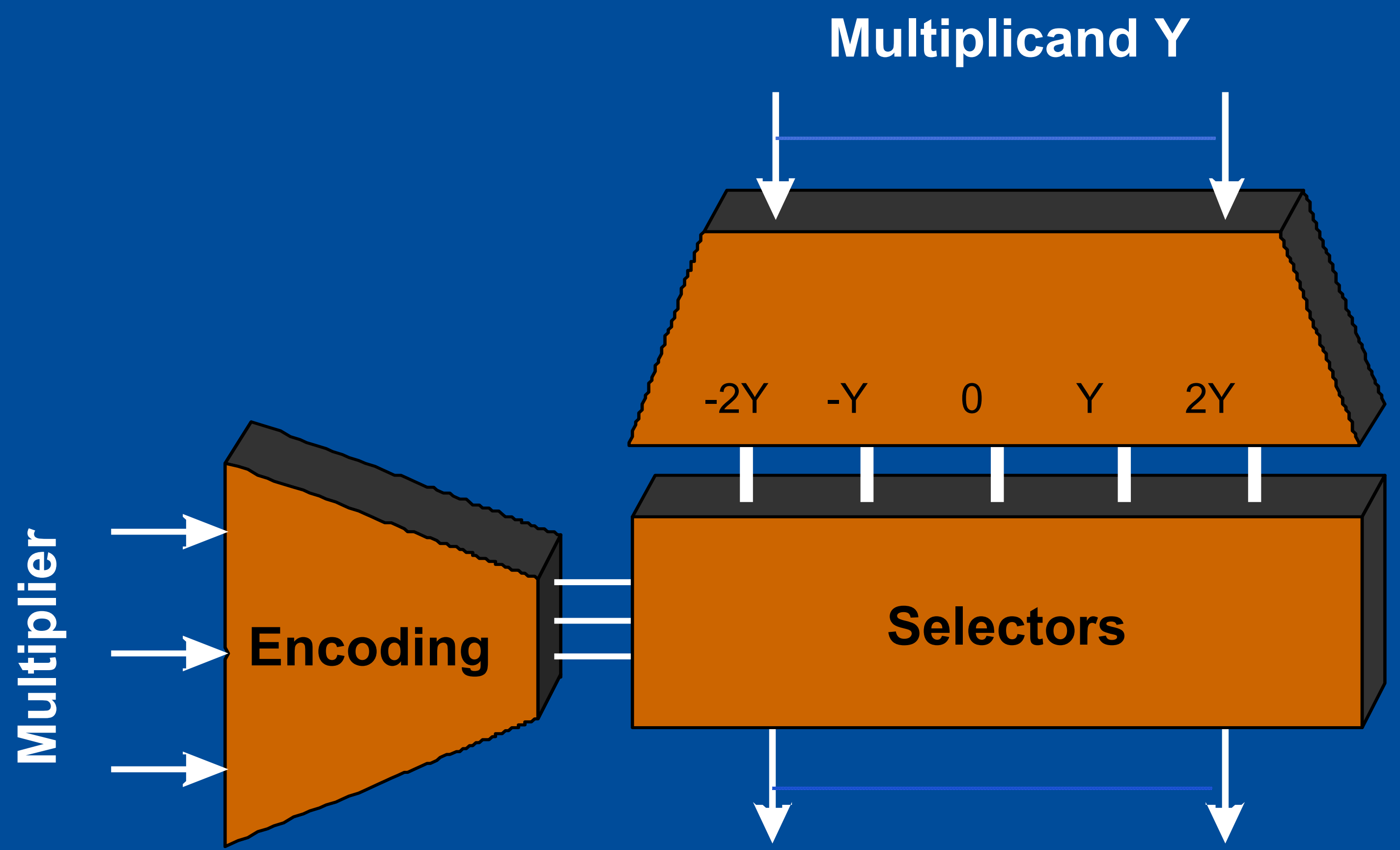
**from Parhami*

Booth recoding and multiple selection logic for high-radix or parallel multiplication.

