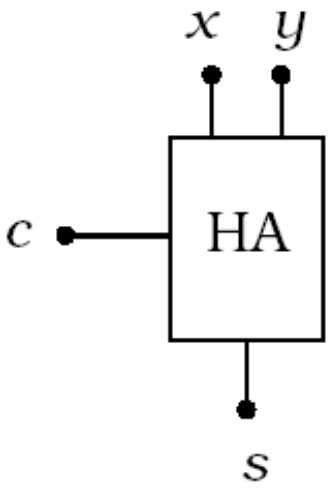
ARITHMETIC CIRCUITS IN CMOS VLSI

Faculty of Engineering
Alexandria University

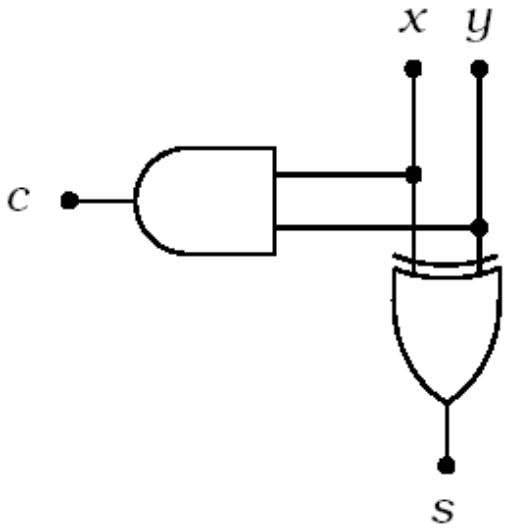
Adders



x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

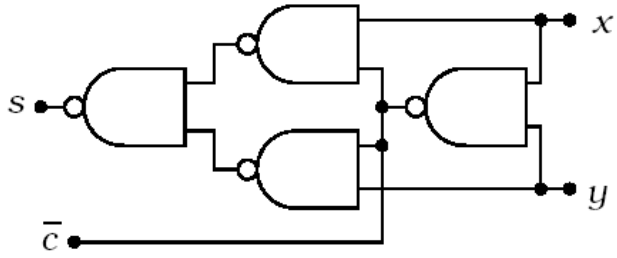
Half-adder symbol and operation.

Adders (2)

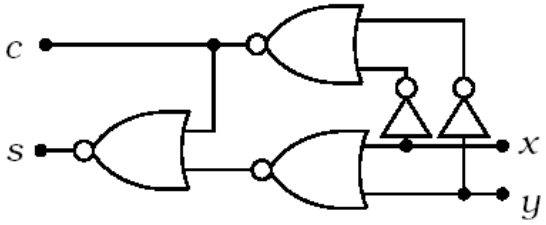


Half-adder logic diagram.

Adders (3)



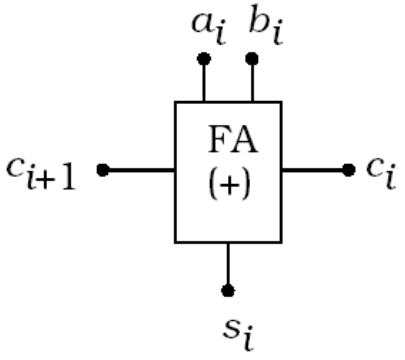
(a) NAND2 logic



(b) NOR-based network

Alternate half-adder logic networks.

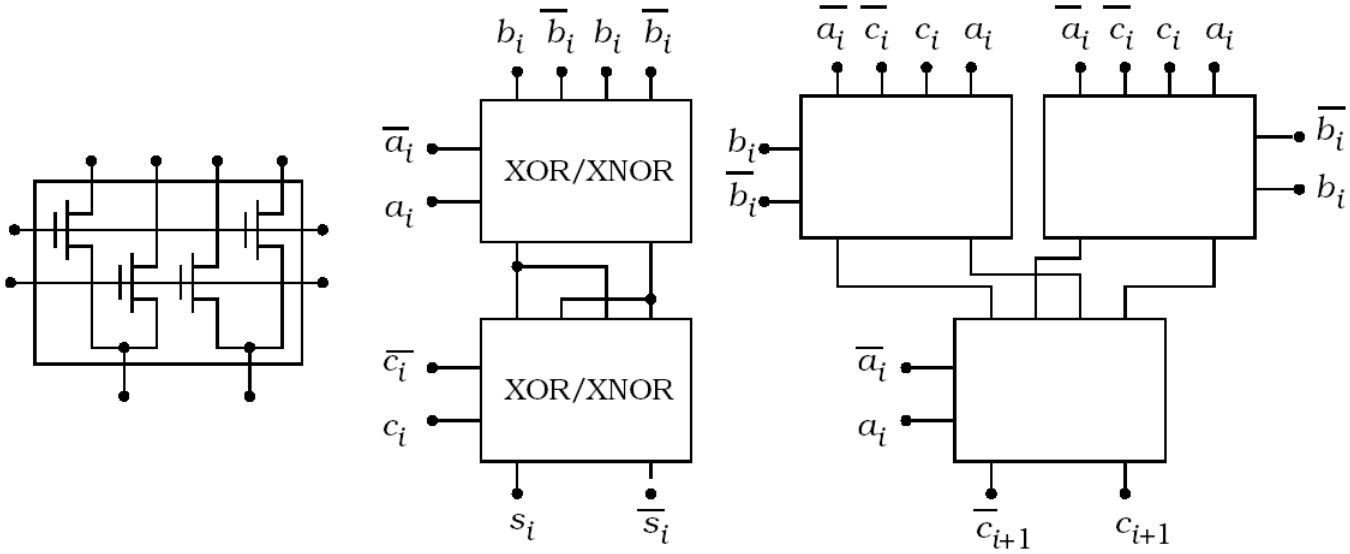
Adders (4)



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

Full-adder symbol and function table.

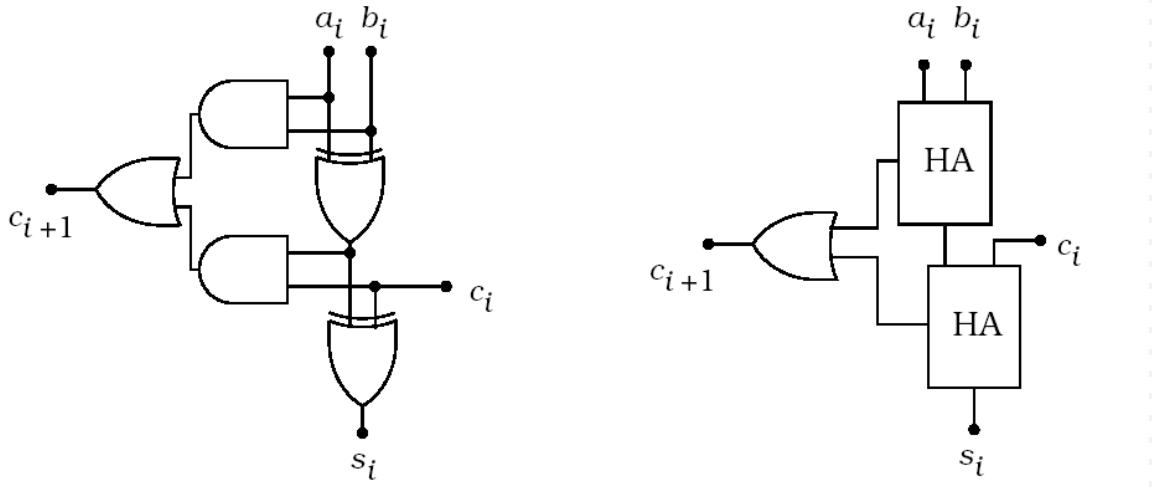
Adders (5)



(a) 2-input array (b) Sum circuit (c) Carry circuit

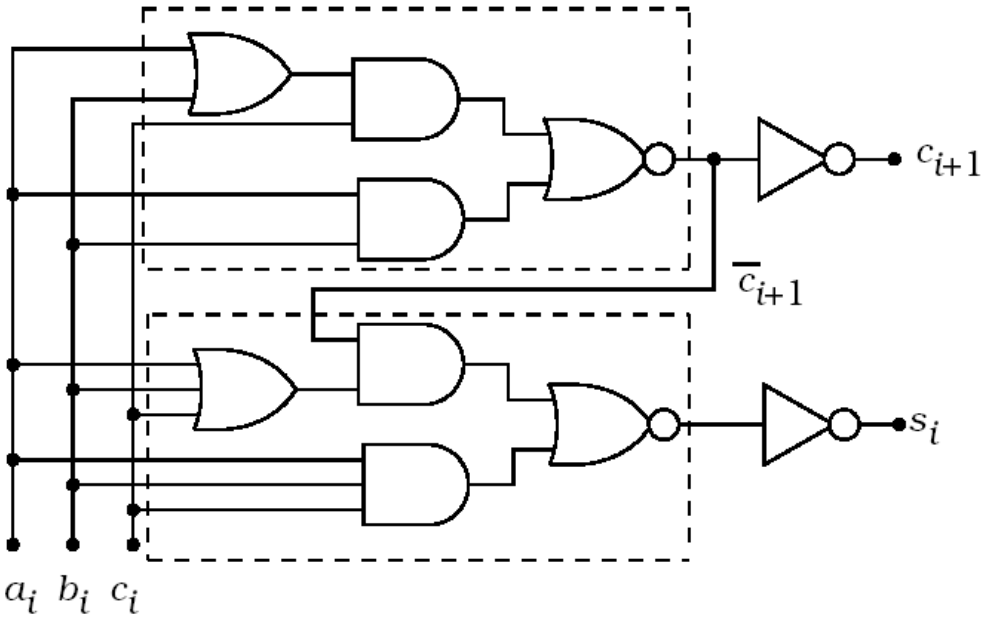
CPL full-adder design.

