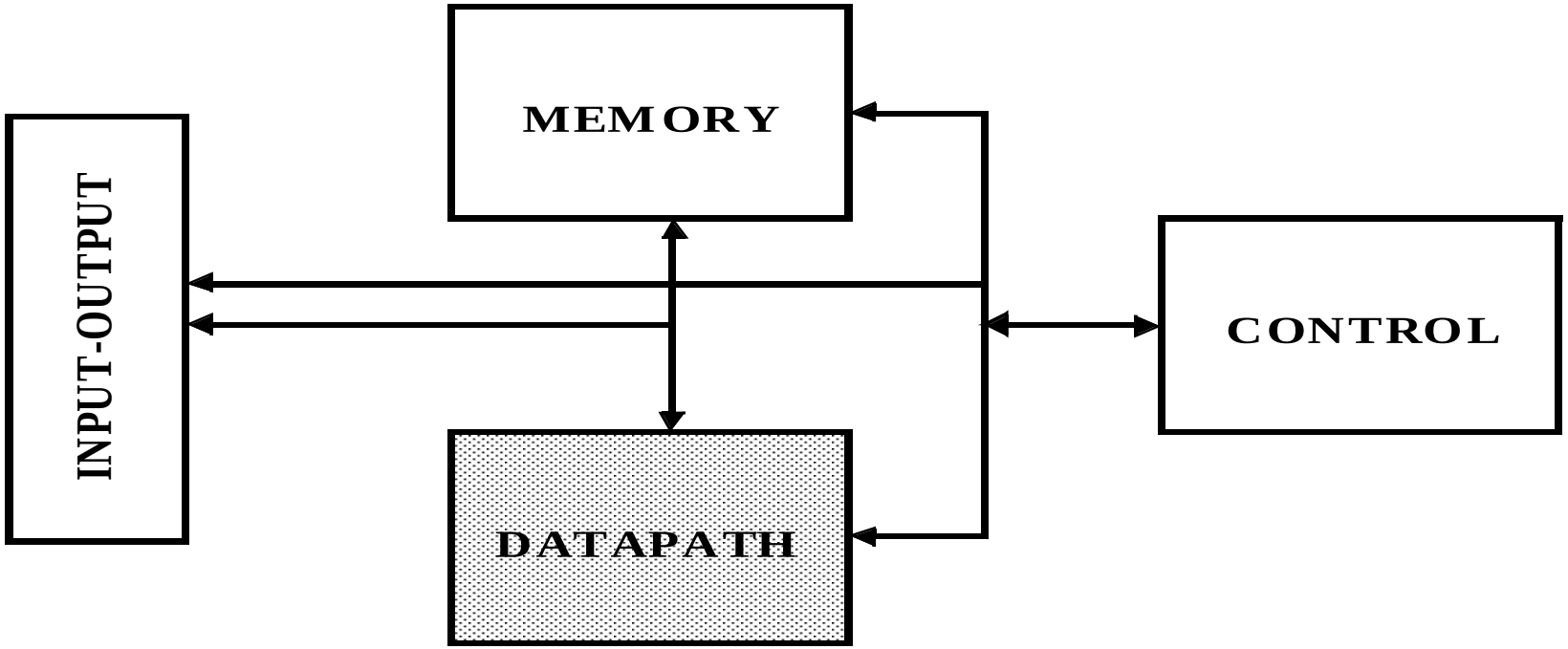
Data Path Design

Contents

- Adder
  - Bit adder circuits
  - Ripple Carry Adder
  - CLA Adder
- Multipliers and Shifter
  - Partial-product generation
  - Partial-product accumulation
  - Final addition
  - Barrel shifter