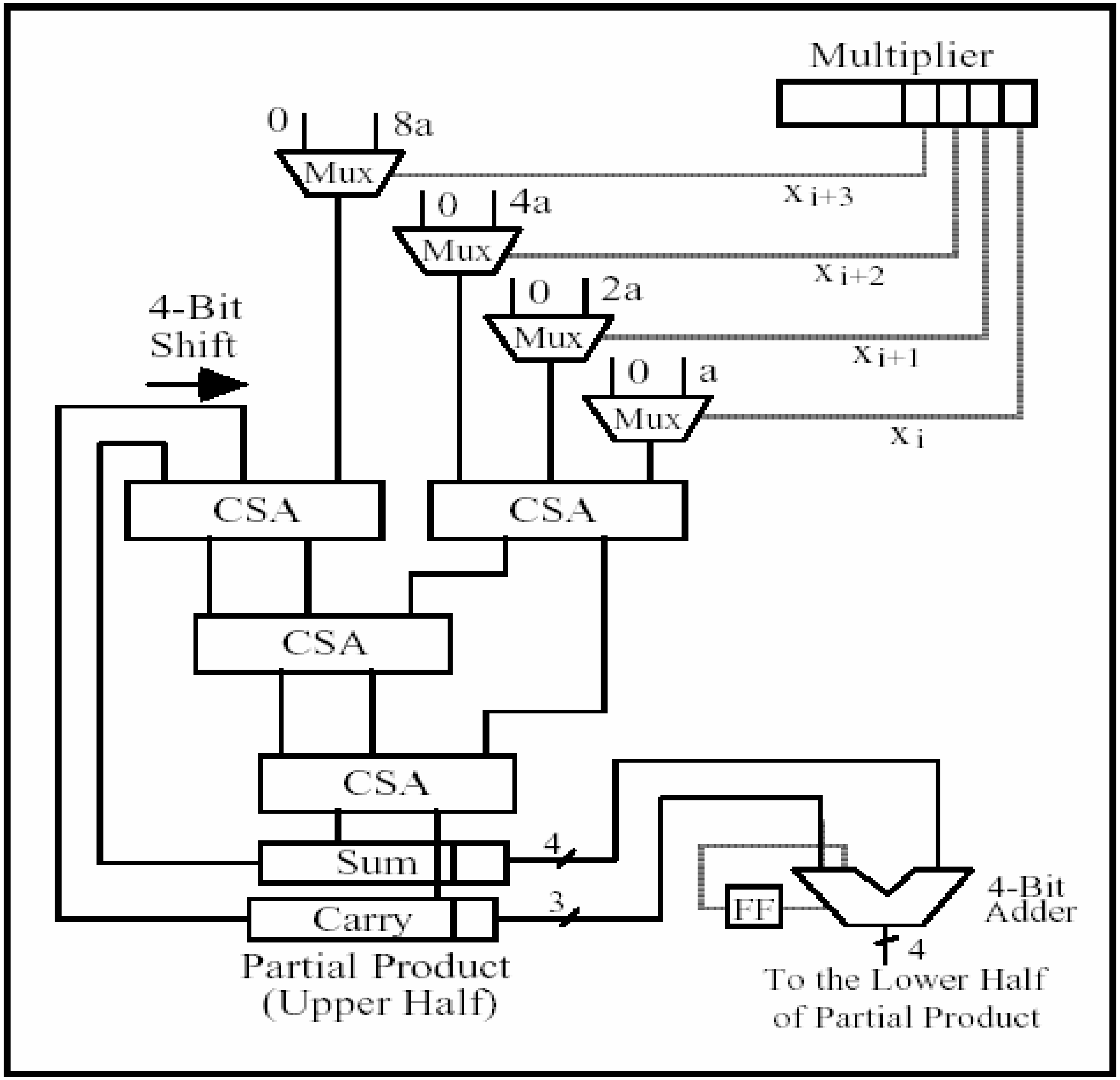


*from Parhami

Modified Booth Recording Implementation



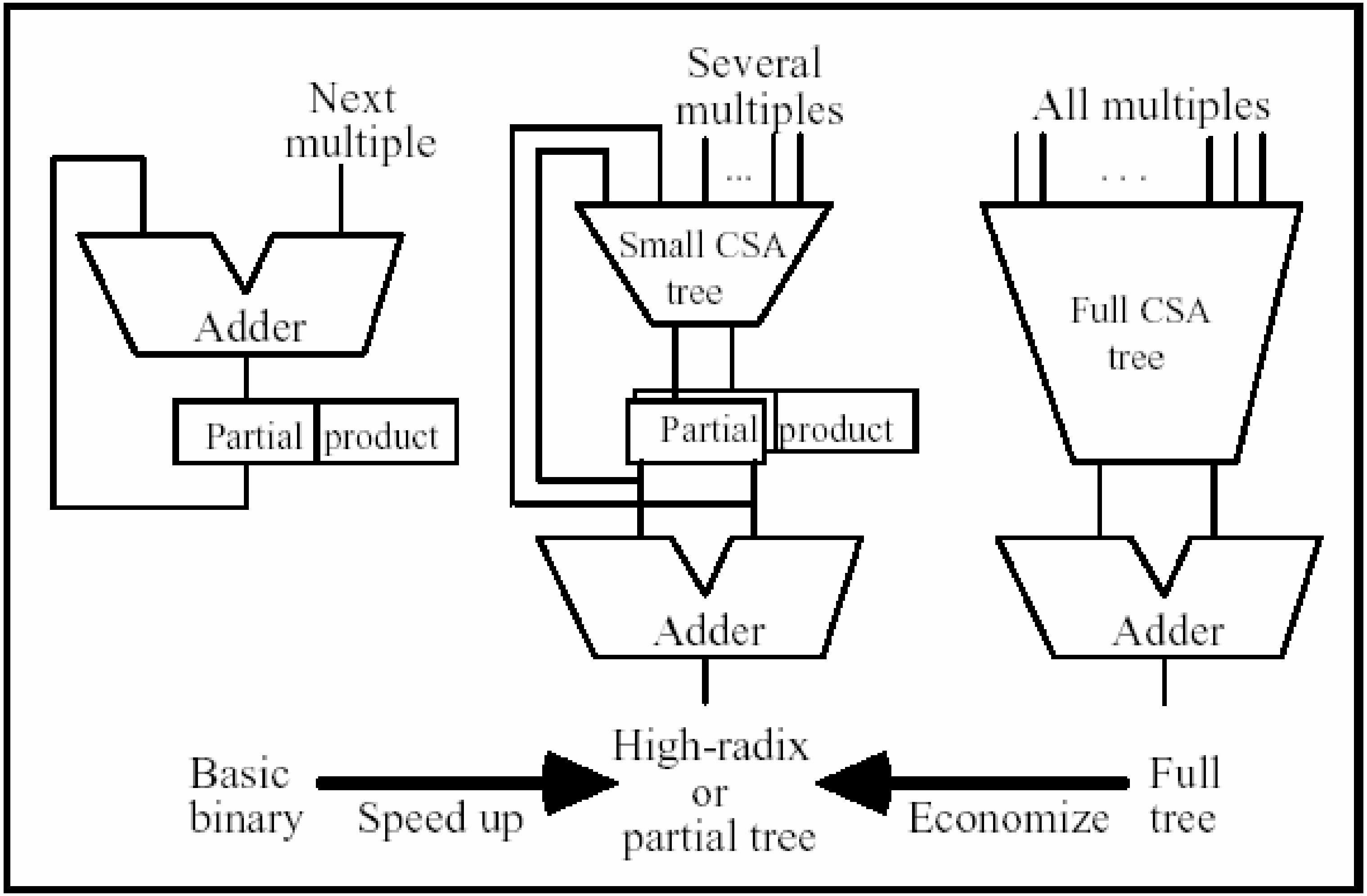
Higher Radix Multipliers



*from Parhami

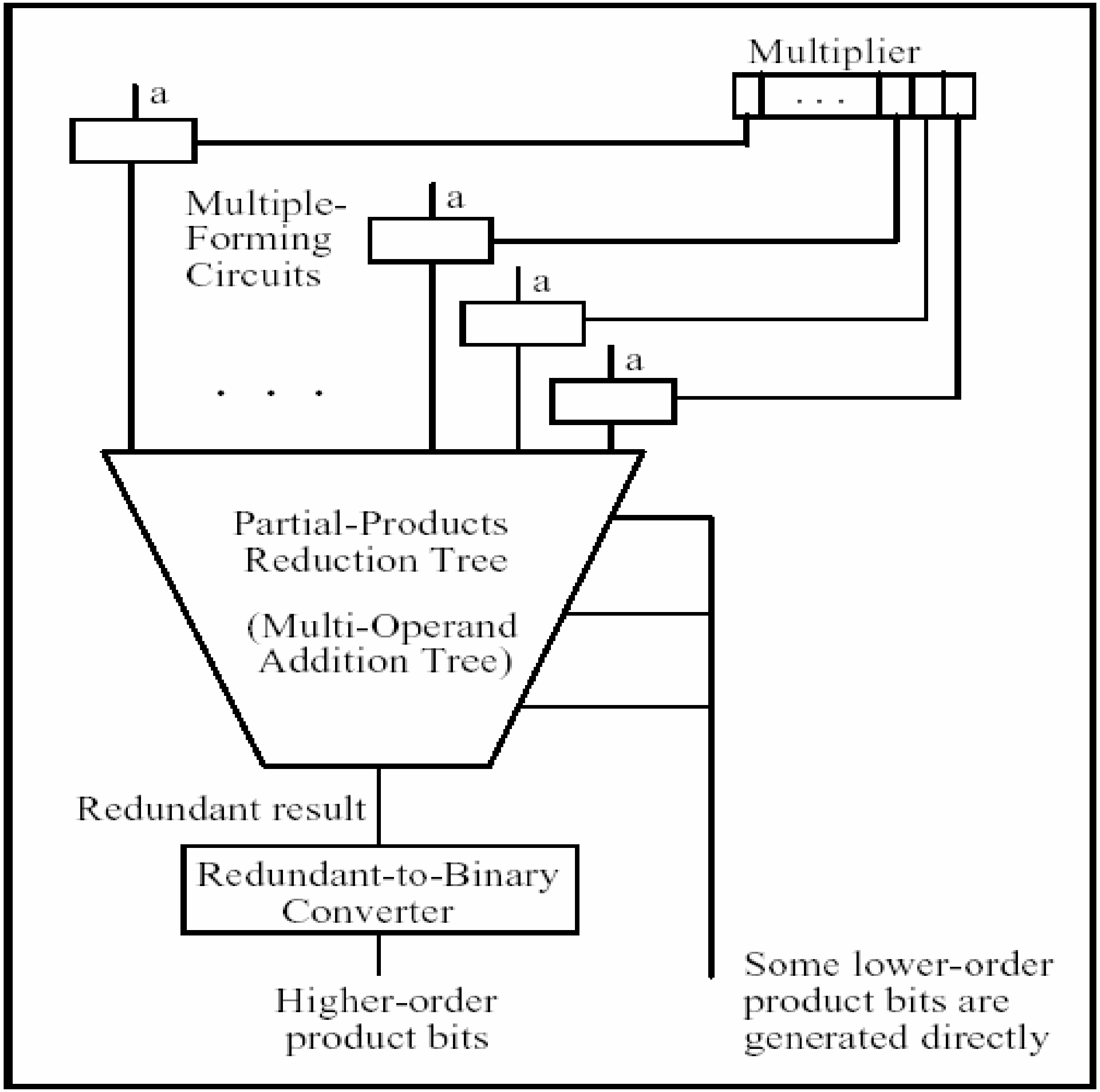
Radix-16 multiplication with the upper half of the cumulative partial product in carry-save form.

Tree and Array Multipliers



**from Parhami*

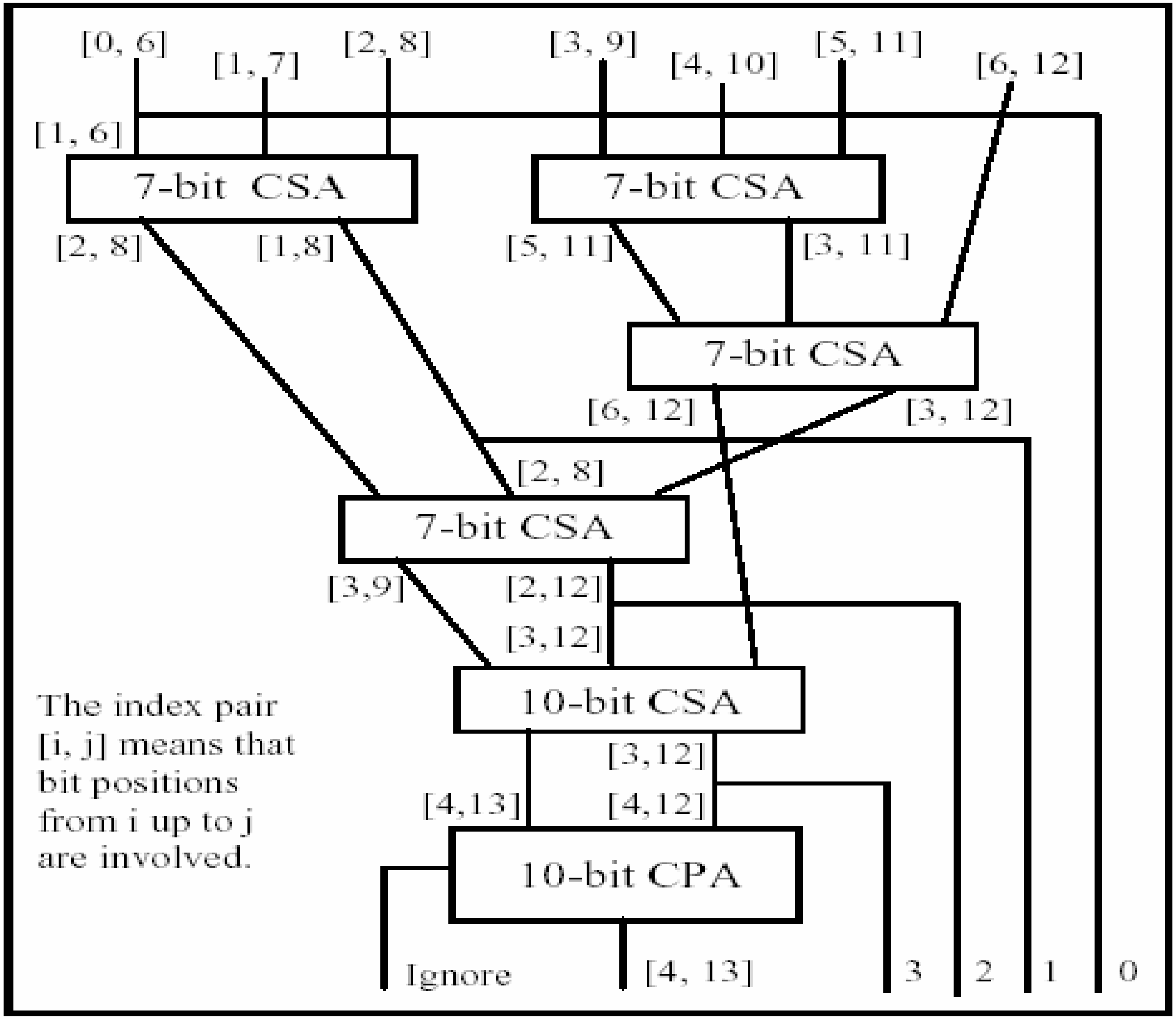
Tree and Array Multipliers



General structure of a full-tree multiplier.