Adders (6)



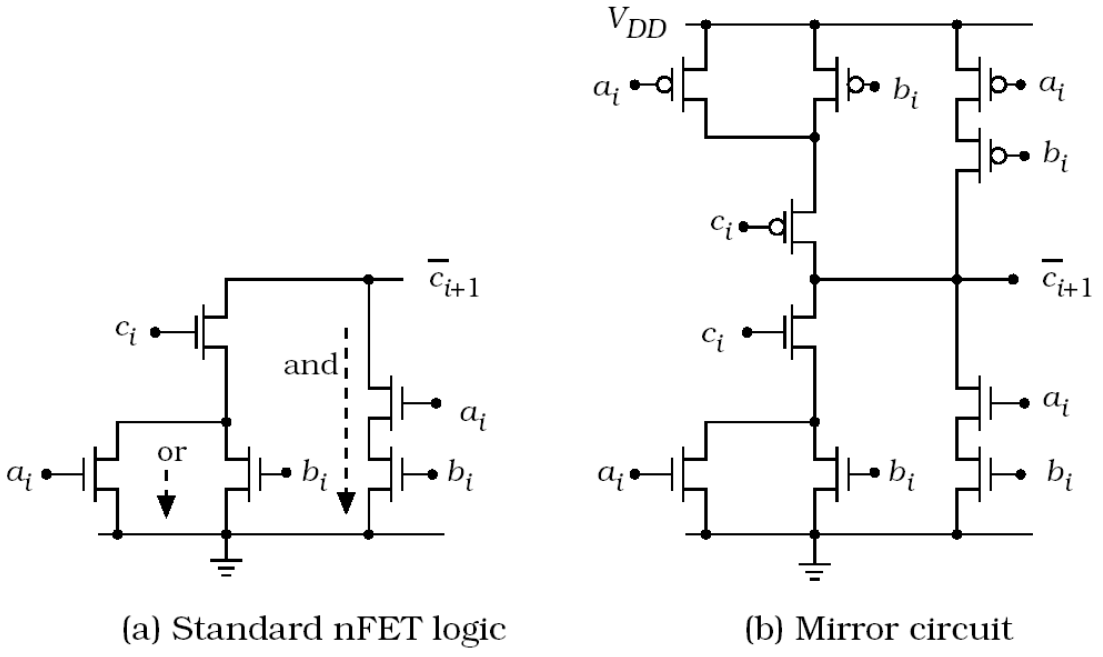
Full-adder logic networks.

Adders (7)



AOI full-adder logic.

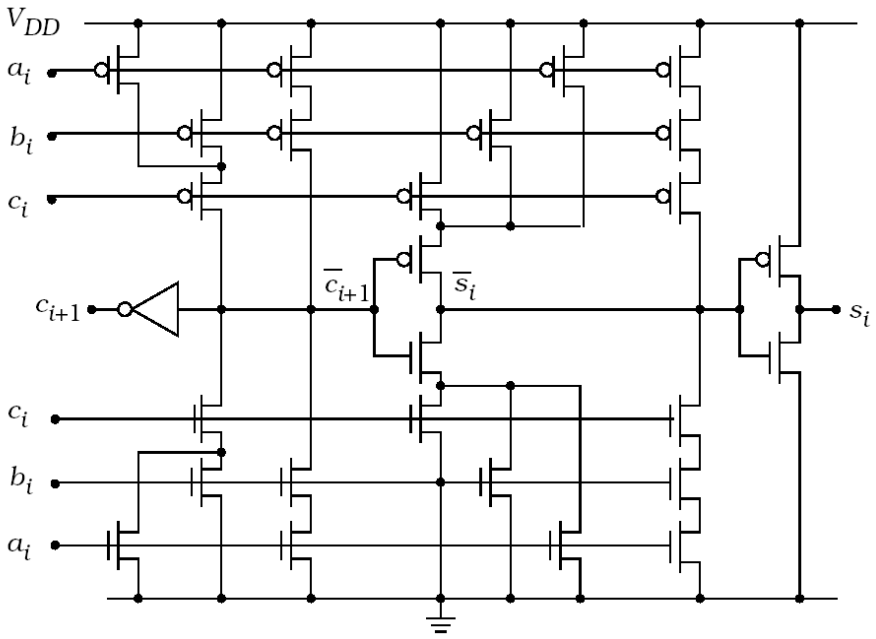
Adders (8)



(a) Standard nFET logic (b) Mirror circuit

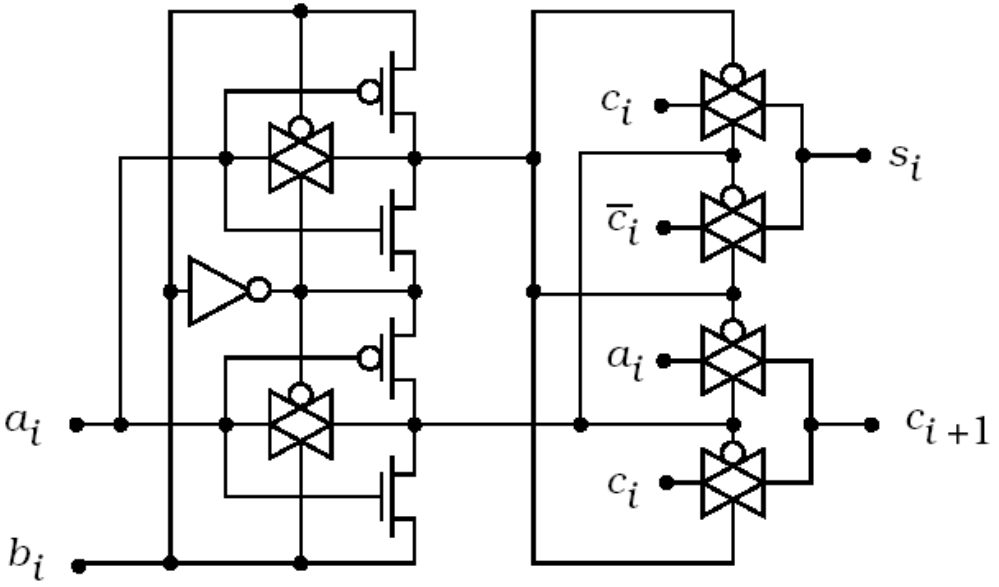
Evolution of carry-out circuit.

Adders (9)



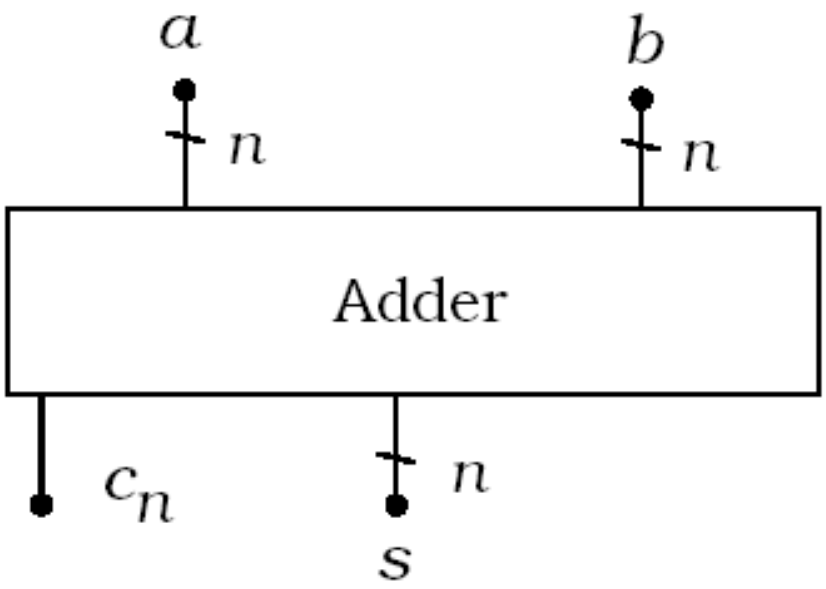
Mirror AOI CMOS full-adder.

Adders (10)



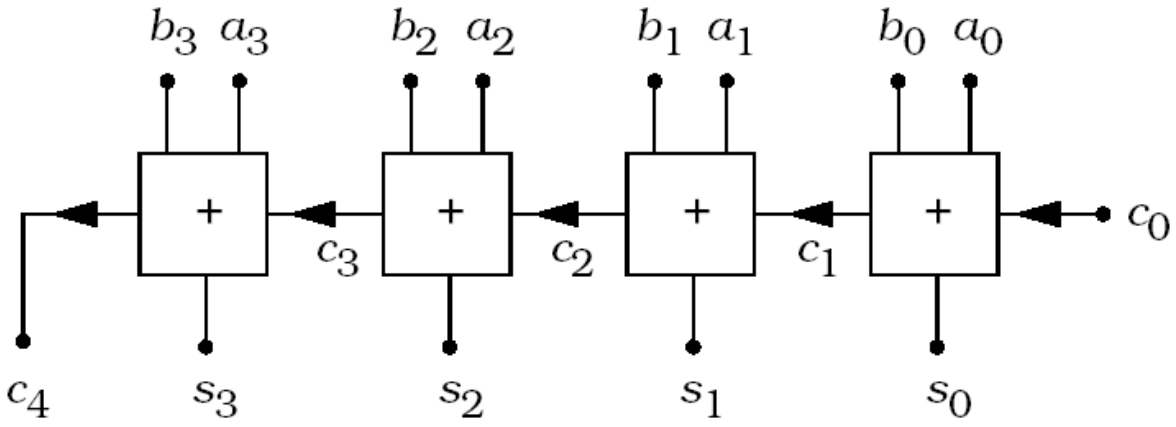
Transmission-gate full-adder circuit.

Adders (II)



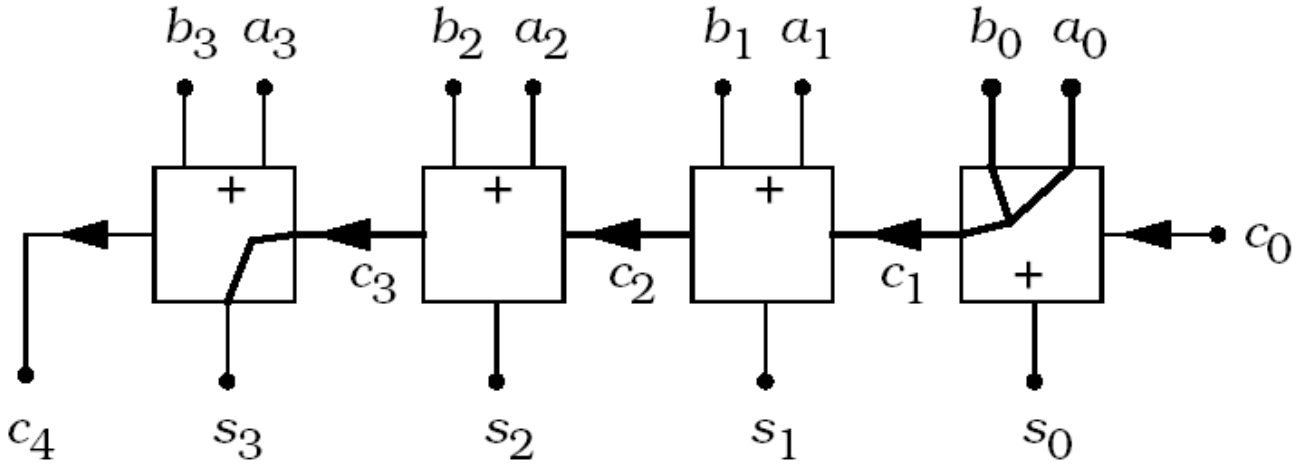
An *n*-bit adder.