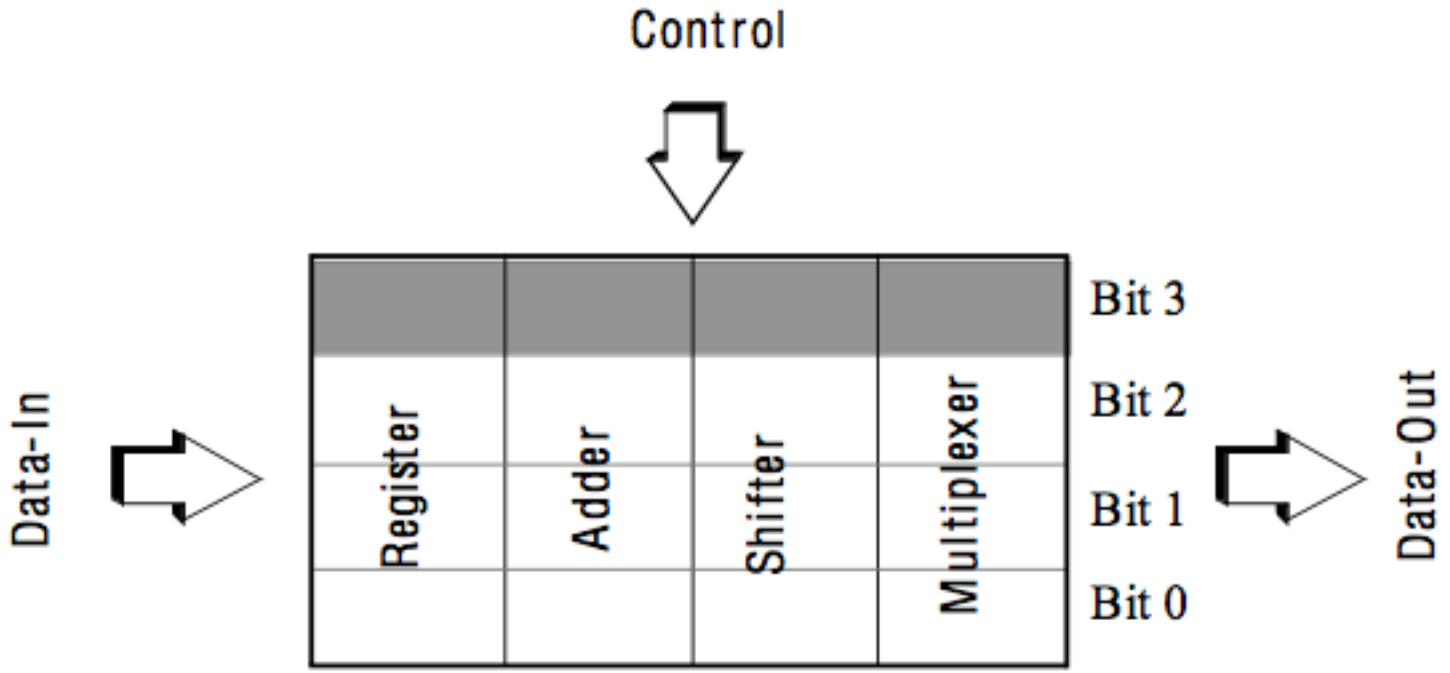
A Generic Digital Processor



Building Blocks for Digital Architectures

- Arithmetic unit
  - Bit sliced data path – adder, multiplier, shifter, comparator, etc.
- Memory
  - RAM, ROM, buffers, shift registers
- Control
  - Finite state machine (PLA, random logic)
  - Counters
- Interconnect
  - Switches, arbiters, bus

Bit-Sliced Design



Tile identical processing elements

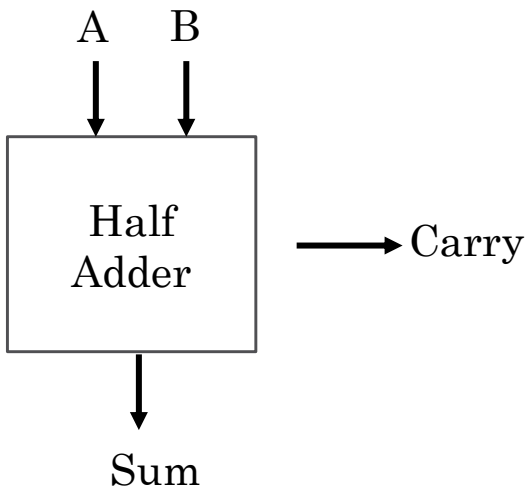
Bit Adder Circuits

- Consider two binary digits A an B, binary sum is denoted by A+B such that

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=10$$

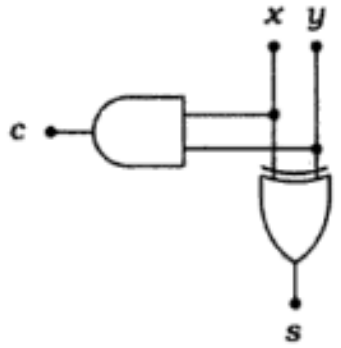
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half adder