**from Parhami*

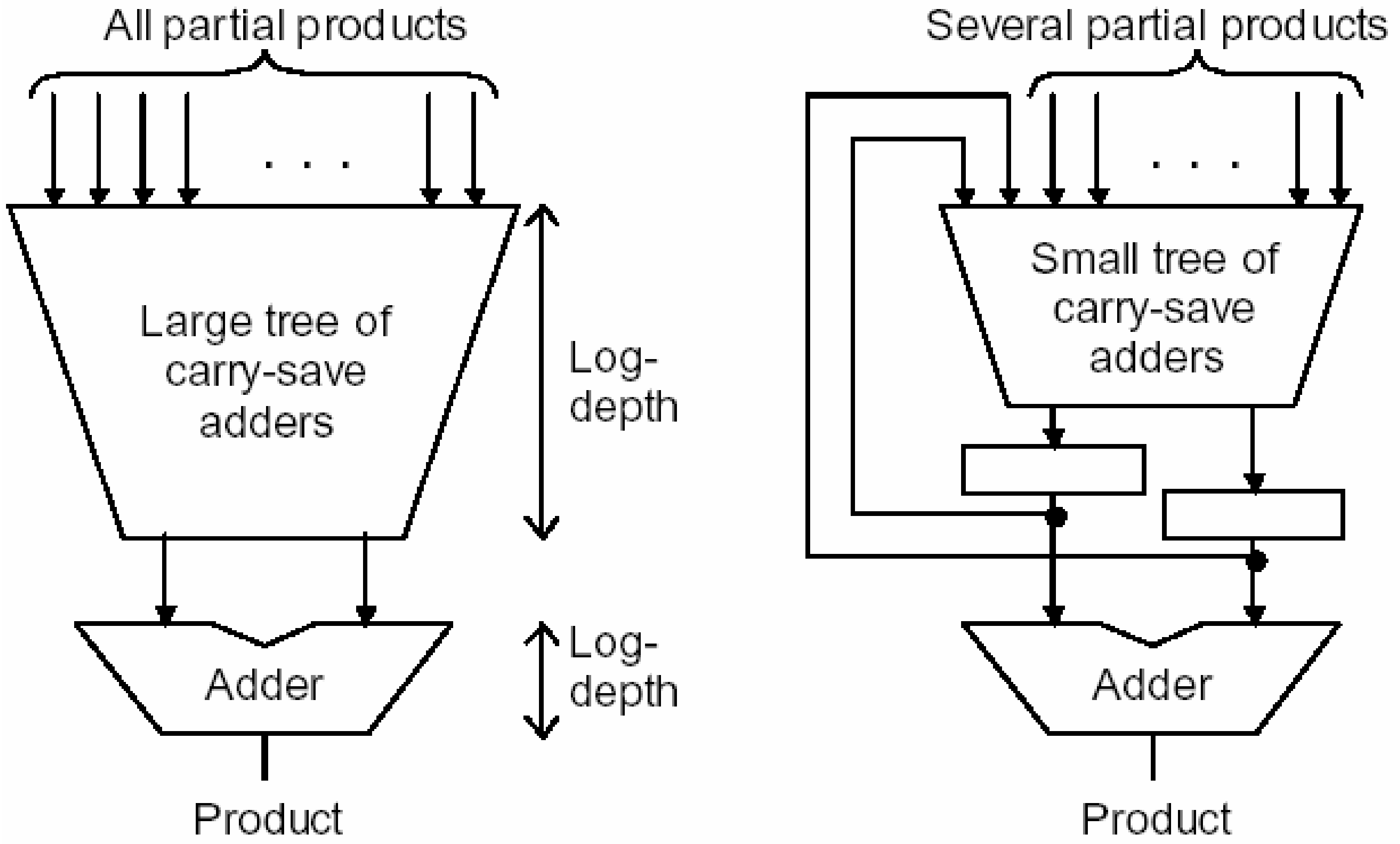
Tree Multipliers



*from Parhami

Possible CSA tree for a 7 × 7 tree multiplier.

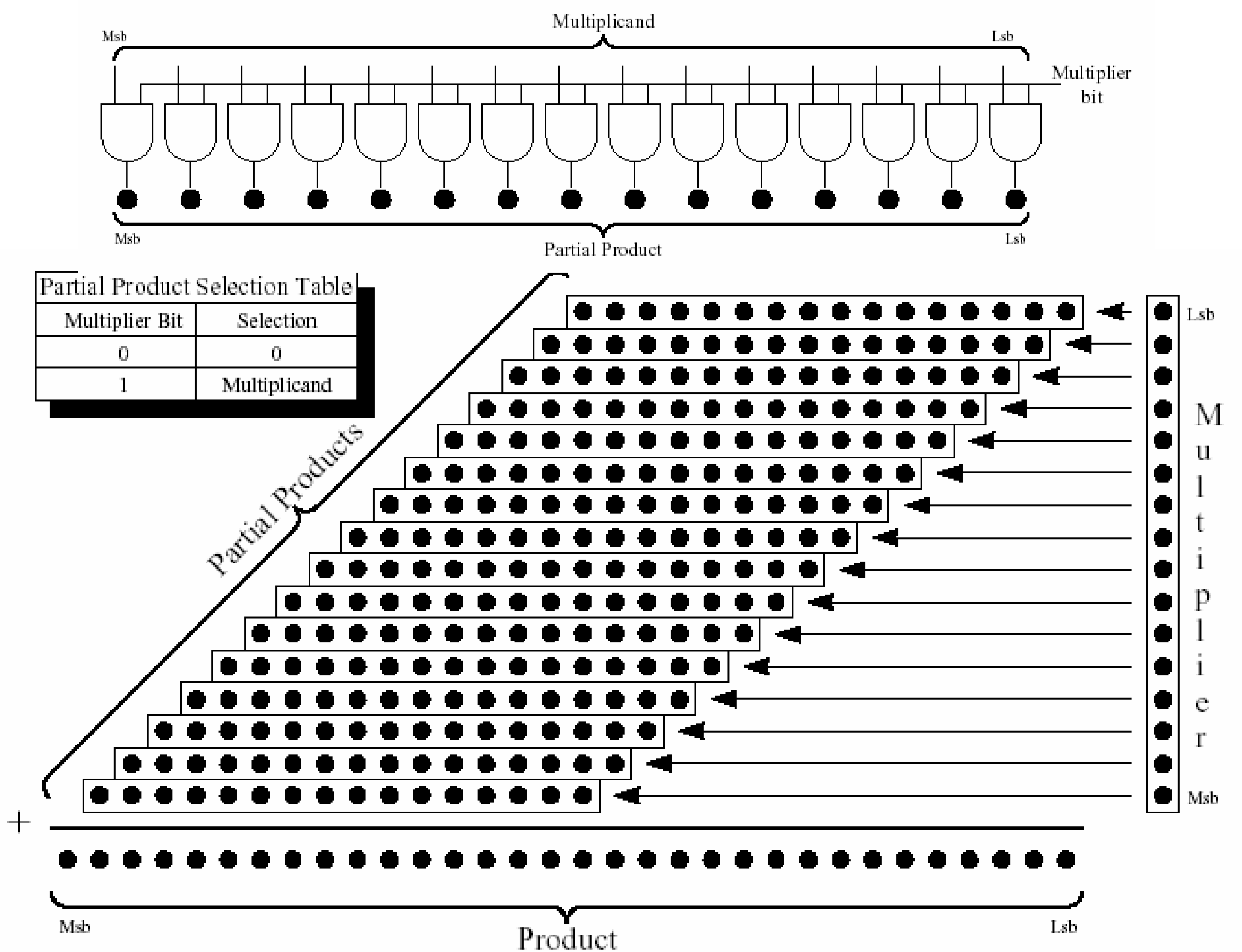
Tree Multipliers



Schematic diagrams for full-tree and partial-tree multipliers.

**from Parhami*

Generating Partial Products



Partial Product Selection Table

Multiplier Bit	Selection
0	0
1	Multiplicand

*from G. Bewick

Generating Partial Products

**from G. Bewick*

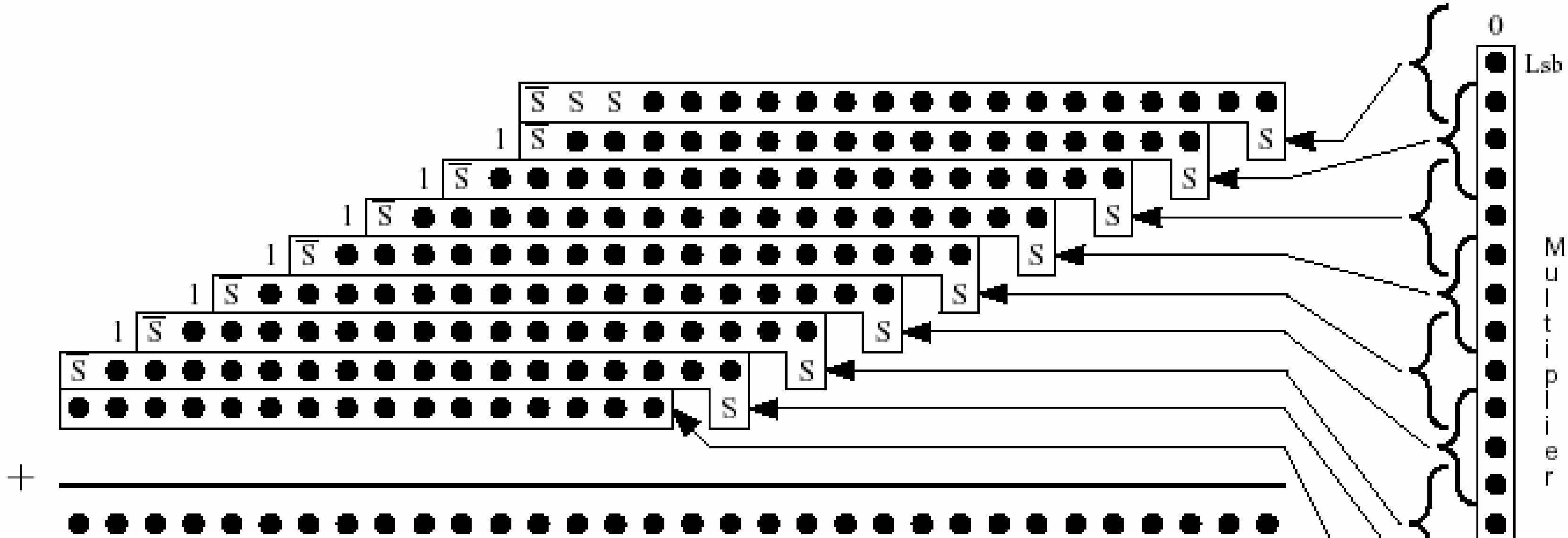
Multiplier = 63669_{10} = 1111100010110101

Multiplicand (M) = 40119_{10} = 1001110010110111

Partial product		Multiplier bit	
1001110010110111	M	1	Lsb
0000000000000000	0	0	
1001110010110111	M	1	M
0000000000000000	0	0	u
1001110010110111	M	1	l
1001110010110111	M	1	t
0000000000000000	0	0	i
1001110010110111	M	1	p
0000000000000000	0	0	l
0000000000000000	0	0	i
0000000000000000	0	0	e
1001110010110111	M	1	r
1001110010110111	M	1	
1001110010110111	M	1	
1001110010110111	M	1	
+ 1001110010110111	M	1	Msb