Adders (12)



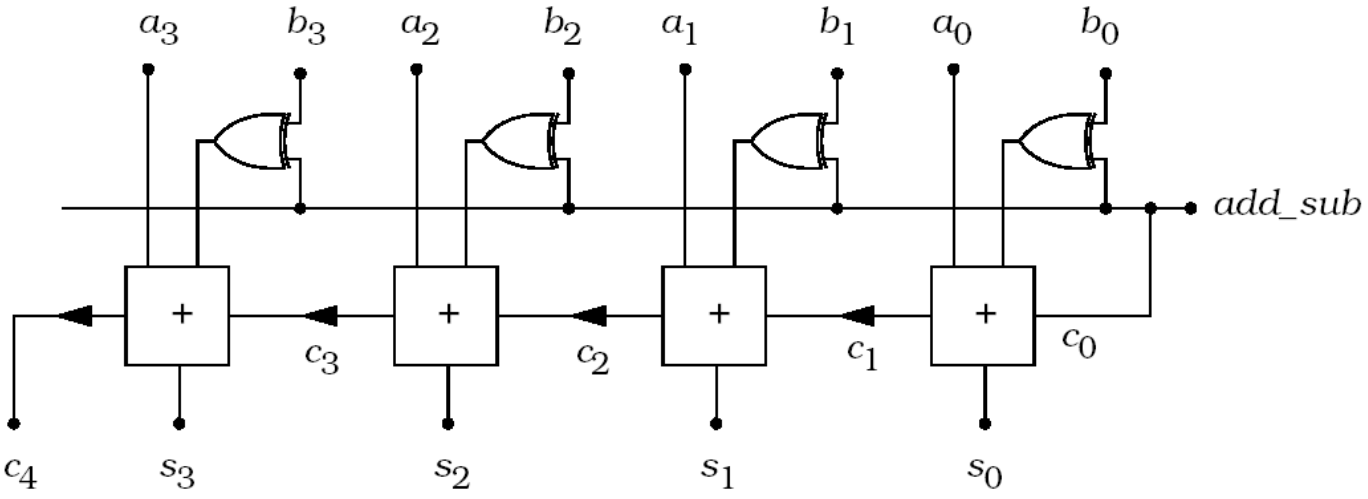
A 4-bit ripple-carry adder.

Adders (13)



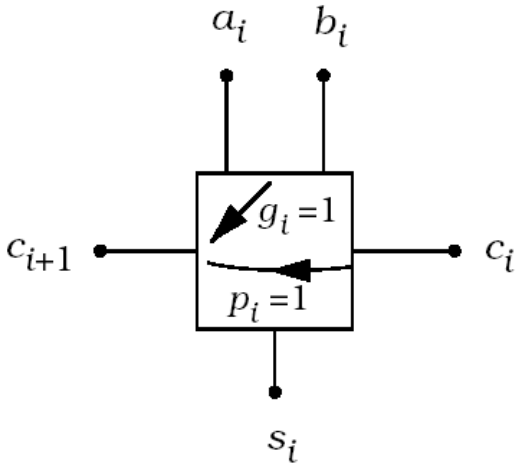
Worst-case delay through the 4-bit ripple adder.

Adders (14)



4-bit adder-subtractor circuit.

Adders (15)

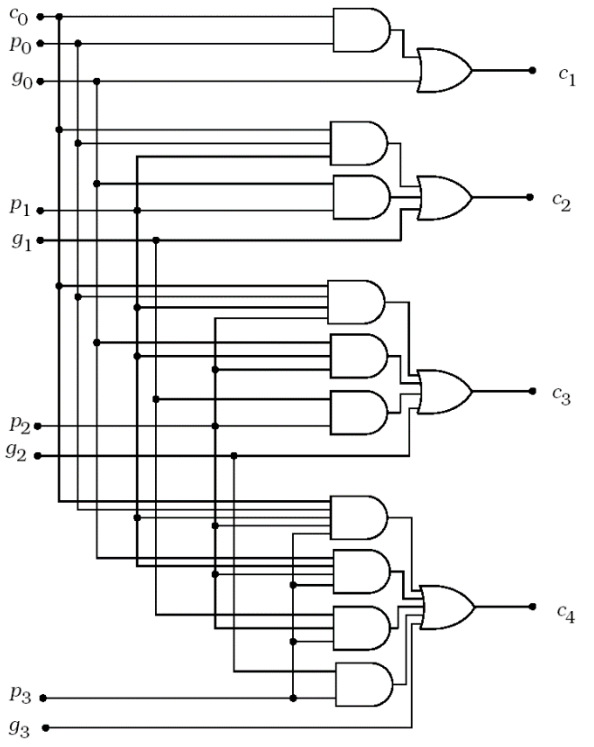


	g_i	p_i
	$a_i \cdot b_i$	$a_i \oplus b_i$
$a_i = b_i = 0$	0	0
$a_i = b_i = 1$	1	0
$a_i \neq b_i$	0	1

$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i)$$

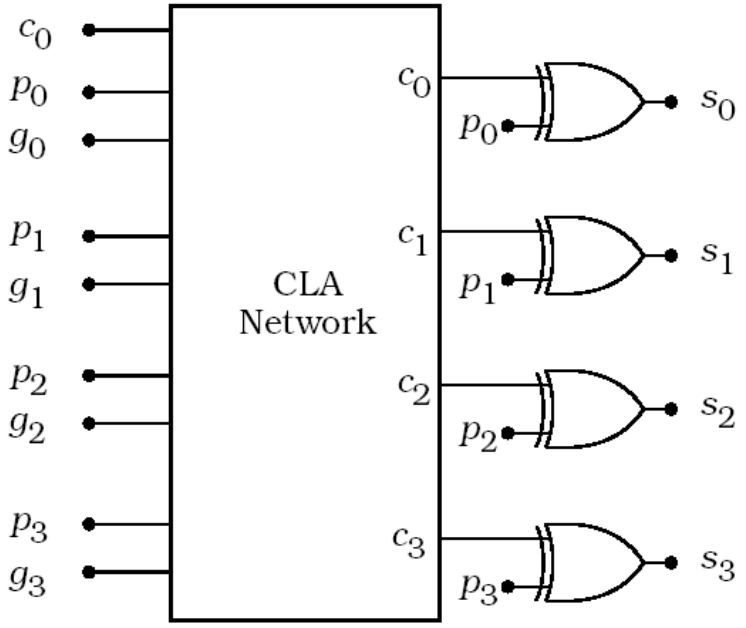
A basis of the carry look-ahead algorithm.

Adders (16)



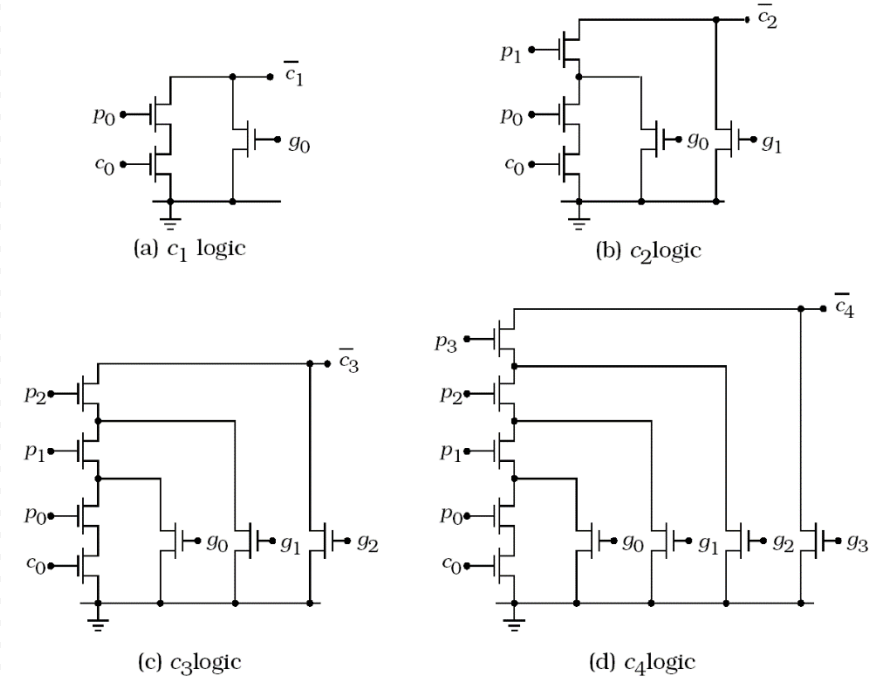
Logic network for 4-bit CLA carry l

Adders (17)



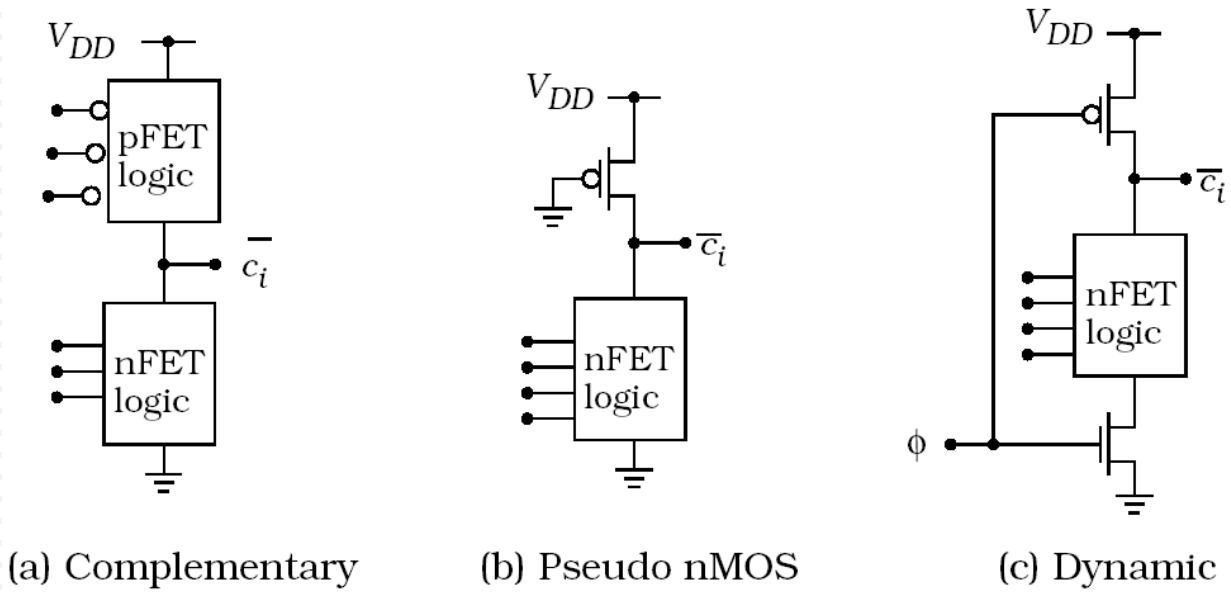
Sum calculation using the CLA network.