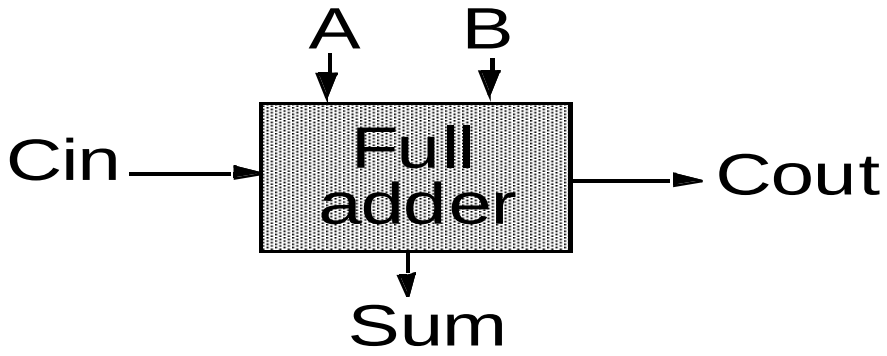


$$Sum = A \oplus B$$

$$Carry = A \cdot B$$



Full-Adder

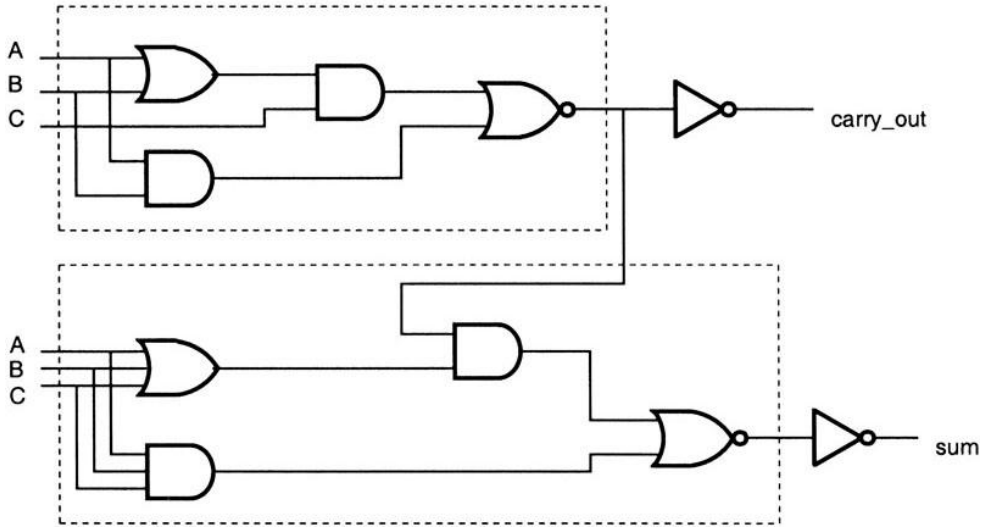


$$\text{sum\_out} = A \oplus B \oplus C$$

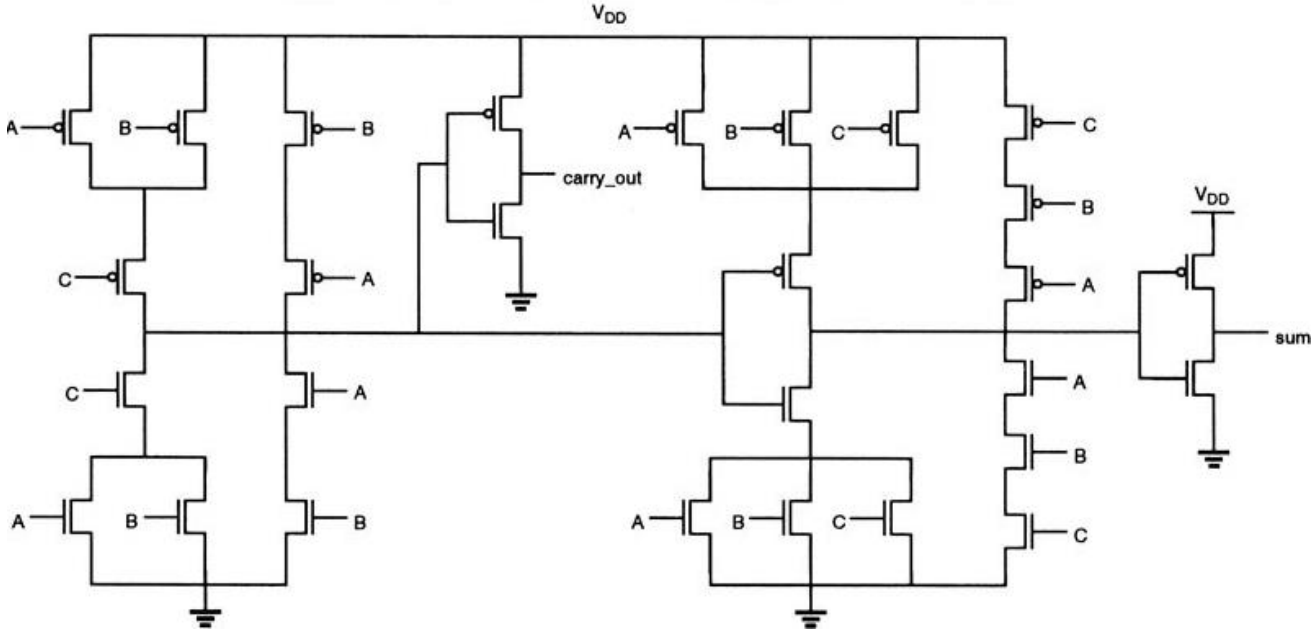
$$= ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{C}B$$