10011000010000000001010101100011 = 2554336611_{10} = Product

Generating Partial Products using Booth's Recoding



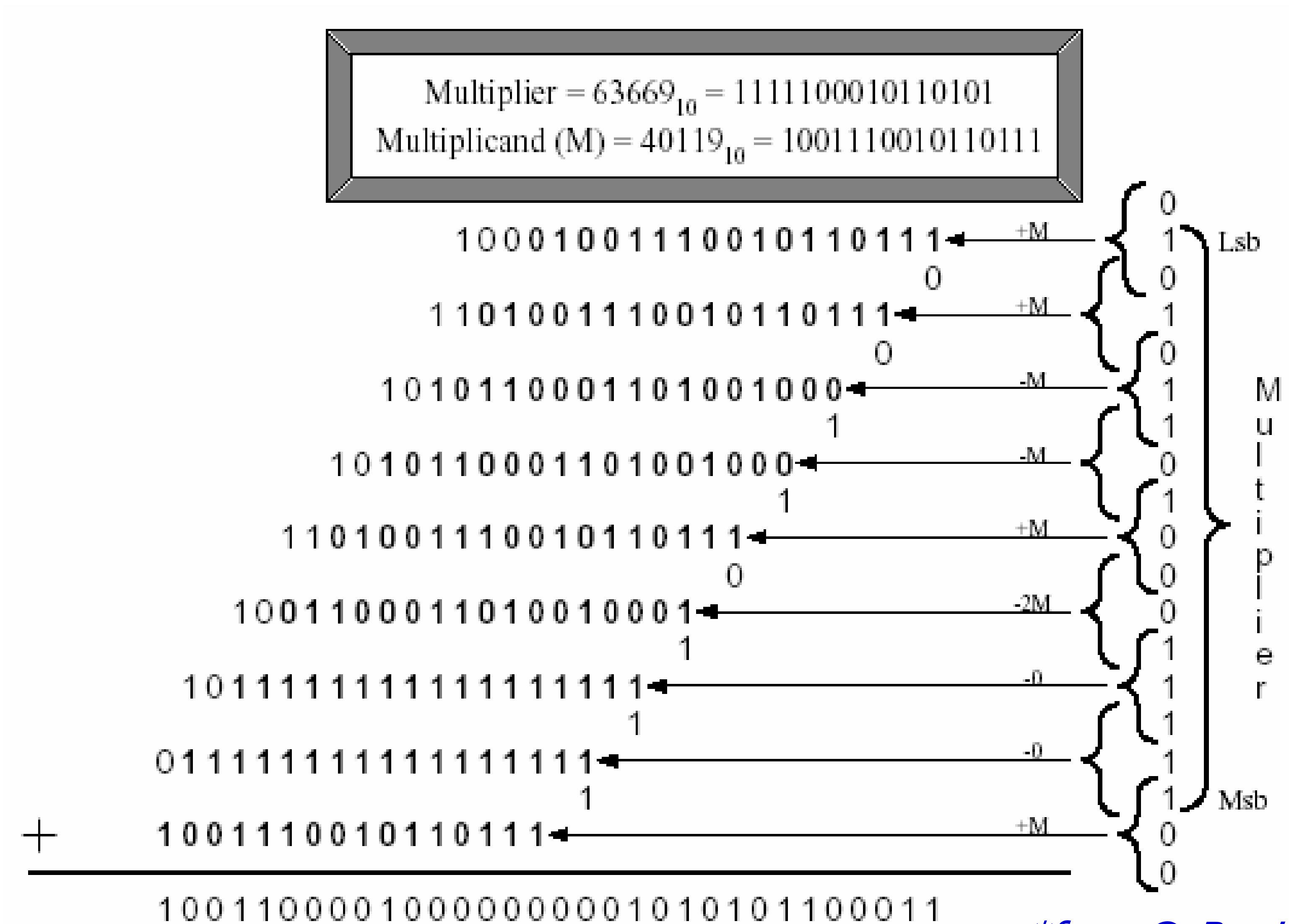
Partial Product Selection Table	
Multiplier Bits	Selection
000	+ 0
001	+ Multiplicand
010	+ Multiplicand
011	+ 2 x Multiplicand
100	-2 x Multiplicand
101	- Multiplicand
110	- Multiplicand
111	- 0

S = 0 if partial product is positive (top 4 entries from table)

S = 1 if partial product is negative (bottom 4 entries from table)