Adders (18)



nFET logic arrays for the CLA terms.

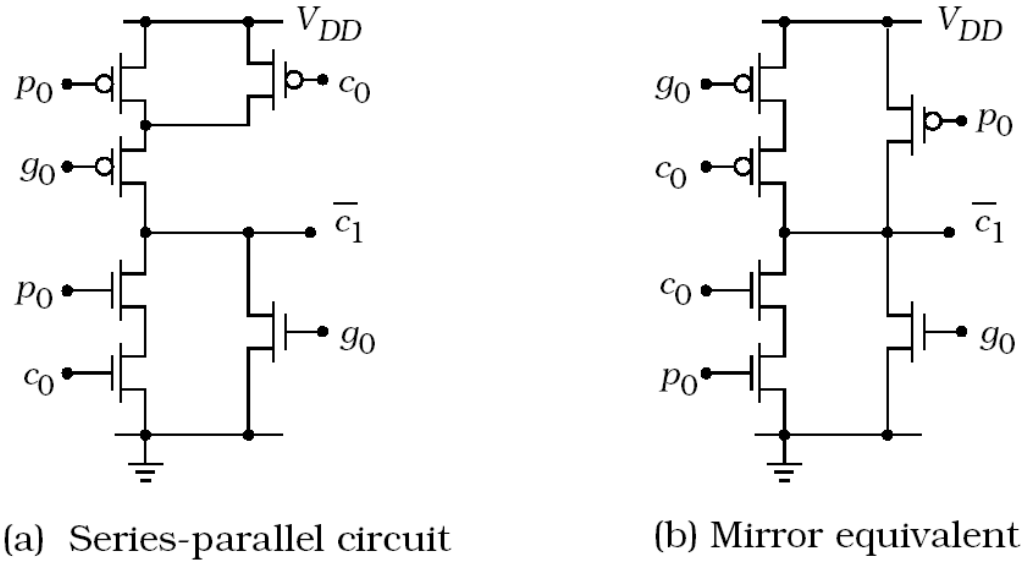
Adders (19)



(a) Complementary (b) Pseudo nMOS (c) Dynamic

Possible uses of the nFET logic arrays in Figure 12.18.

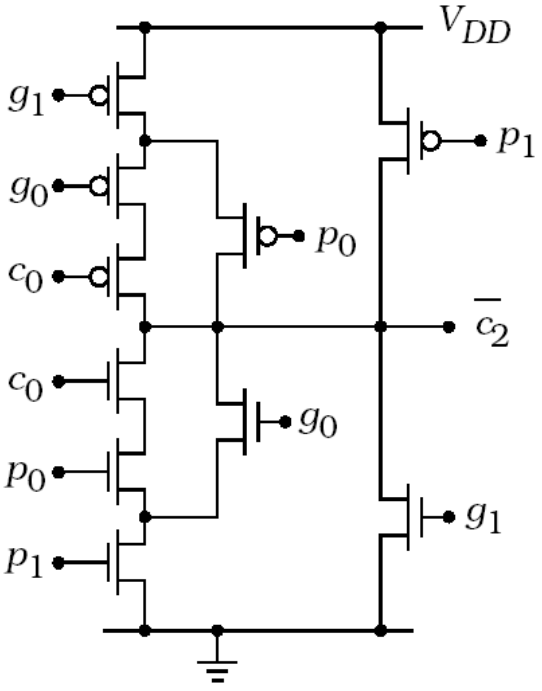
Adders (20)



(a) Series-parallel circuit (b) Mirror equivalent

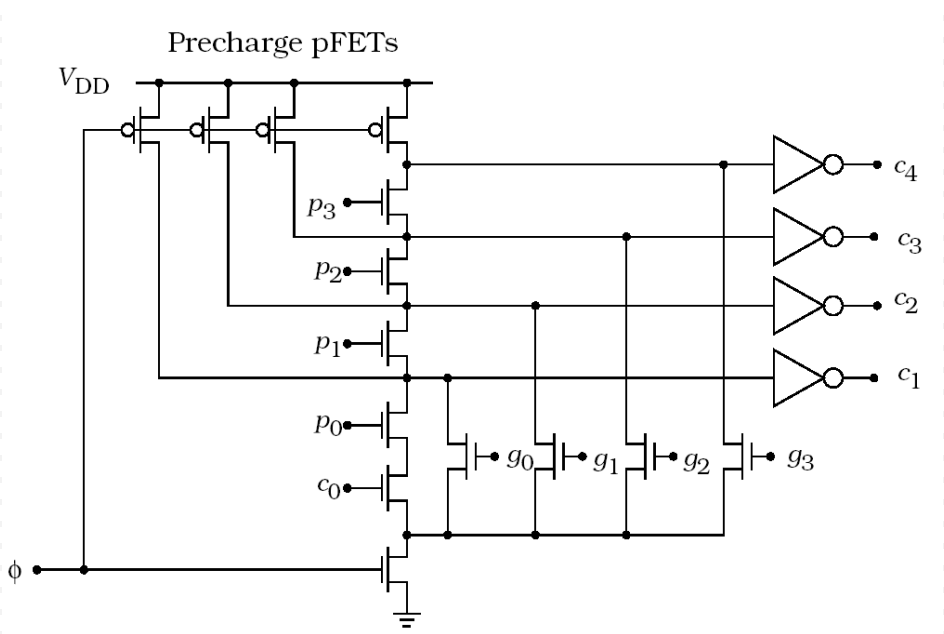
Static CLA mirror circuit.

Adders (21)



Static mirror circuit for c_2.

Adders (22)



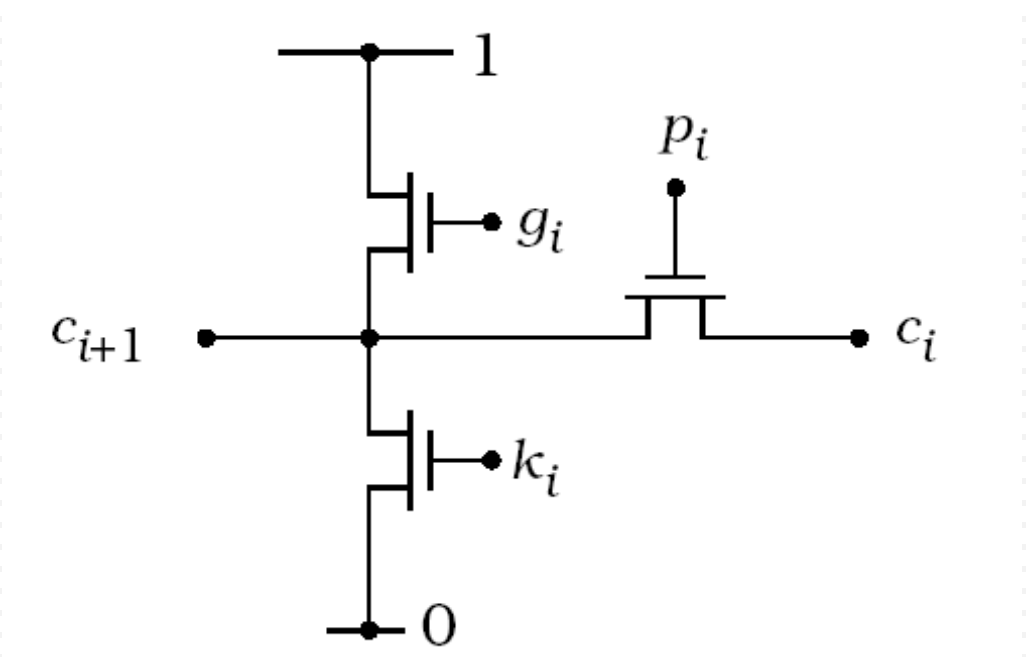
MODL carry circuit.

Adders (23)

a_i	b_i	p_i	g_i	k_i
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	0	1	0

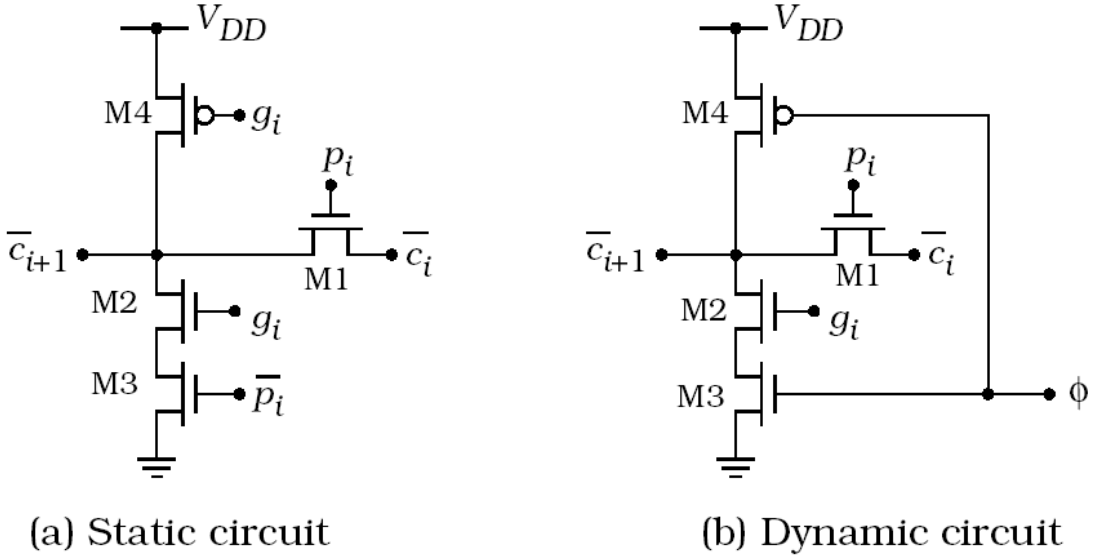
Propagate, generate, and carry-kill values

Adders (24)