$$\text{carry\_out} = AB + AC + BC$$

AOI Full-Adder Logic



Transistor-level schematic of the one-bit full-adder circuit



Full-Adder Circuits

- Adding n-bit binary words

$$\begin{array}{rr} & a_3a_2a_1a_0 \\ + & b_3b_2b_1b_0 \\ \hline c_4 & s_3s_2s_1s_0 \end{array} \qquad (12.3)$$

- In the standard carry algorithm, each of the i-th columns ($i = 0, 1, 2, 3$) operates according to the full-adder equation

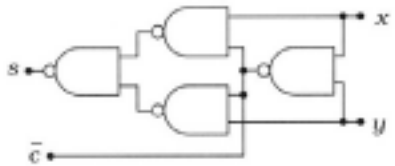
$$\begin{array}{rr} & c_i \\ & a_i \\ + & b_i \\ \hline c_{i+1} & s_i \end{array} \qquad (12.4)$$

- Expressions for the network are

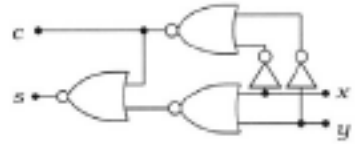
$$s_i = a_i \oplus b_i \oplus c_i$$
$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i) \qquad (12.5)$$

or

$$c_{i+1} = a_i b_i + c_i \cdot (a_i + b_i) \qquad (12.6)$$

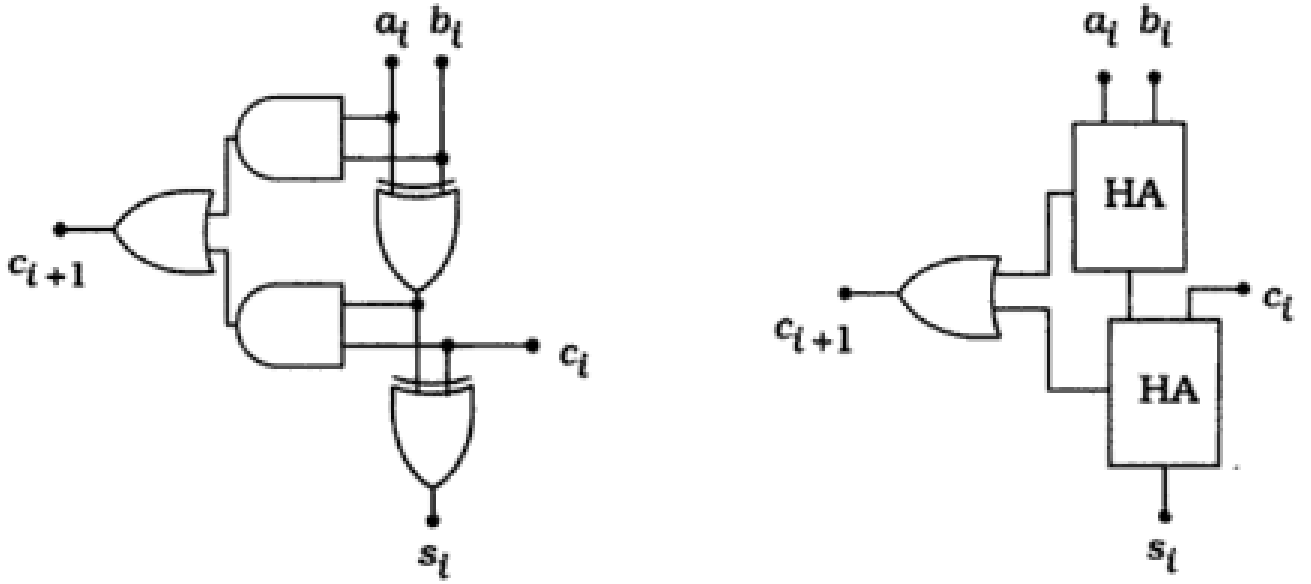


(a) NAND2 logic



(b) NOR-based network

Full-adder logic networks



(a) Gate-level logic

(b) HA-based design

Full-adder logic networks

Full-adder truth-table

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

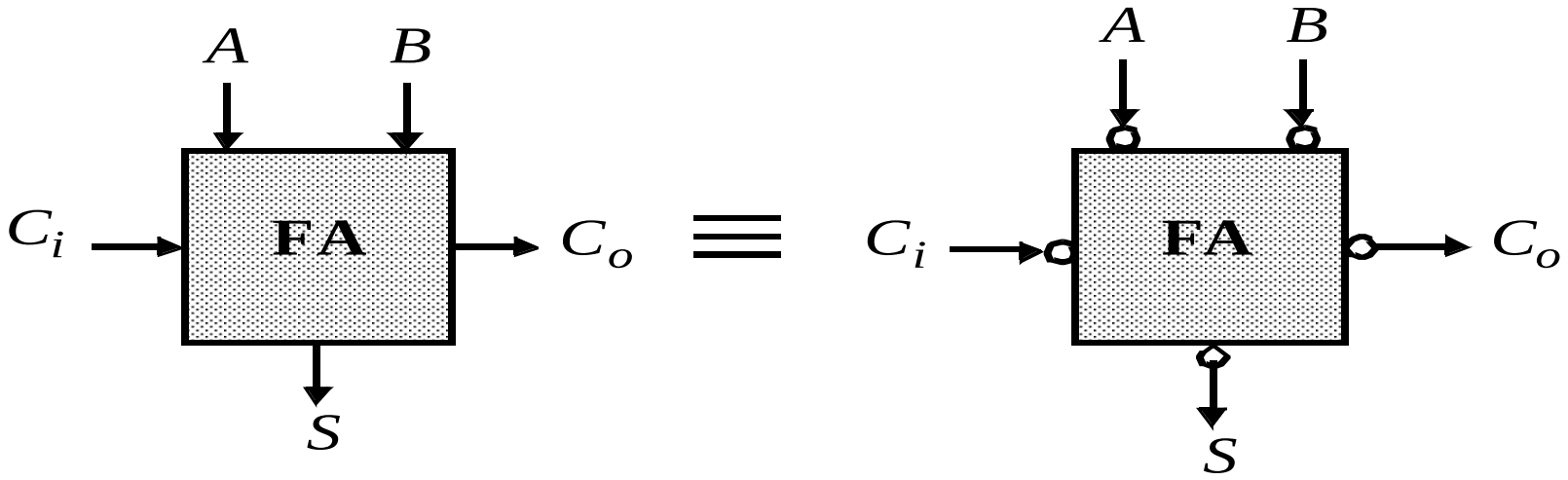
Generate (G) = AB

Propagate (P) = A ⊕ B

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

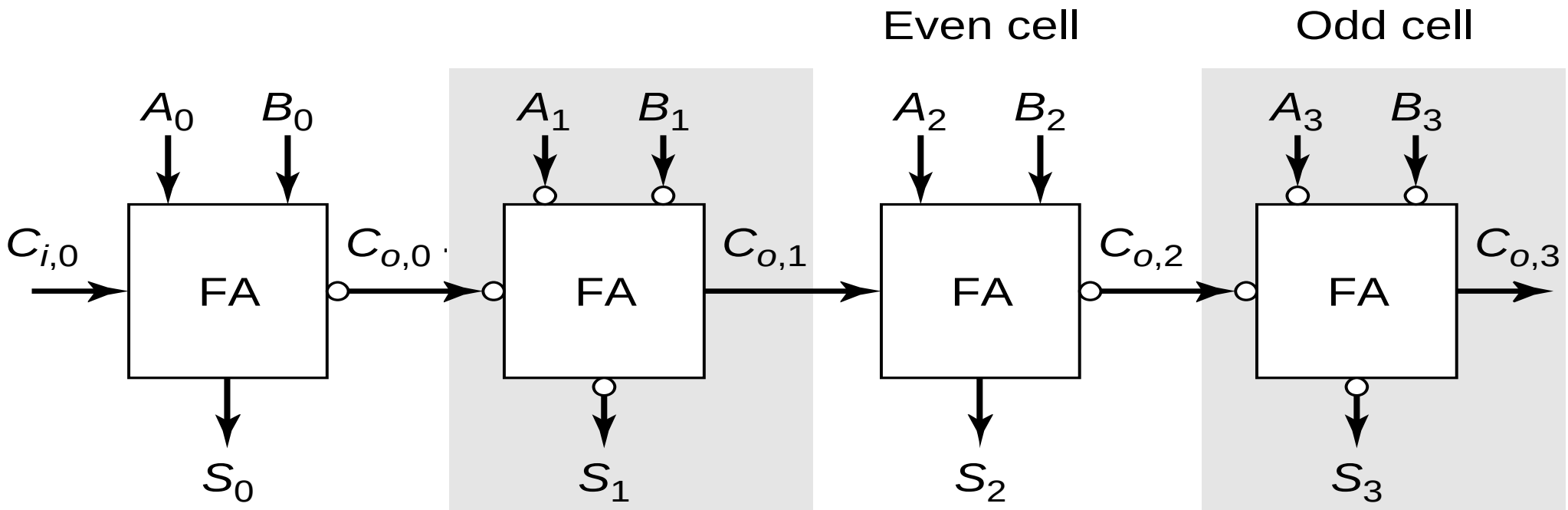
Inversion Property



$$\bar{S}(A, B, C_i) = S(\bar{A}, \bar{B}, \overline{C_i})$$