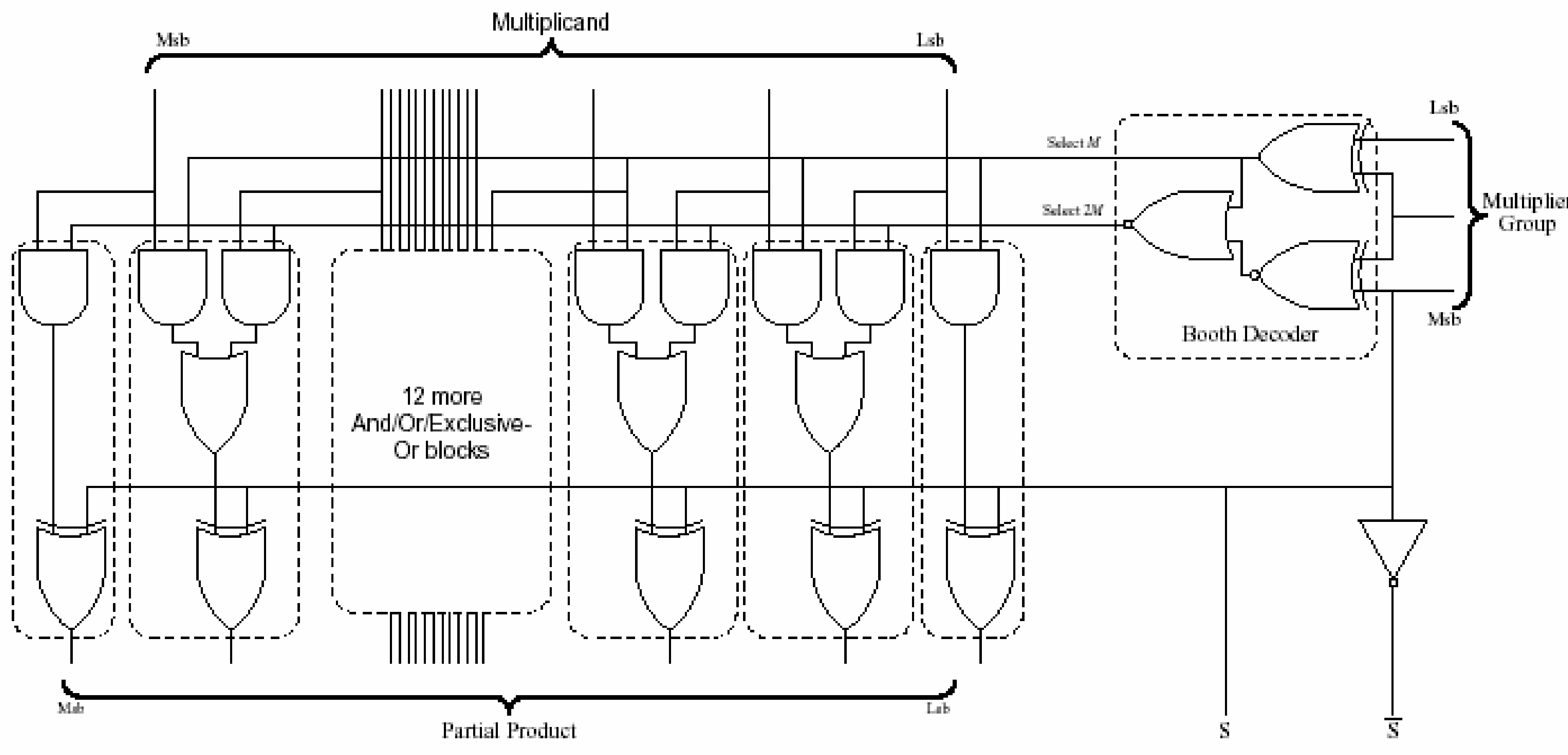
**from G. Bewick*

Generating Partial Products using Booth's Recoding



**from G. Bewick*

Booth Partial Product Selector Logic



**from G. Bewick*

Radix-2 Booth Recoded Multiplier with Negative Partial Products

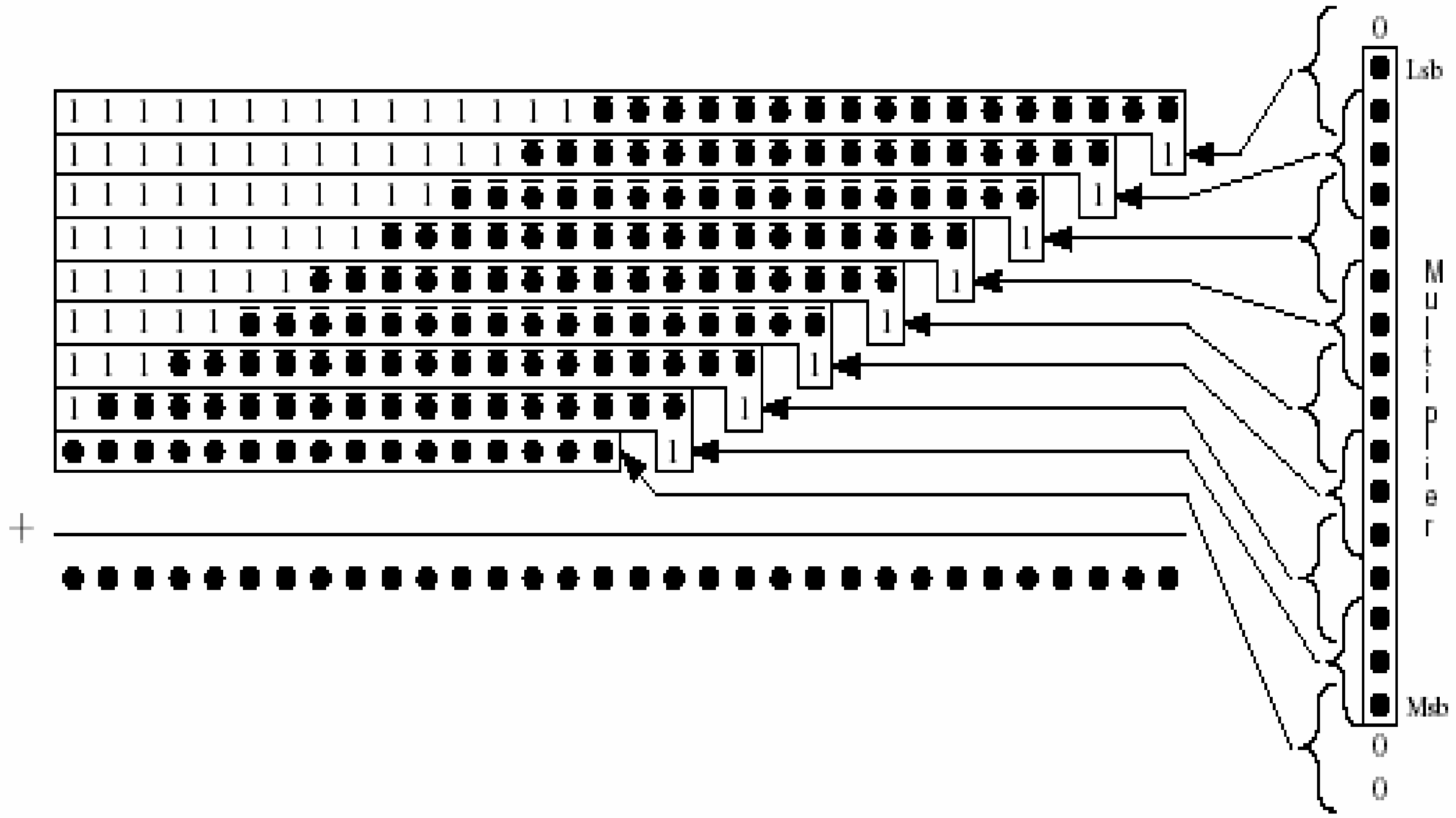


Figure A.2: 16 bit Booth 2 multiplication with negative partial products.

*from G. Bewick

Radix-2 Booth Recoded Multiplier with Summed Sign Extension

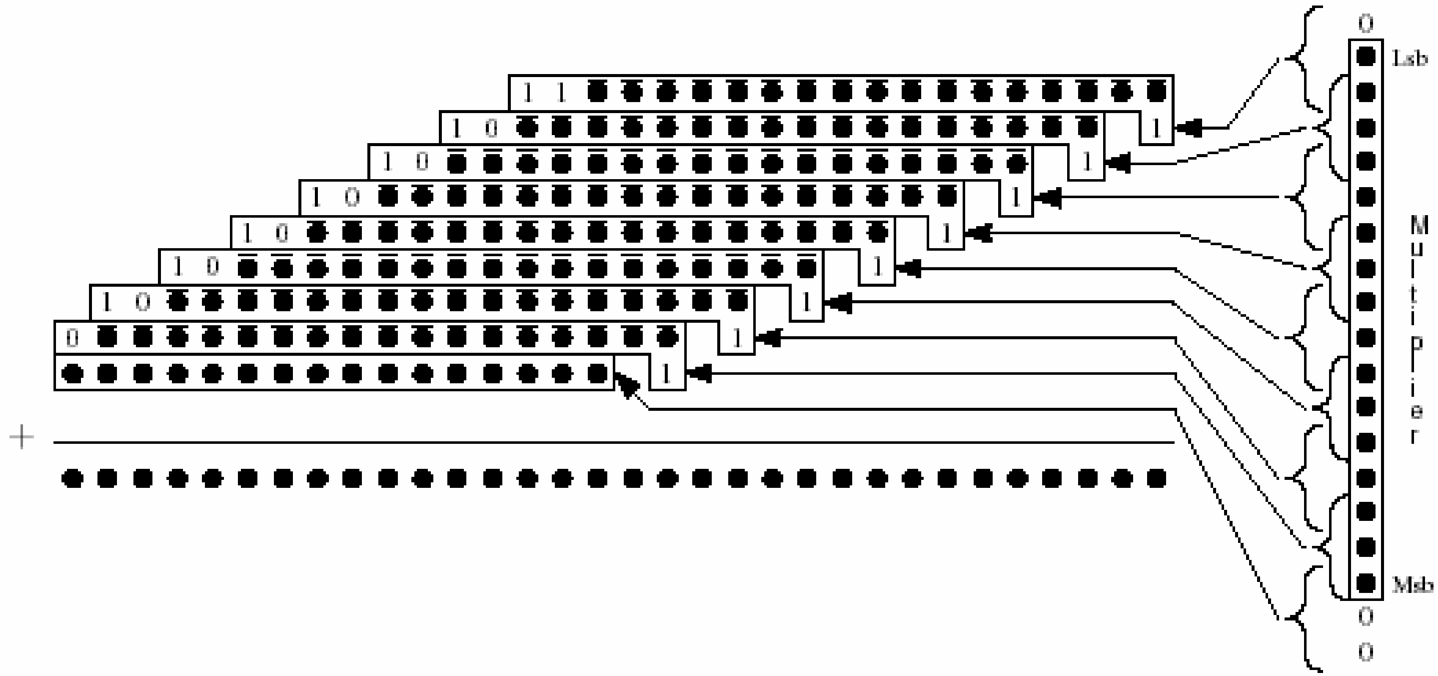
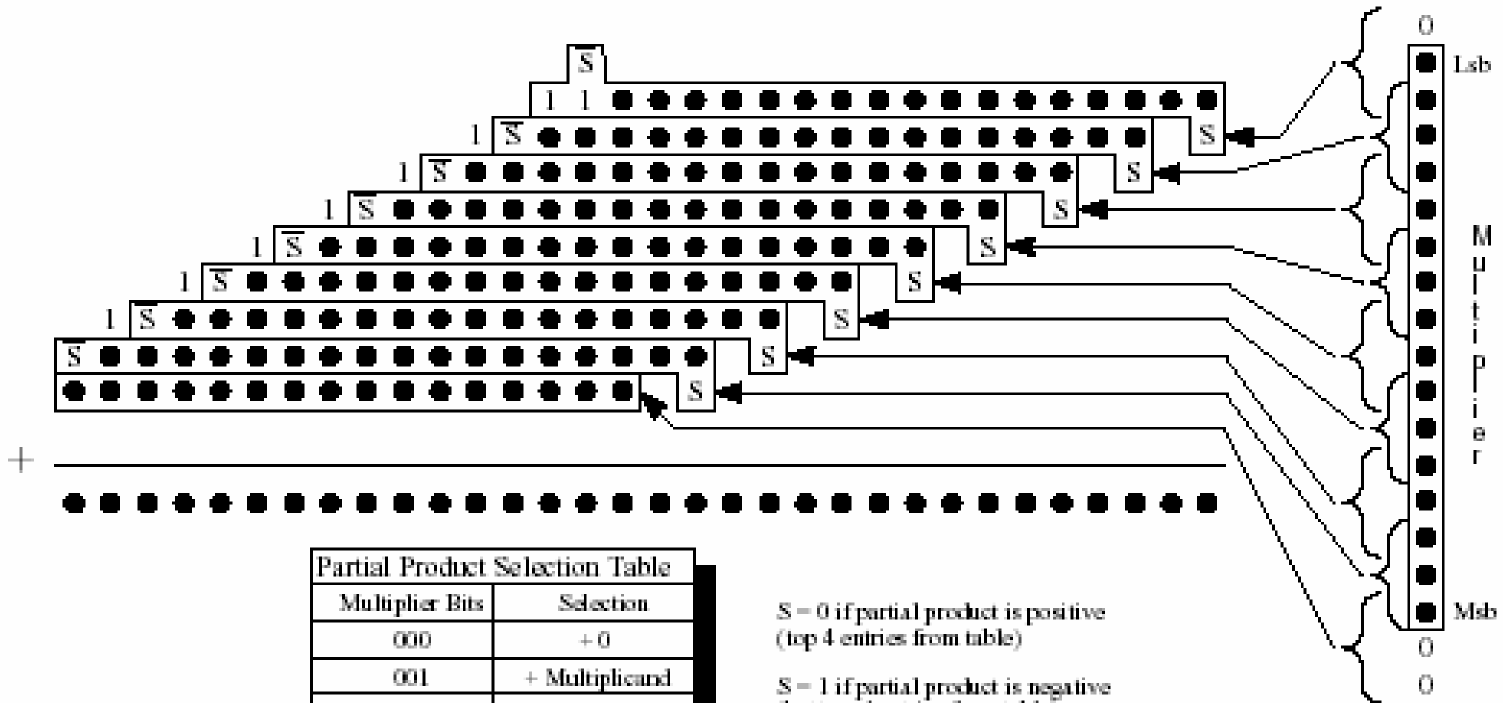


Figure A.3: Negative partial products with summed sign extension.

*from G. Bewick

Radix-2 Booth Recoded Multiplier with Summed Sign Extension



Partial Product Selection Table	
Multiplier Bits	Selection
000	+0
001	+ Multiplicand
010	+ Multiplicand
011	+ 2 x Multiplicand
100	-2 x Multiplicand
101	- Multiplicand
110	- Multiplicand
111	- 0

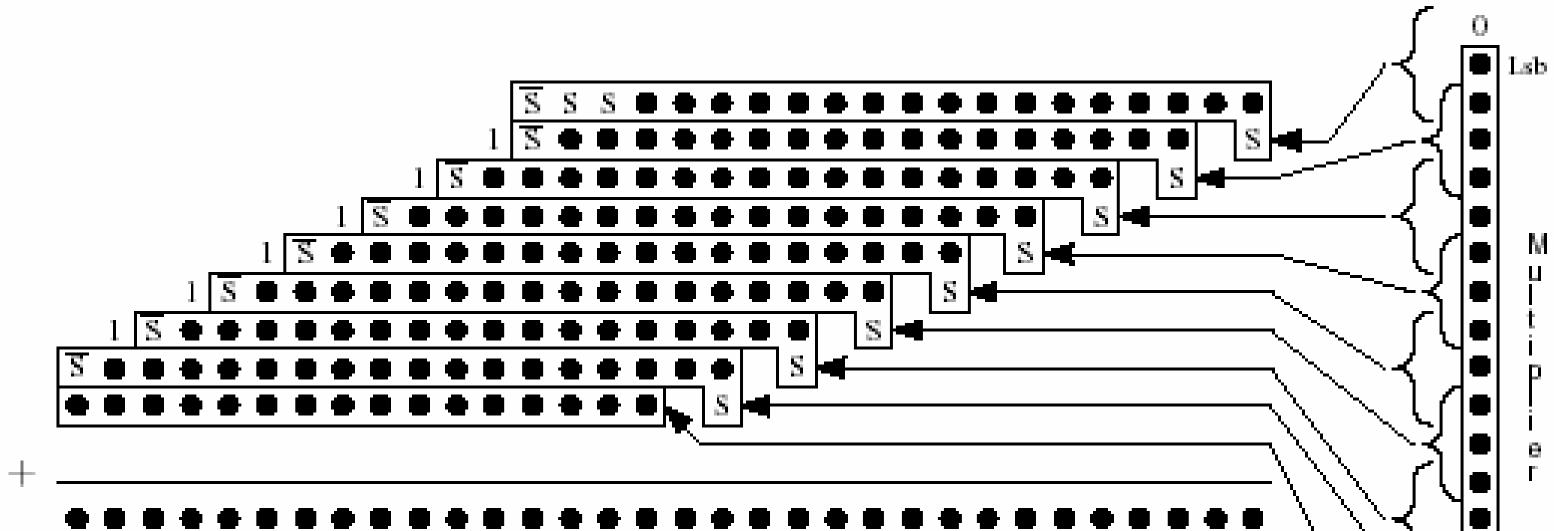
S = 0 if partial product is positive (top 4 entries from table)

S = 1 if partial product is negative (bottom 4 entries from table)

Figure A.4: Complete 16 bit Booth 2 multiplication.