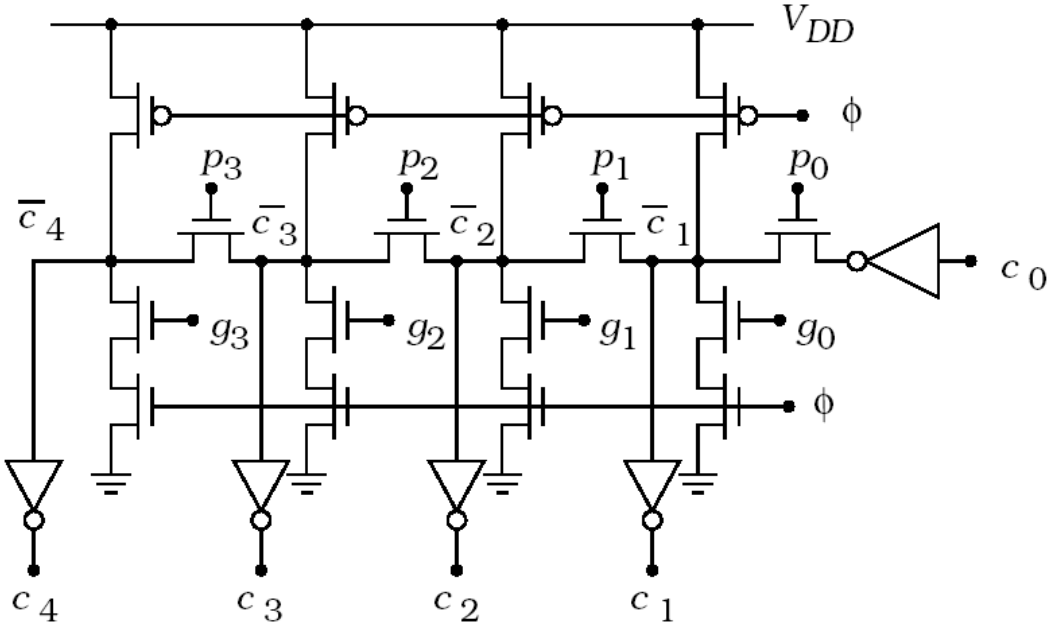
Switching network for the carry-out equation.

Adders (25)



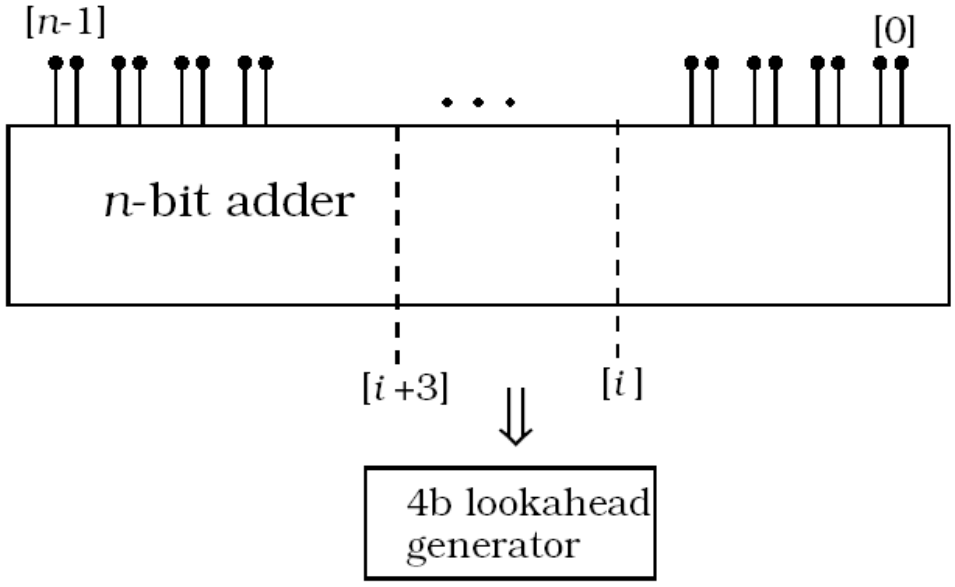
Manchester circuit styles.

Adders (26)



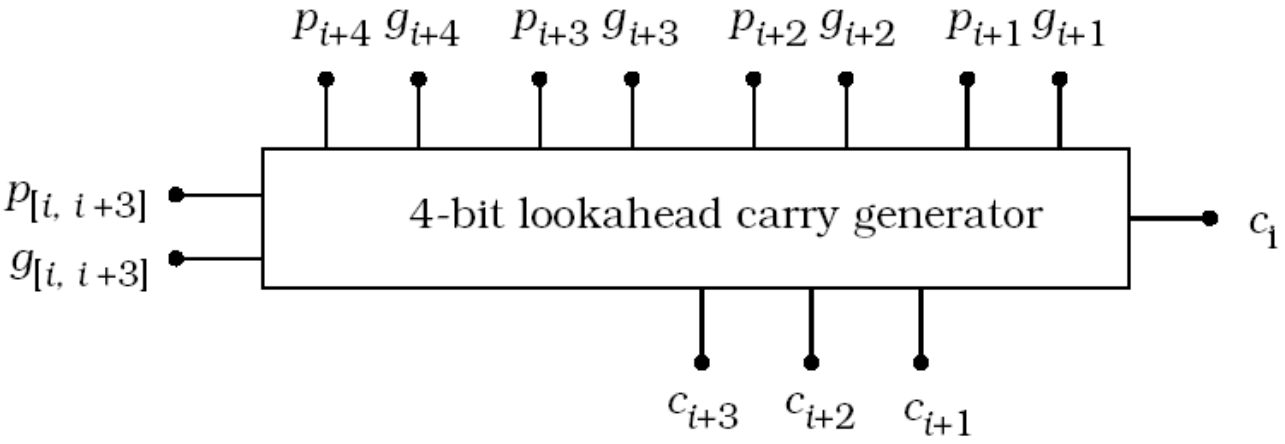
Dynamic Manchester carry chain.

Adders (27)



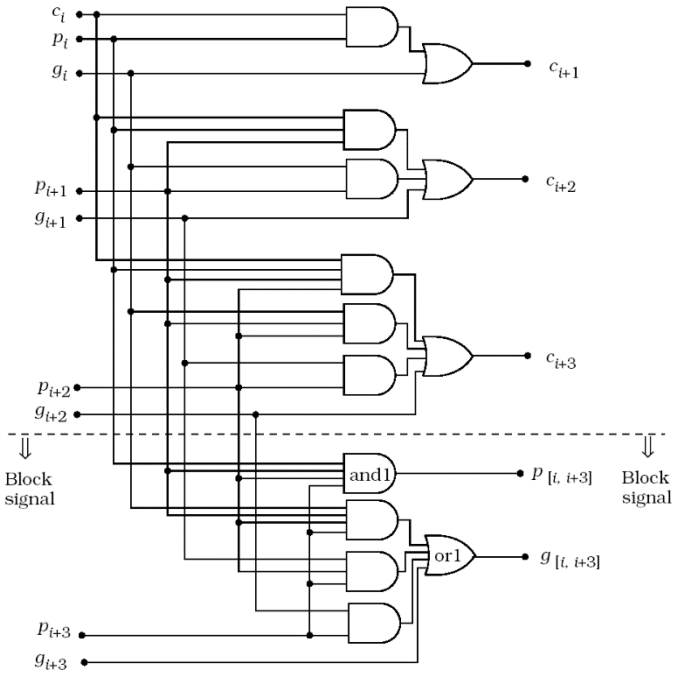
An *n*-bit adder network.

Adders (28)



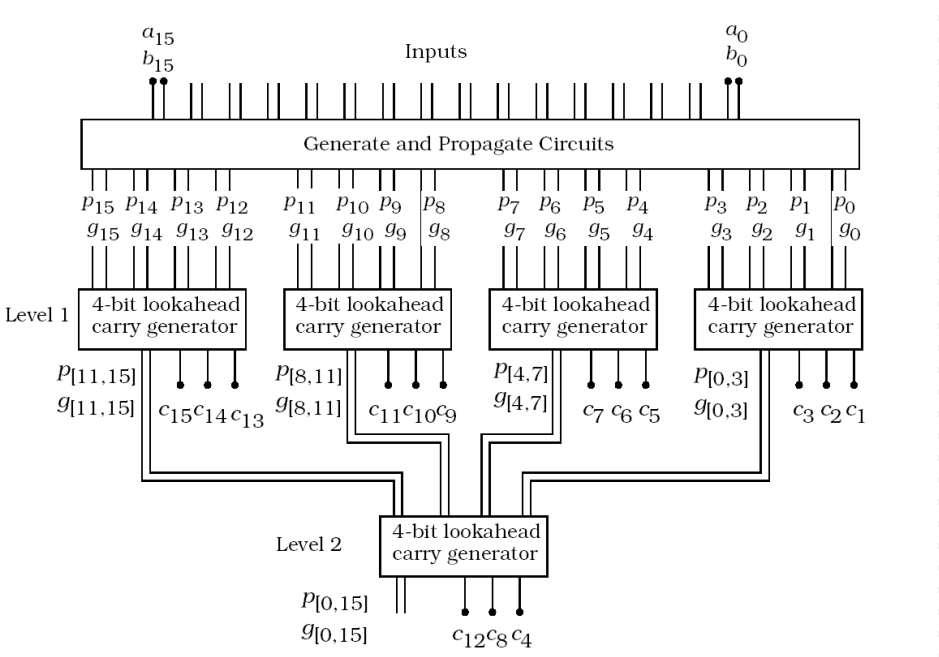
4-bit lookahead carry generator signals.

Adders (29)



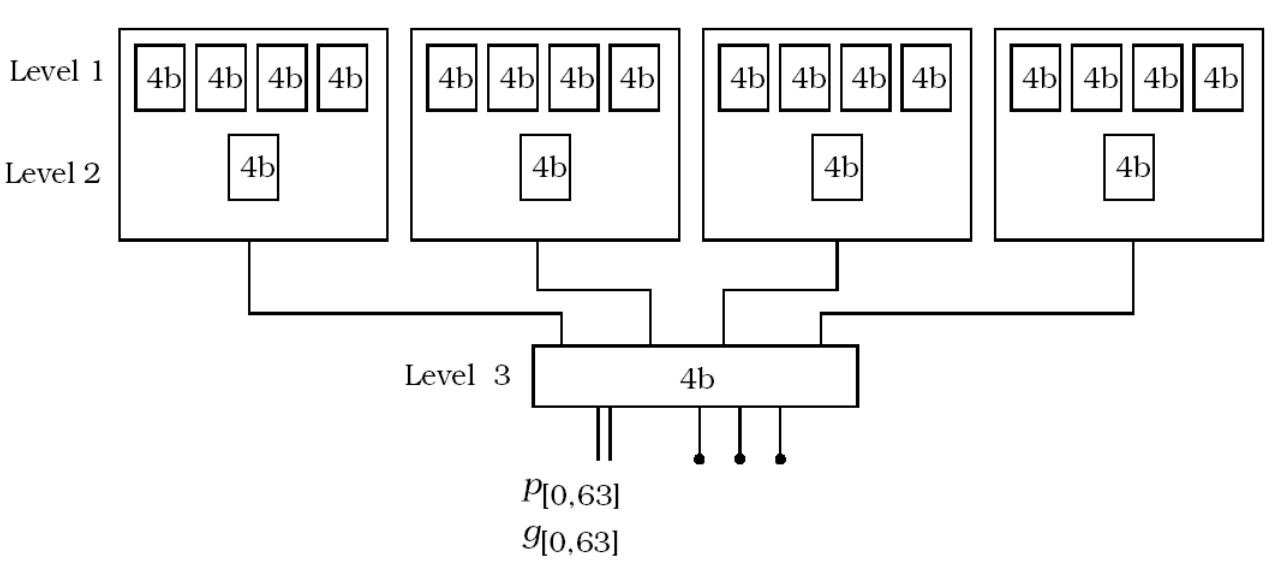
Block lookahead generator logic.