$$\overline{C_o}(A, B, C_i) = C_o(\bar{A}, \bar{B}, \overline{C_i})$$

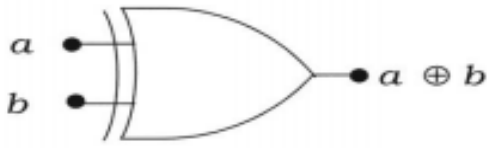
Minimize Critical Path by Reducing Inverting Stages



Exploit Inversion Property

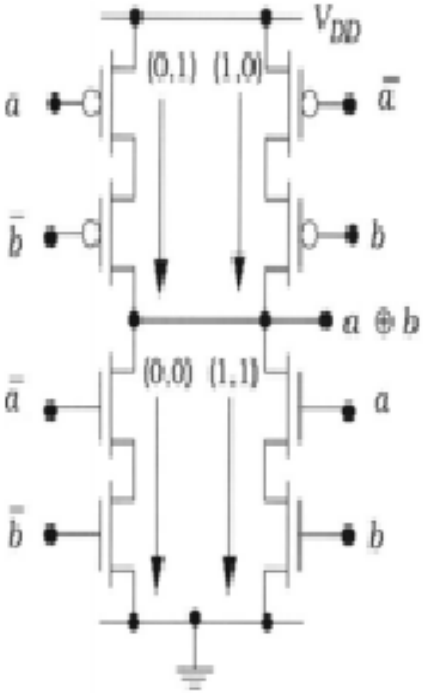
Mirror Circuits

- Mirror circuits are based on series-parallel logic gates, but are usually faster and have a more uniform layout
  - » Output 0's imply that an nFET chain is conducting to ground
  - » Output 1's means that a pFET group provides support from the power supply



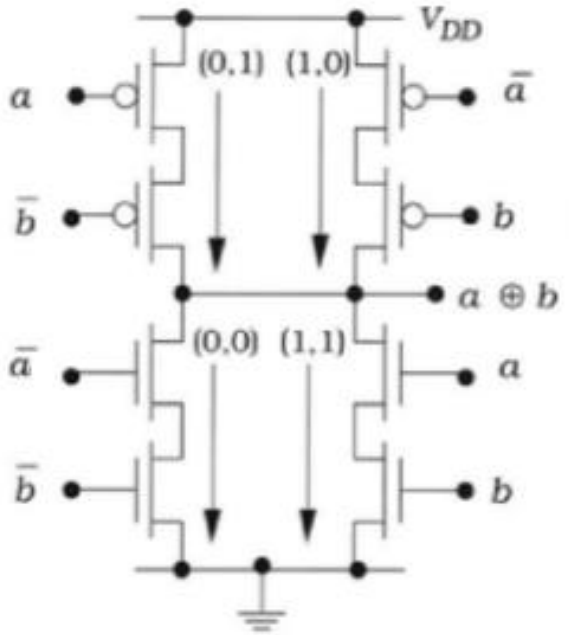
a	b	a ⊕ b	On devices
0	0	0	← nFET
0	1	1	← pFET
1	0	1	← pFET
1	1	0	← nFET