**from G. Bewick*

Radix-2 Booth Recoded Multiplier with Summed Sign Extension and Reduced Logic Depth



Partial Product Selection Table	
Multiplier Bits	Selection
000	+0
001	+ Multiplicand
010	+ Multiplicand
011	+2 x Multiplicand
100	-2 x Multiplicand
101	- Multiplicand
110	- Multiplicand
111	-0

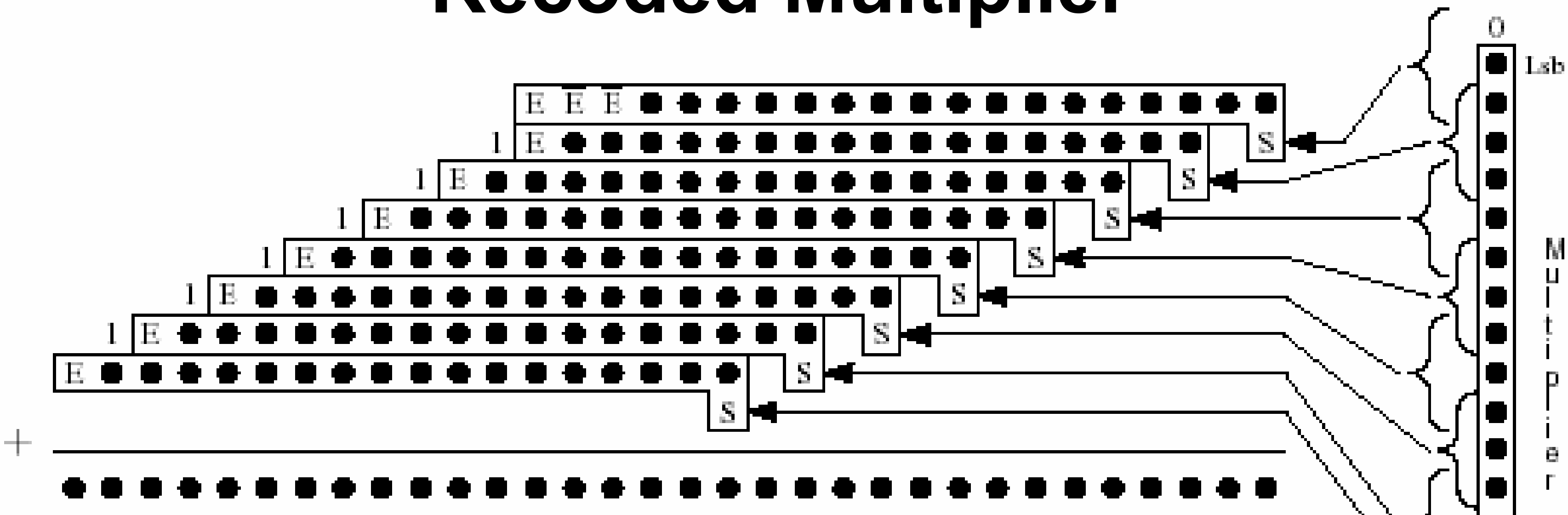
S = 0 if partial product is positive (top 4 entries from table)

S = 1 if partial product is negative (bottom 4 entries from table)

from G. Bewick

Figure A.5: Complete 16 bit Booth 2 multiplication with height reduction.

Complete Signed Radix-2 Booth Recoded Multiplier



Partial Product Selection Table	
Multiplier Bits	Selection
000	+ 0
001	+ Multiplicand
010	+ Multiplicand
011	+ 2 x Multiplicand
100	-2 x Multiplicand
101	- Multiplicand
110	- Multiplicand
111	- 0

S = 0 if partial product is positive
(top 4 entries from table)

S = 1 if partial product is negative
(bottom 4 entries from table)

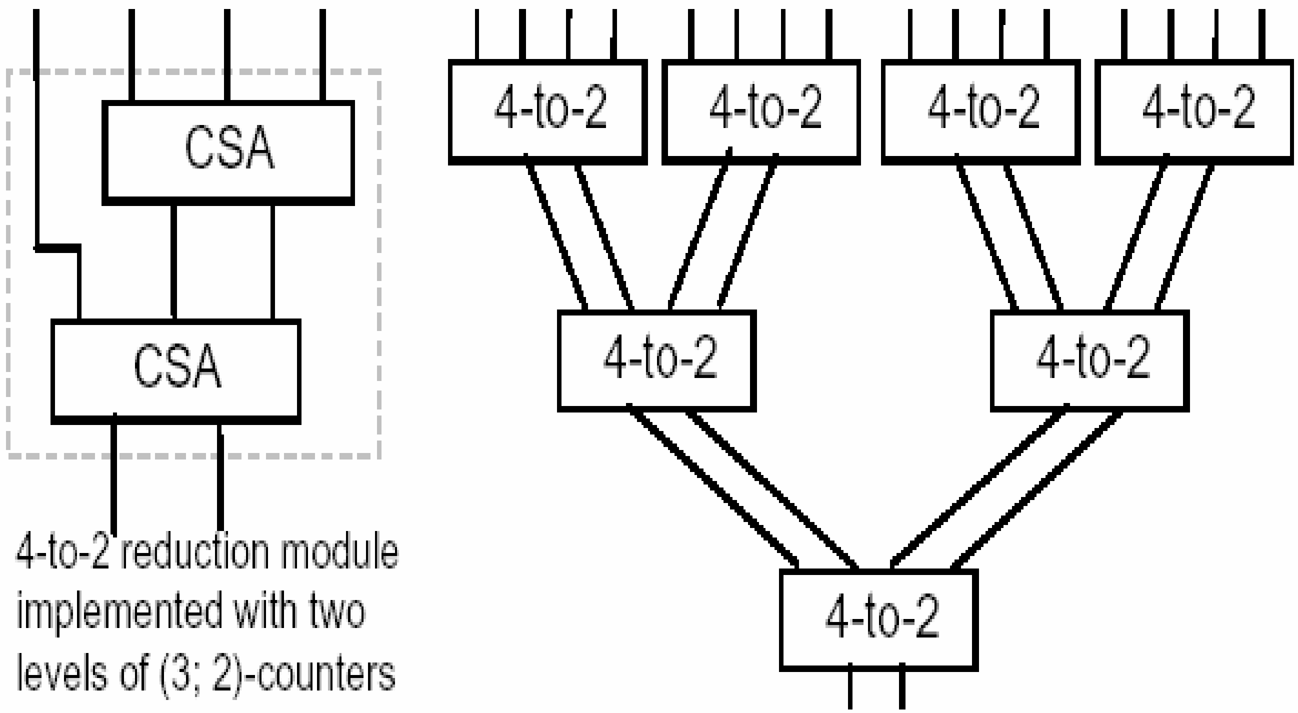
E = 1 if multiplicand is positive and partial product is positive, or if multiplicand is negative and partial product is negative or if partial product is +0.

E = 0 if multiplicand is positive and partial product is negative, or if multiplicand is negative and partial product is positive or if partial product is -0.

Figure A.6: Complete signed 16 bit Booth 2 multiplication.