Adders (30)



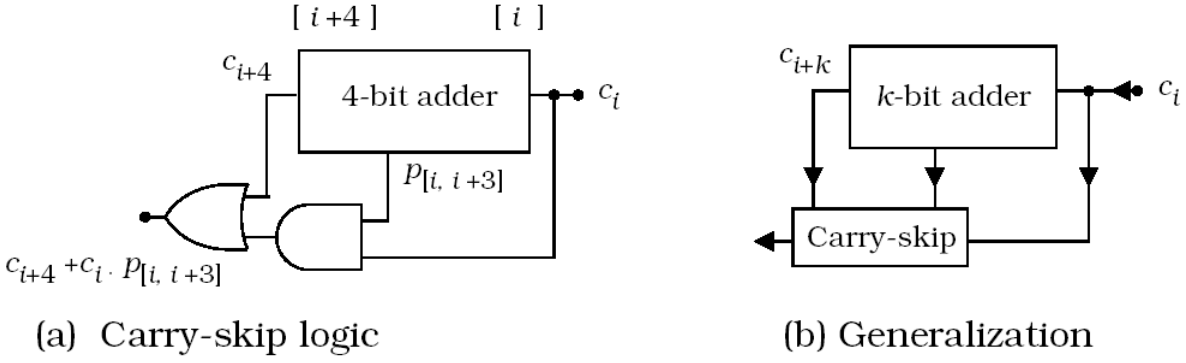
Multilevel CLA block scheme for a 16-bit adder.

Adders (31)



64-bit CLA adder architecture.

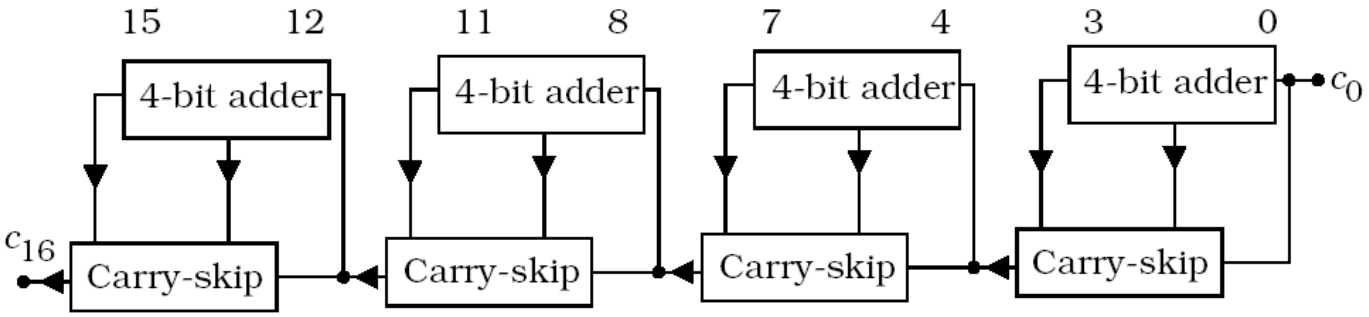
Adders (32)



(a) Carry-skip logic (b) Generalization

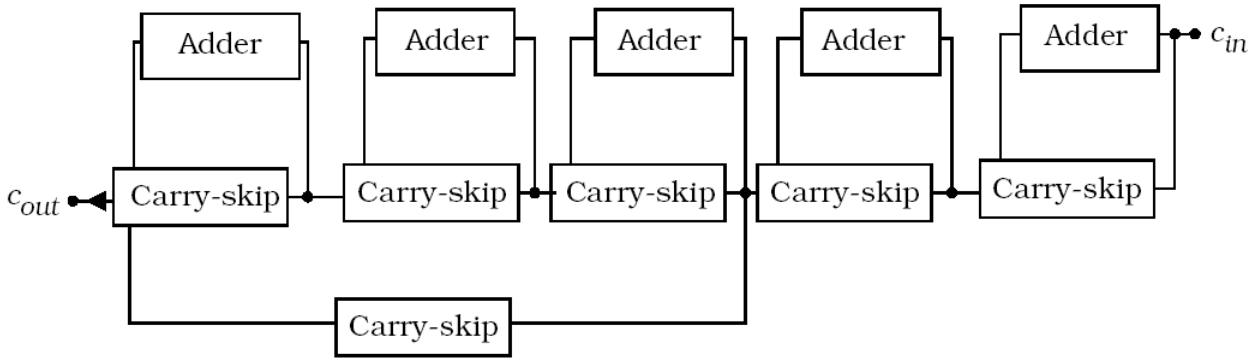
Carry-skil circuitry.

Adders (33)



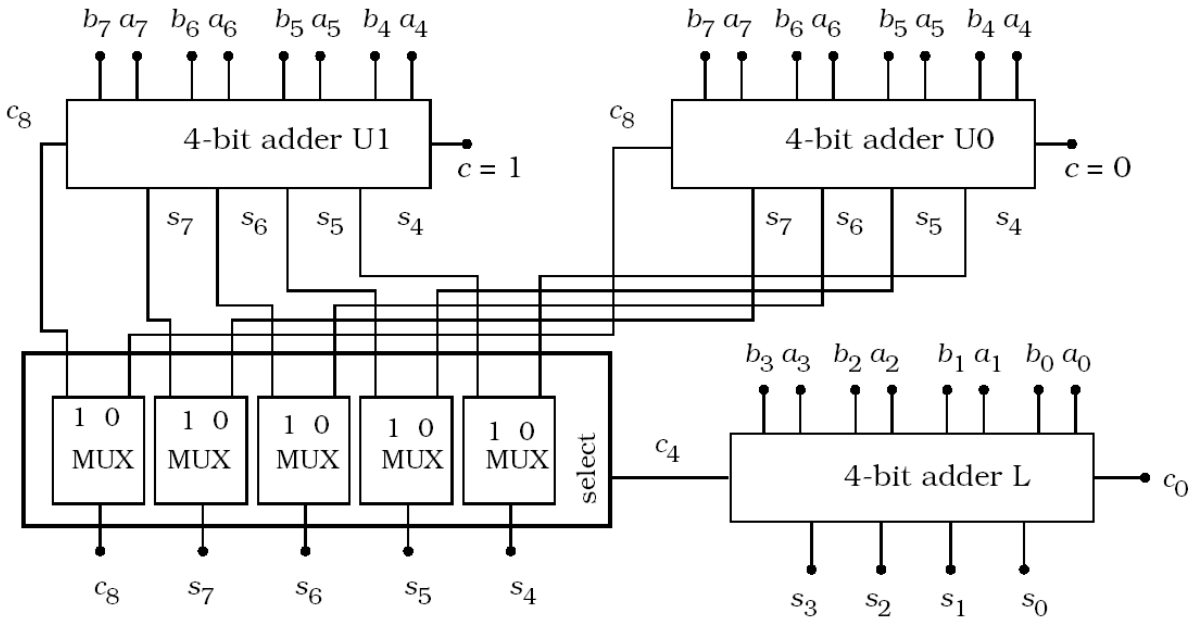
A 16-bit adder using carry-skip circuits.

Adders (34)



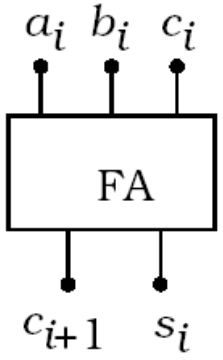
A 2-level carry-skip adder.

Adders (35)

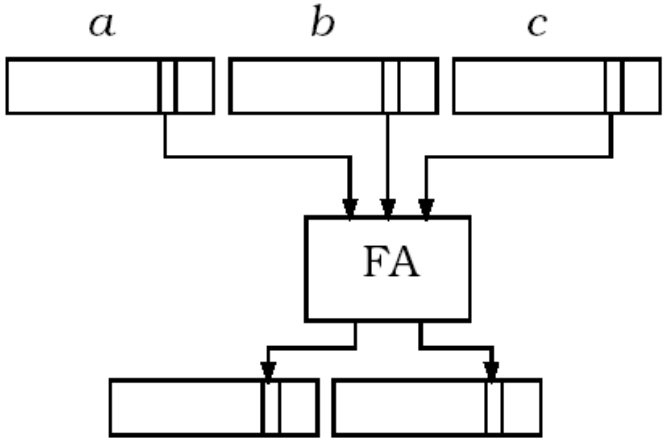


A8-bit carry-select adder.