Figure 9.1 XOR function table

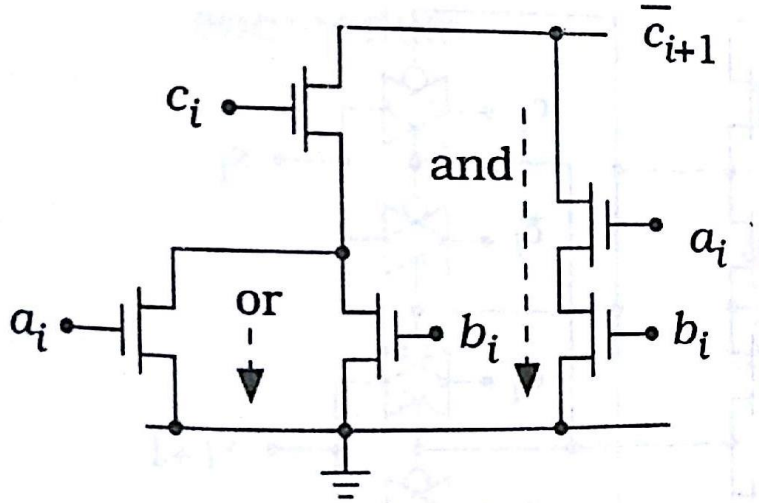


Mirror Circuits

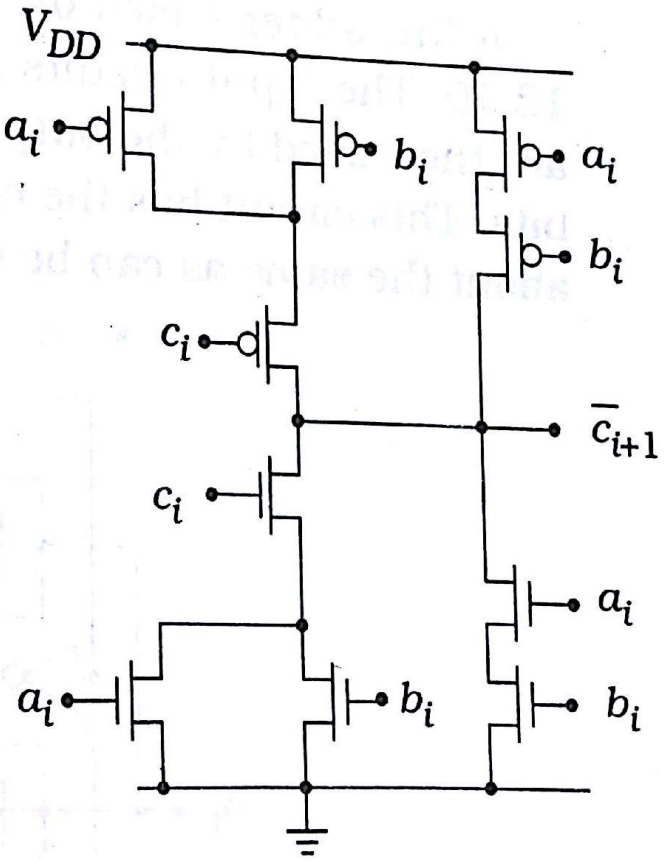
- Based on series-parallel logic gates
- Usually faster, more uniform layout
- Same transistor topology for nFETs and pFETs