**from G. Bewick*

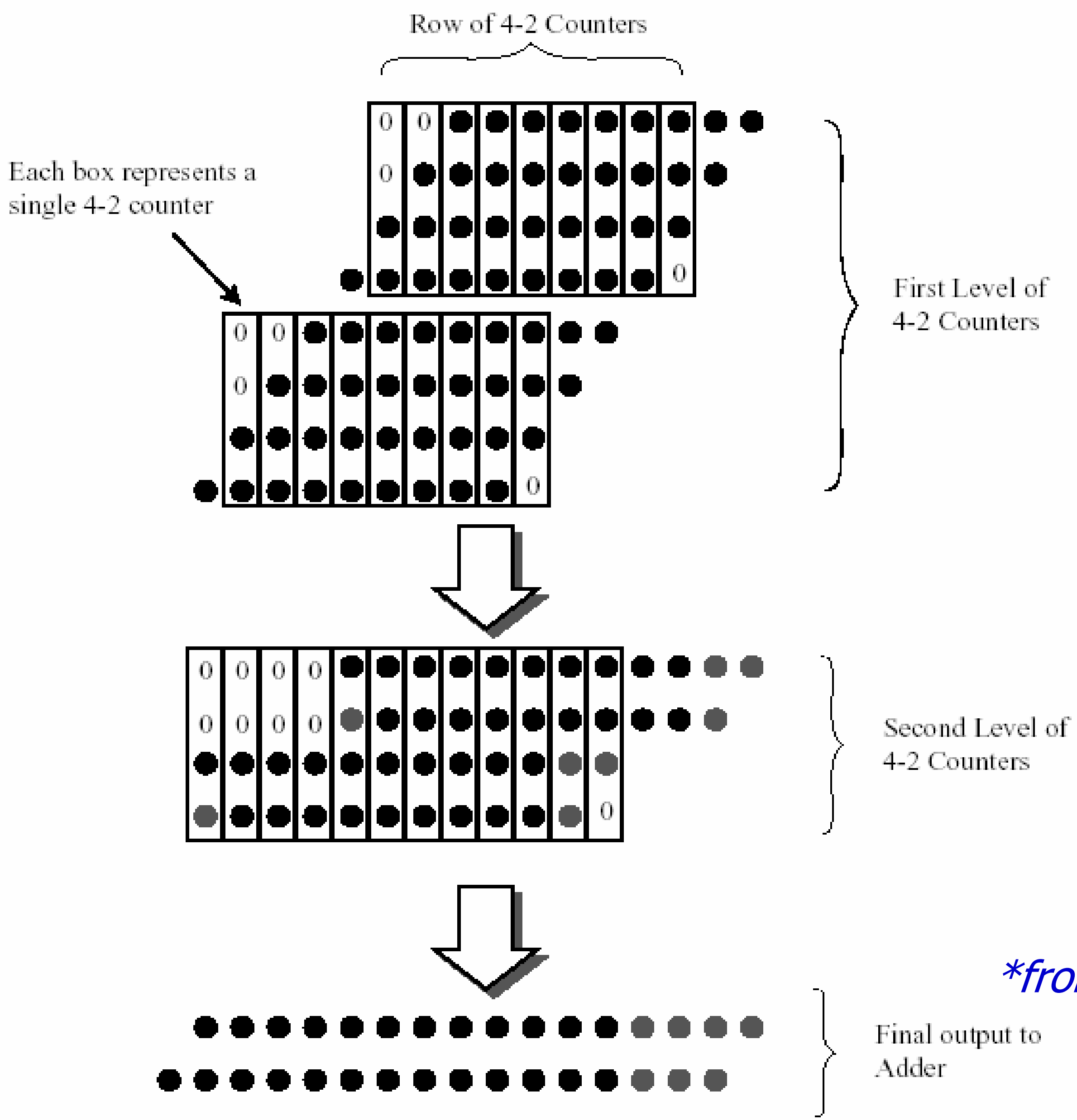
Tree Multipliers



Tree multiplier with a more regular structure based on 4-to-2 reduction modules.

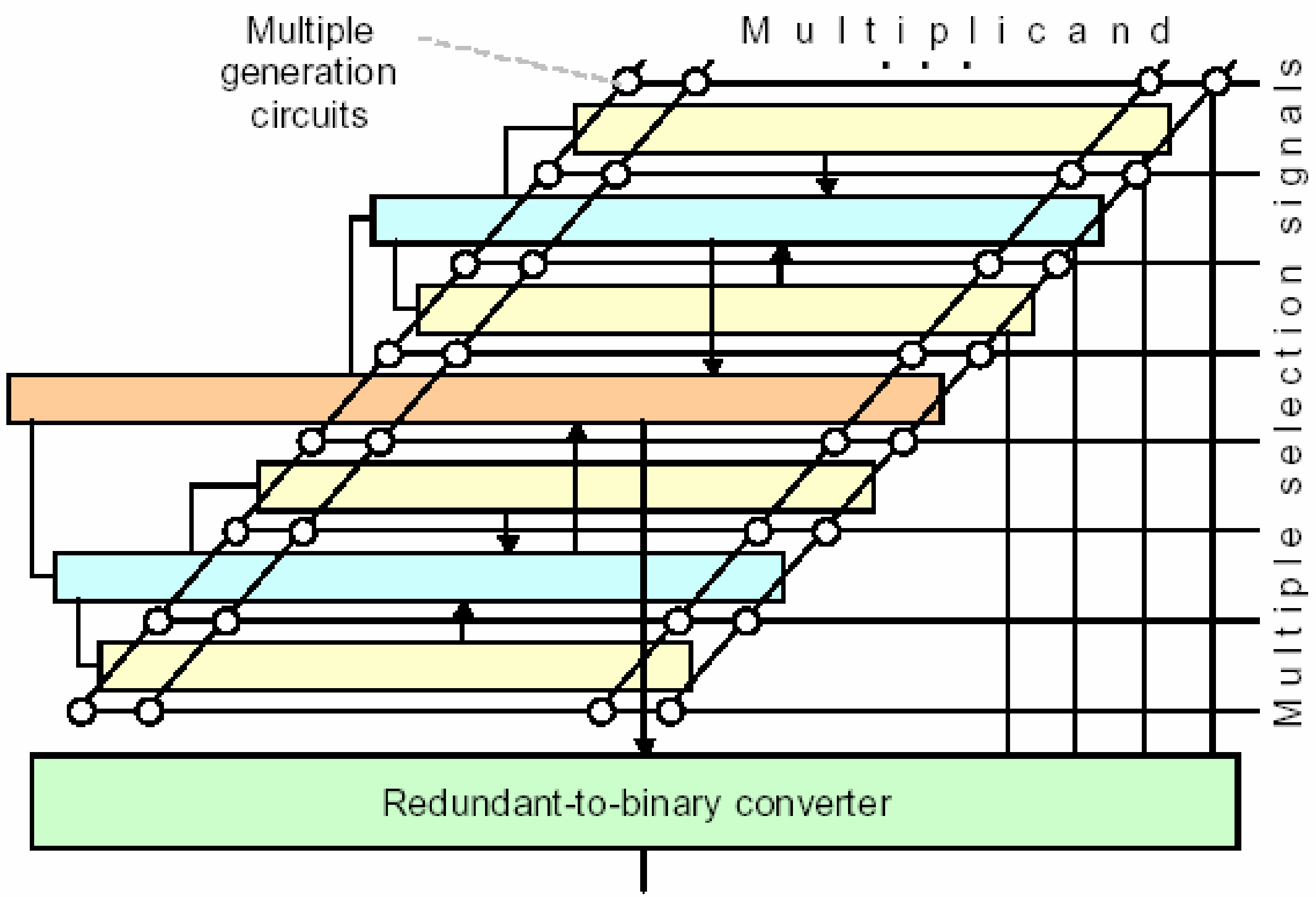
**from Parhami*

Reduction using 4:2 Compressors



*from G. Bewick

Tree Multipliers



Layout of a partial-products reduction tree composed of 4-to-2 reduction modules. Each solid arrow represents two numbers.

**from Parhami*

Multiplier Placement in a Standard Grid Topology

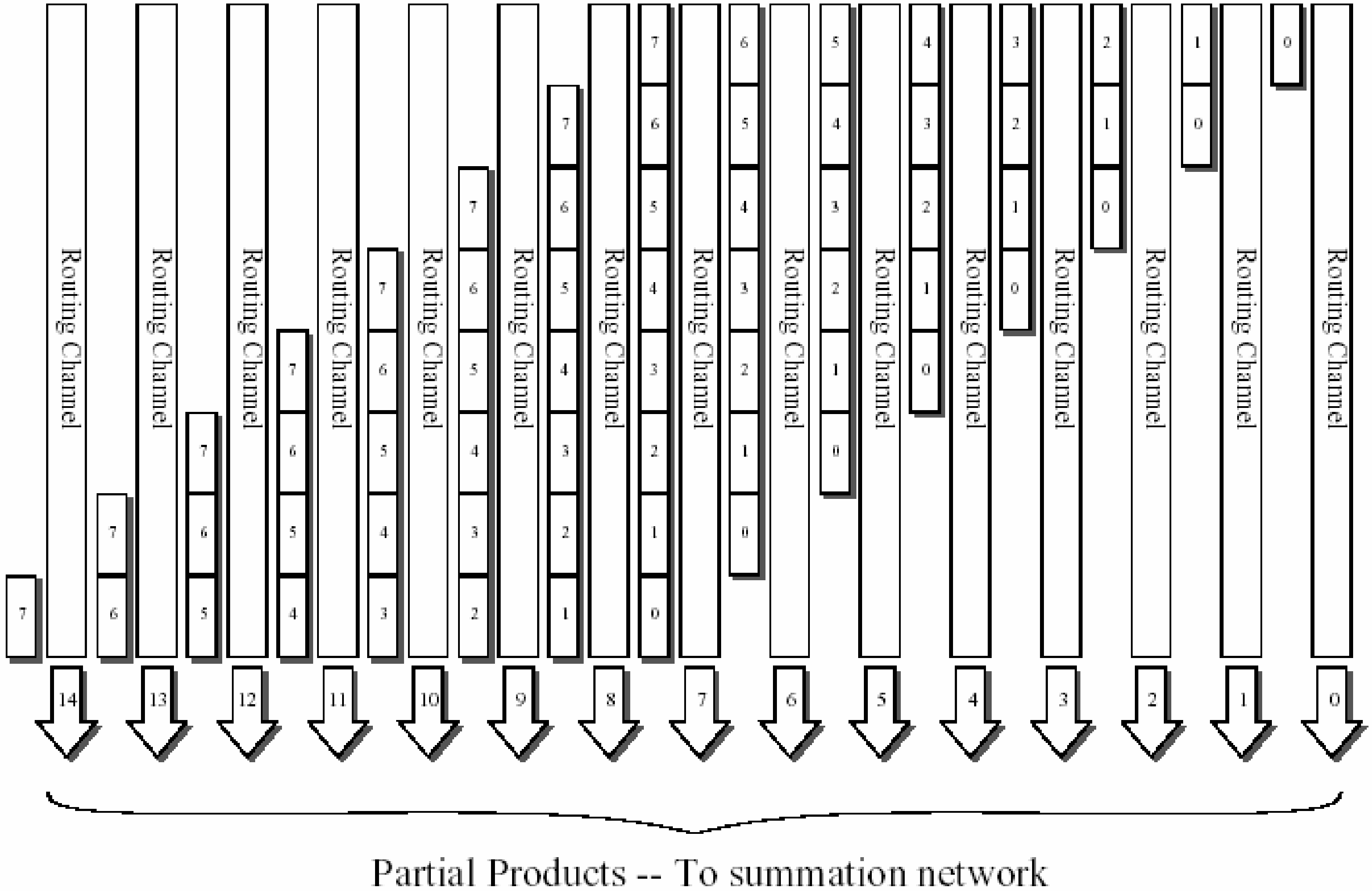


Figure 4.10: Multiplexer placement for 8x8 multiplier.