Adders (35)



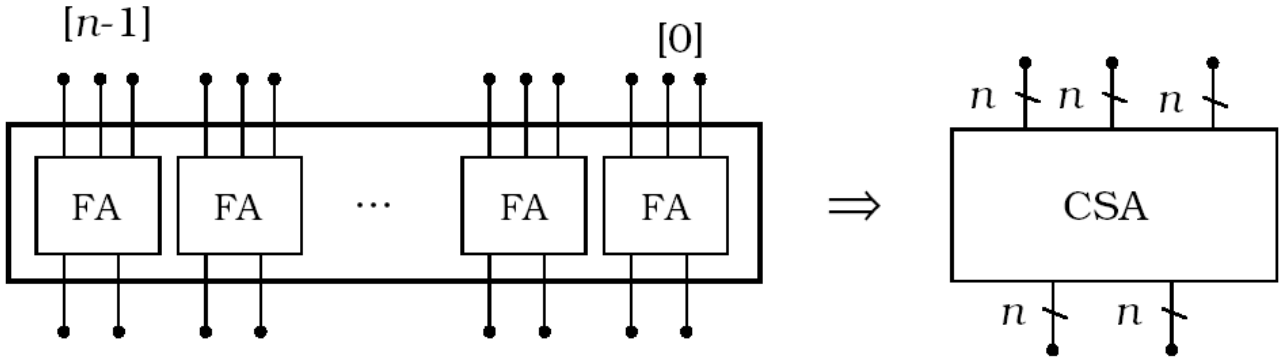
(a) Symbol



(b) 3-to-2 reduction

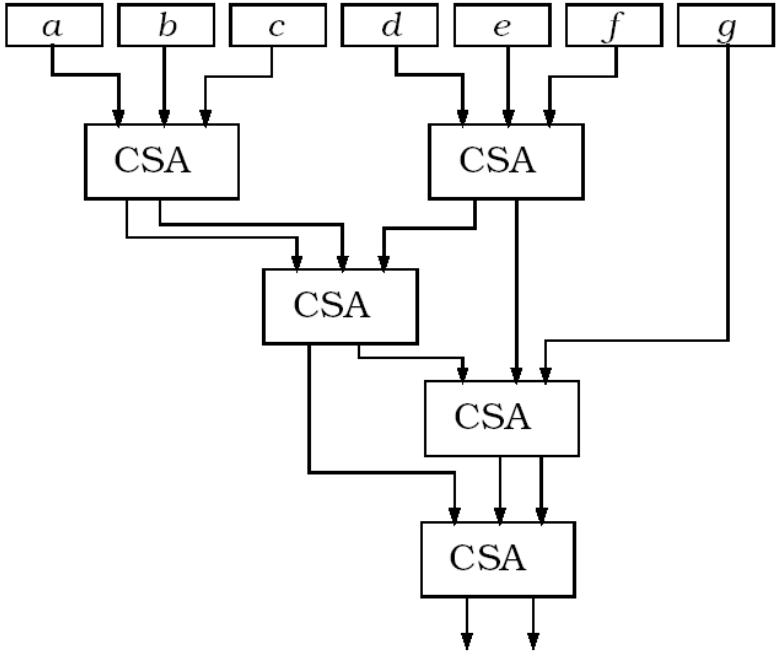
Basis of a carry-save adder.

Adders (36)



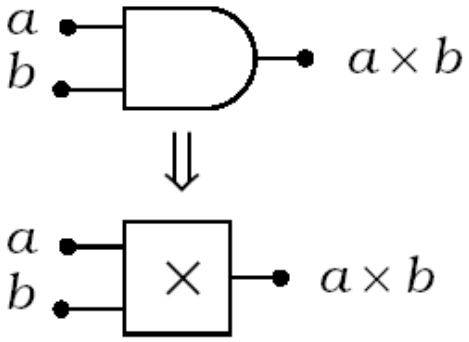
Creation of an *n*-bit carry-save adder.

Adders (37)



A 7-to-12 reduction using carry-save adders.

Multipliers



a	b	$a \times b$
0	0	0
0	1	0
1	0	0
1	1	1

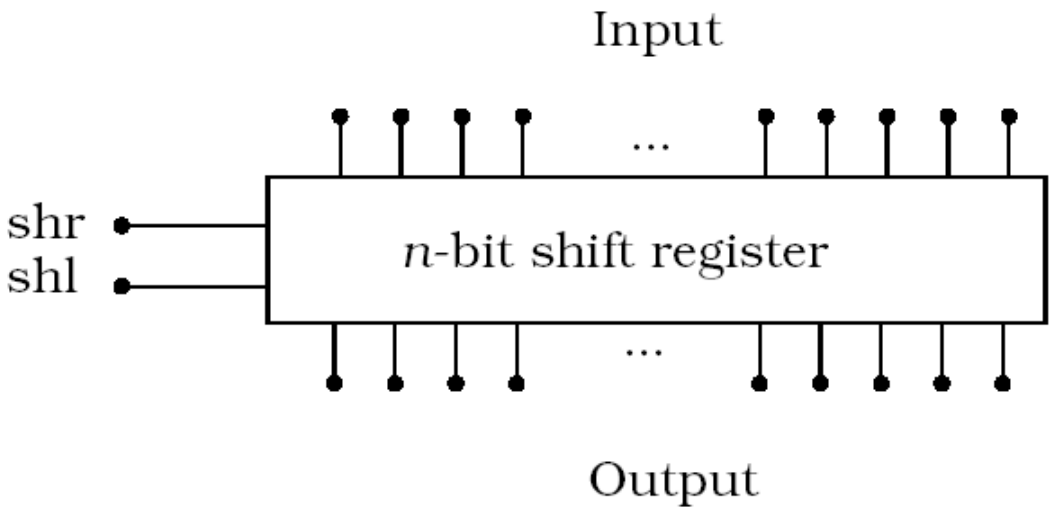
Bit-level multiplier.

Multipliers (2)

$$
\begin{array}{cccccccc}
 & & & a_3 & a_2 & a_1 & a_0 & \text{multiplicand} \\
 & & \times & b_3 & b_2 & b_1 & b_0 & \text{multiplier} \\
\hline
 & & & a_3\,b_0 & a_2\,b_0 & a_1\,b_0 & a_0\,b_0 & \\
 & & + & a_3\,b_1 & a_2\,b_1 & a_1\,b_1 & a_0\,b_1 & \\
 & & + a_3\,b_2 & a_2\,b_2 & a_1\,b_2 & a_0\,b_2 & & \\
 & + a_3\,b_3 & a_2\,b_3 & a_1\,b_3 & a_0\,b_3 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 \quad \text{product}
\end{array}
$$

Multiplication of two 4-bit words.

Multipliers (3)



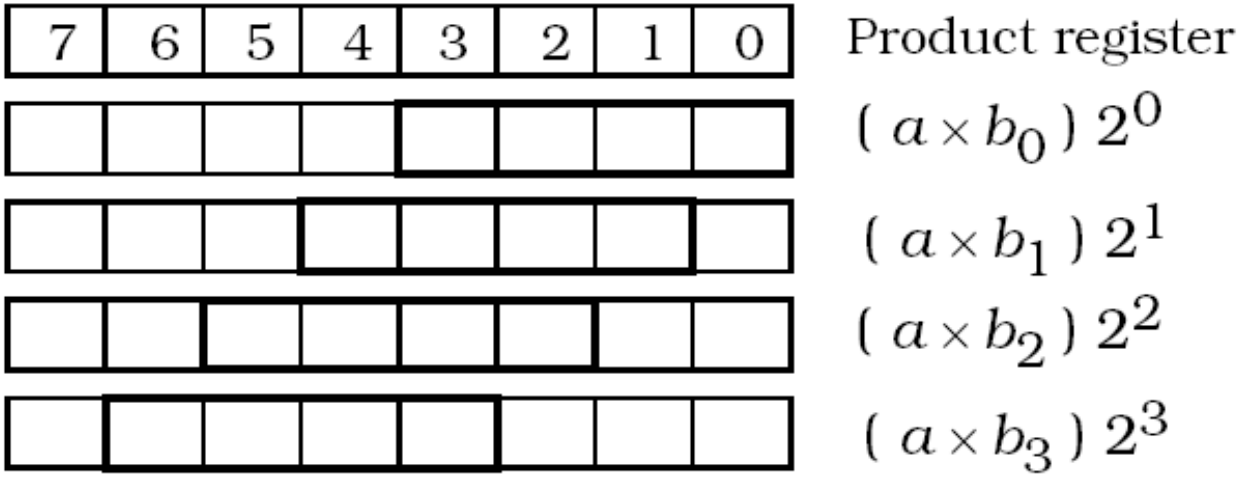
Shift register for multiplication or division by a factor of 2.

Multipliers (4)

$$
\begin{array}{rcccccccl}
 & & a_3 & a_2 & a_1 & a_0 & & & \text{multiplicand} \\
 & \times & b_3 & b_2 & b_1 & b_0 & & & \text{multiplier} \\
\hline
 & & (a_3 & a_2 & a_1 & a_0) \times b_0 & & & (a \times b_0)\,2^0 \\
 & (a_3 & a_2 & a_1 & a_0) \times b_1 & & & & (a \times b_1)\,2^1 \\
(a_3 & a_2 & a_1 & a_0) \times b_2 & & & & & (a \times b_2)\,2^2 \\
+\ (a_3 & a_2 & a_1 & a_0) \times b_3 & & & & & (a \times b_3)\,2^3 \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 & \text{product}
\end{array}
$$

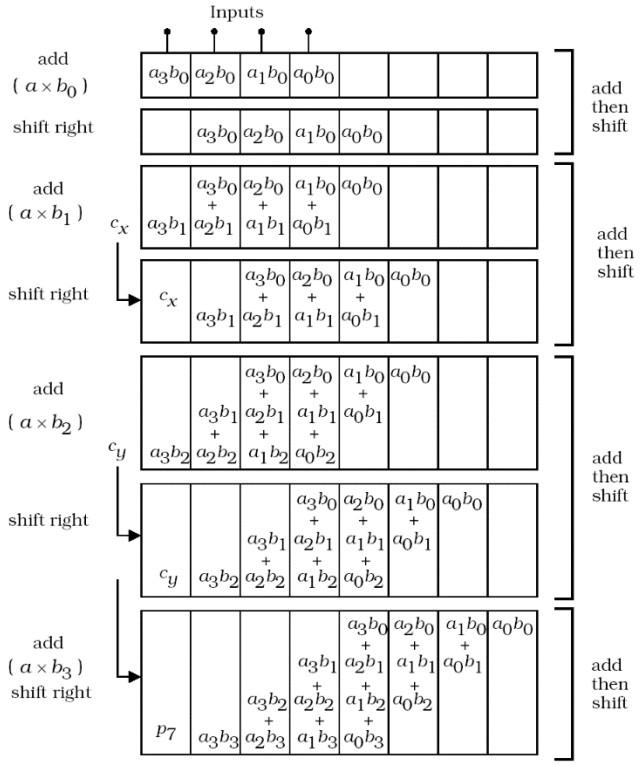
Alternate view of multiplication process.