**from G. Bewick*

Floor Plan of a Multiplier

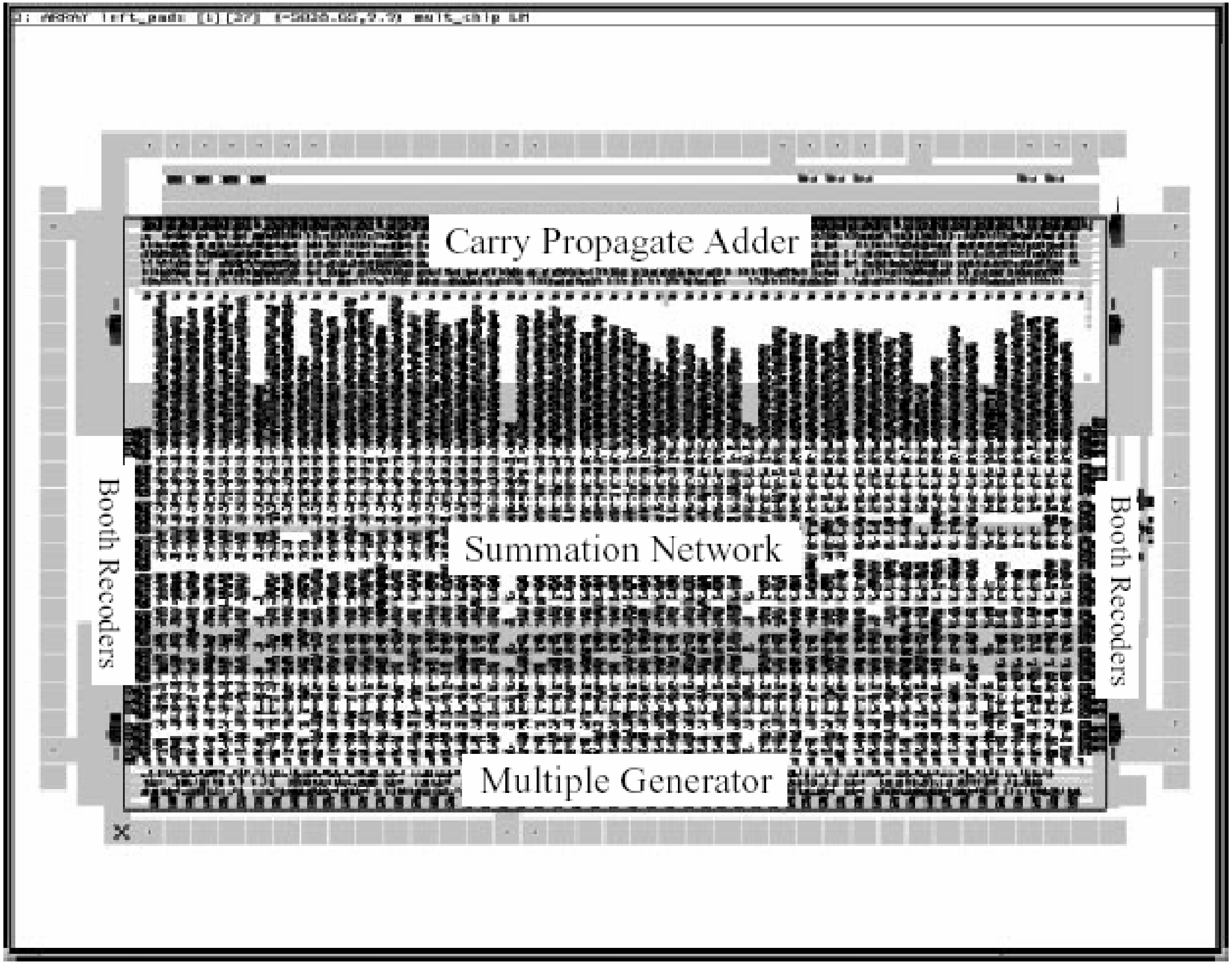


Figure 5.20: Floor plan of multiplier chip

*from G. Bewick

Delay Components of a Booth Recoded Parallel Multiplier

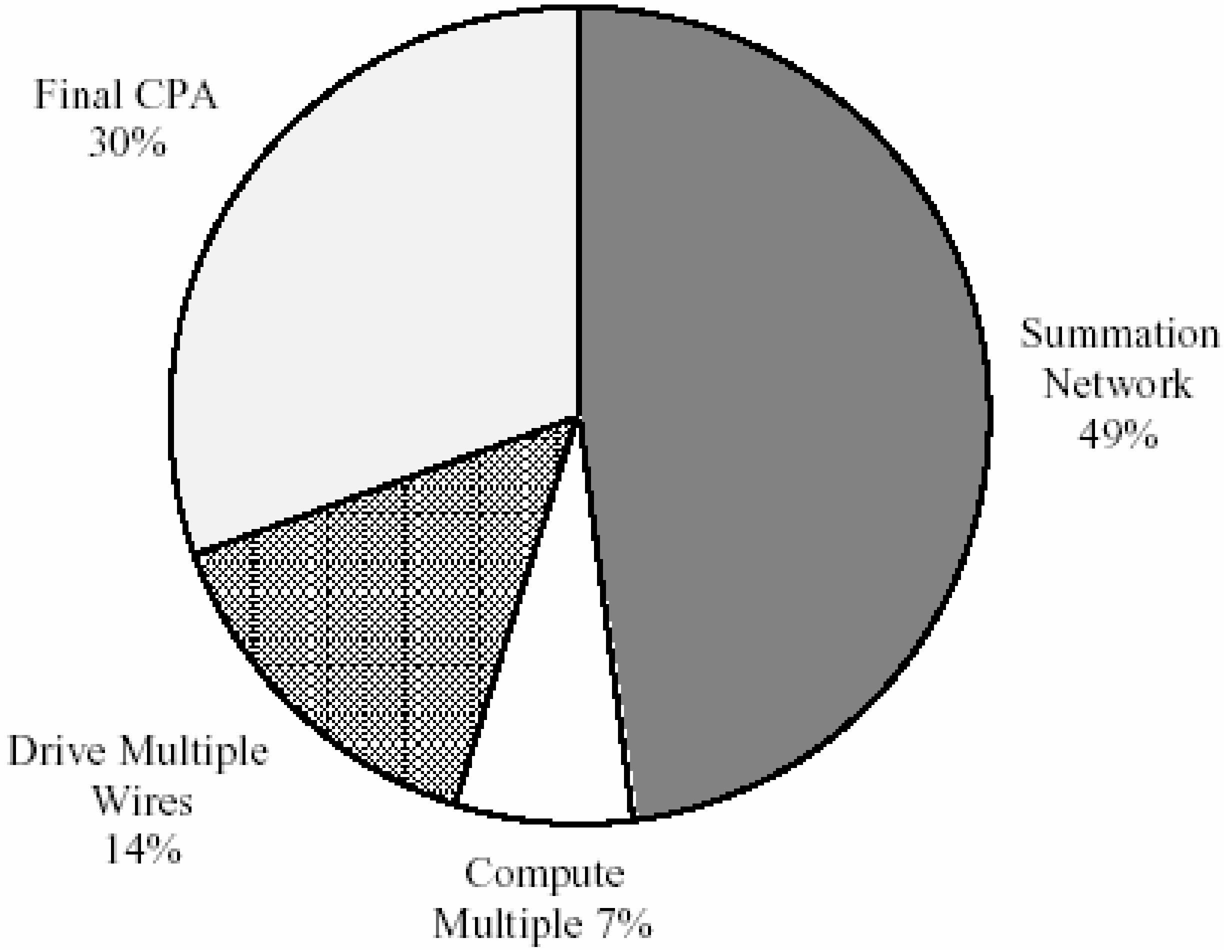
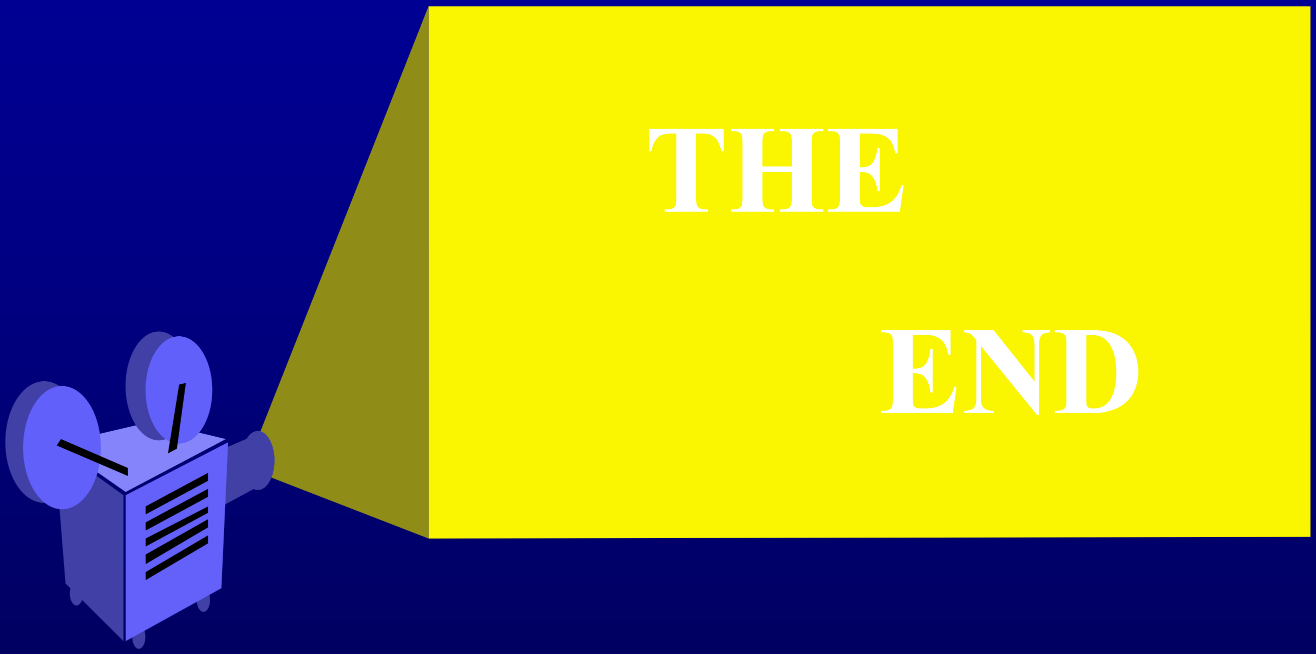


Figure 6.1: Delay components of Booth 3-14 multiplier.

*from G. Bewick


Hollywood


THE
END