Multipliers (5)



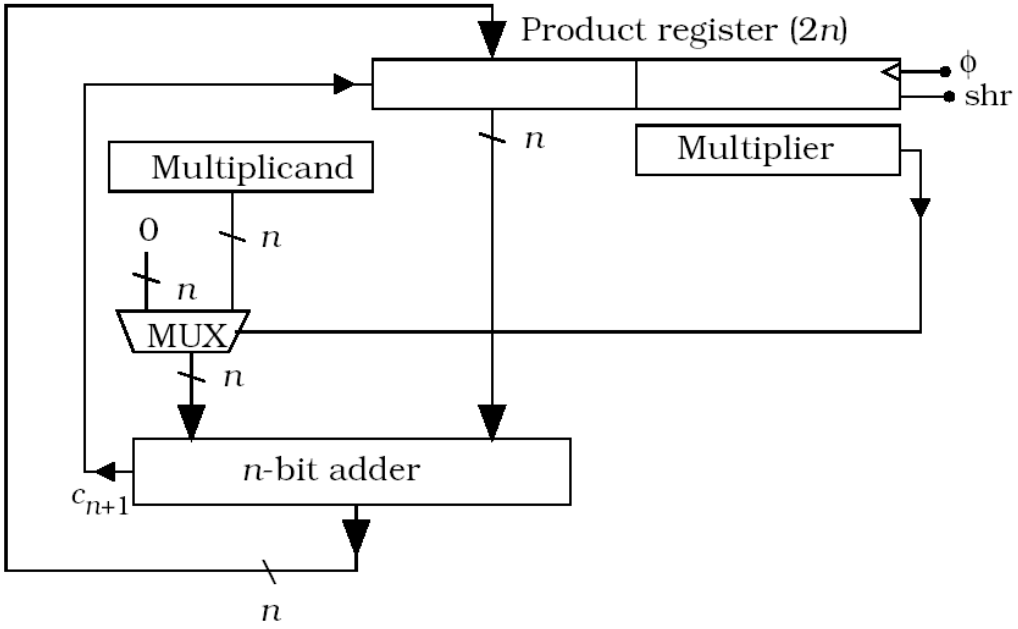
Using a product register for multiplication.

Multipliers (6)



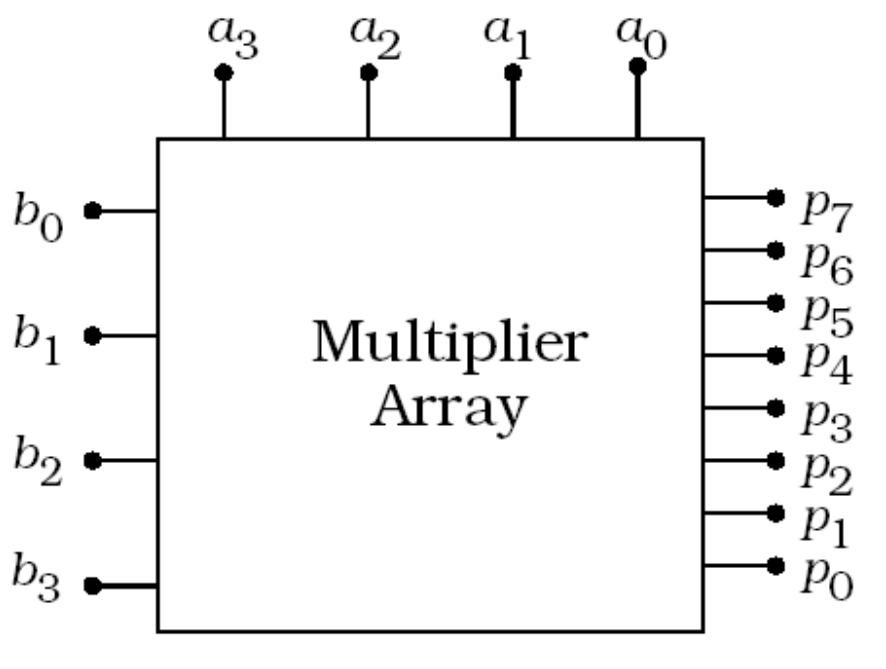
Shift-right multiplication sequence.

Multipliers (7)



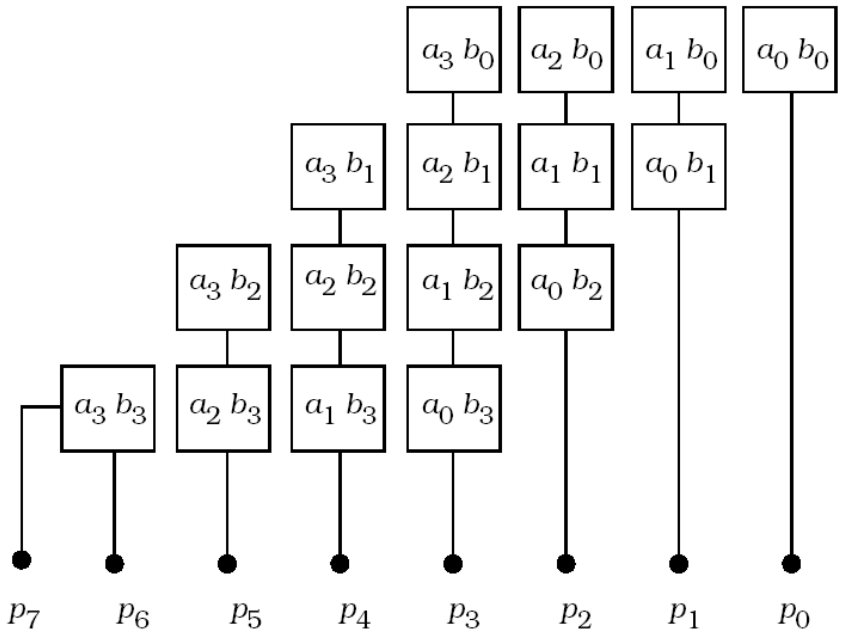
Register-based multiplier network.

Multipliers (8)



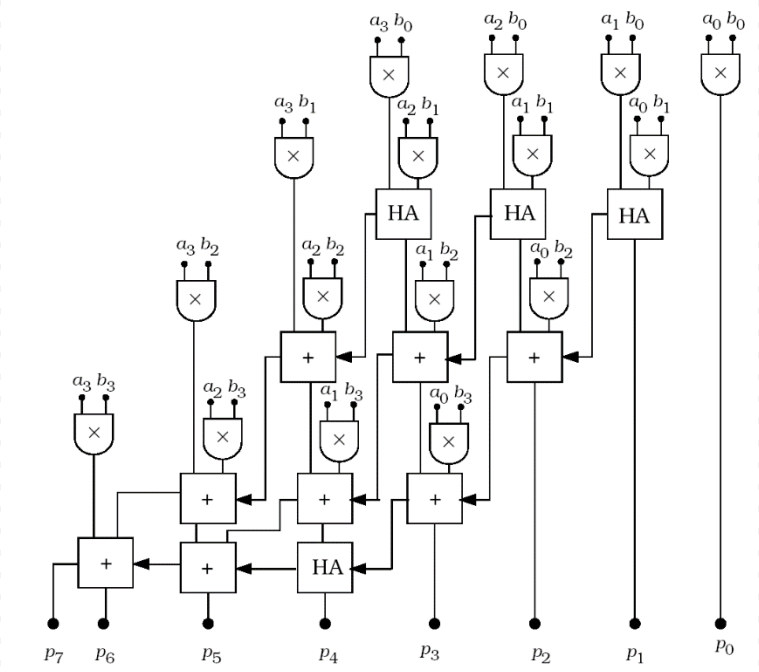
An array multiplier.

Multipliers (9)



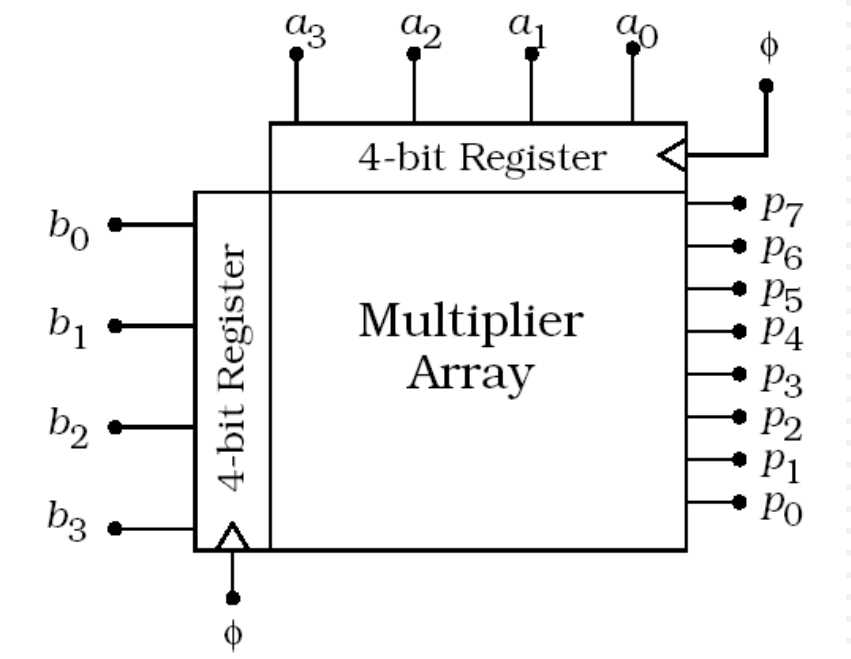
Modularized view of the multiplication sequence.

Multipliers (10)



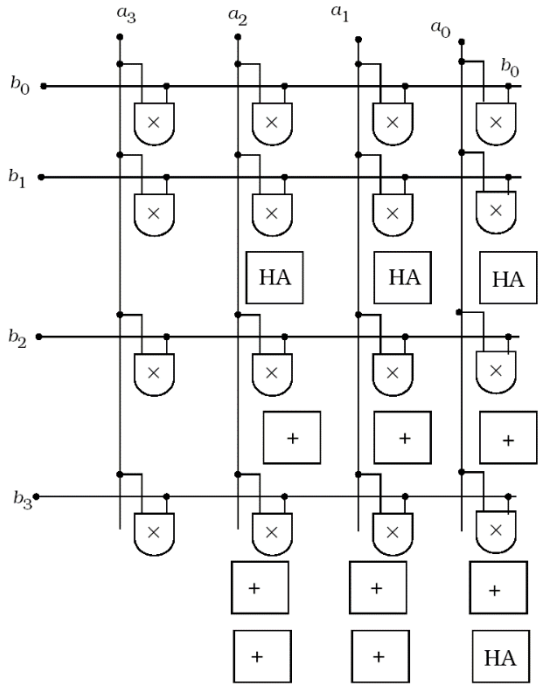
Details for a 4 x 4 array multiplier.

Multipliers (11)



Clocked input registers.

Multipliers (12)



Initial cell placement for the array.

Multipliers (13)

b_{2k+1}	b_{2k}	b_{2k-1}	E_k	Effect on sum
0	0	0	0	add 0
0	0	1	+ 1	add *A*
0	1	0	+ 1	add *A*
0	1	1	+ 2	shift A left, add
1	0	0	- 2	take two's (A), shift left, add
1	0	1	- 1	add two's (*A*)
1	1	0	- 1	add two's (*A*)
1	1	1	0	add 0

Summary of booth encoded digit operations.