Evolution of carry-out circuit

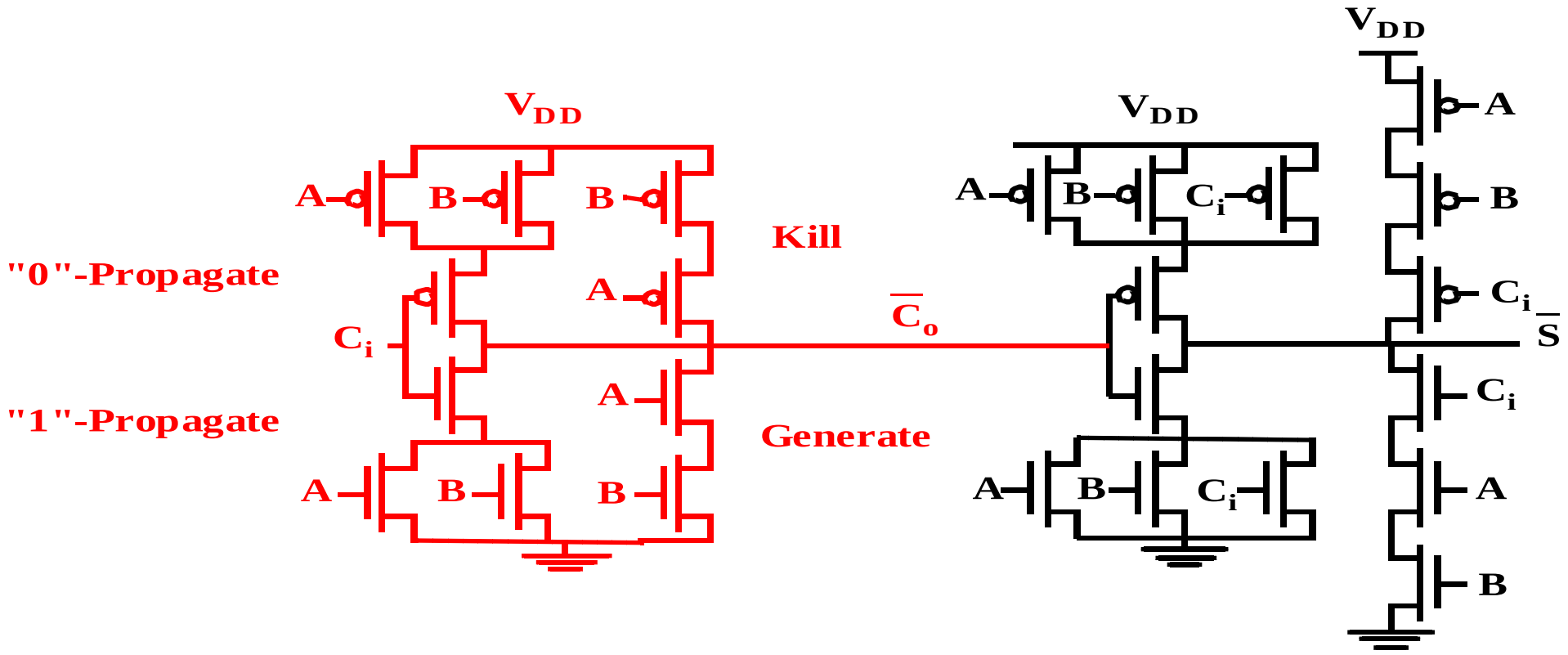


(a) Standard nFET logic



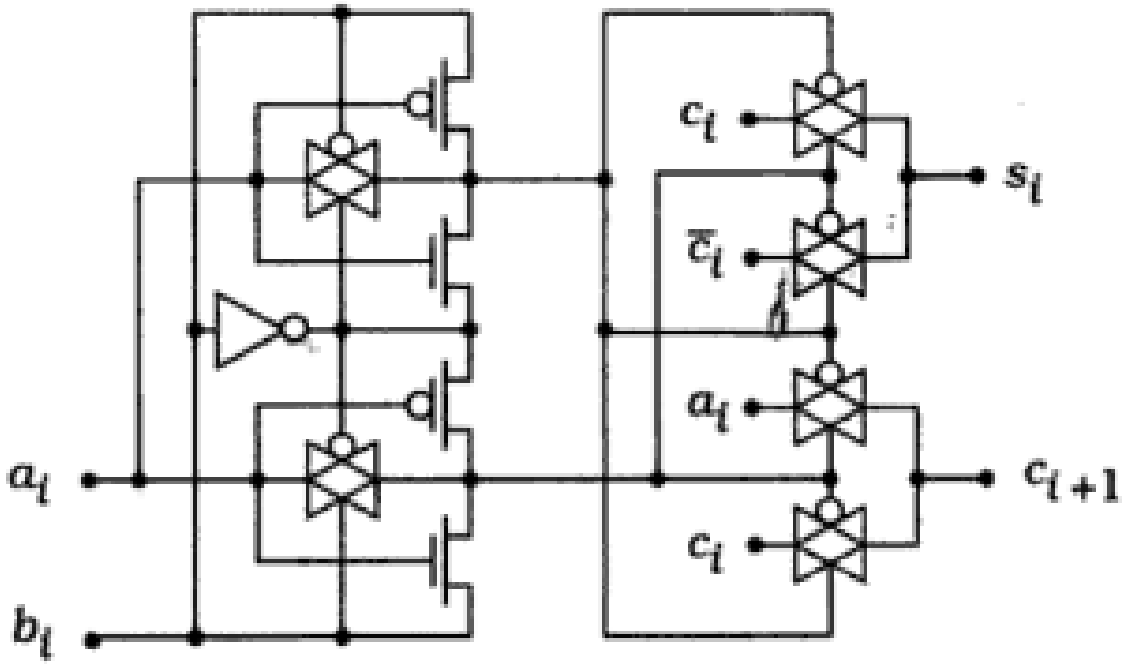
(b) Mirror circuit

A Better Structure: The Mirror Adder



24 transistors

Transmission-gate full-adder circuit



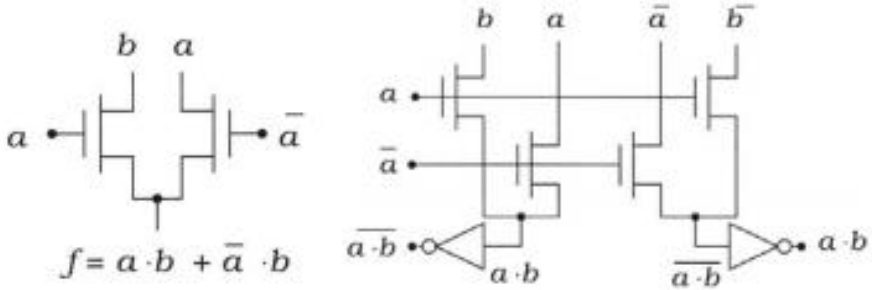
Complementary Pass-Transistor Logic

- Complementary Pass-Transistor (CPL): an dual-rail tech. that is based on nFET logic equations

$$f = a \cdot b + \bar{a} \cdot a \qquad (9.41)$$

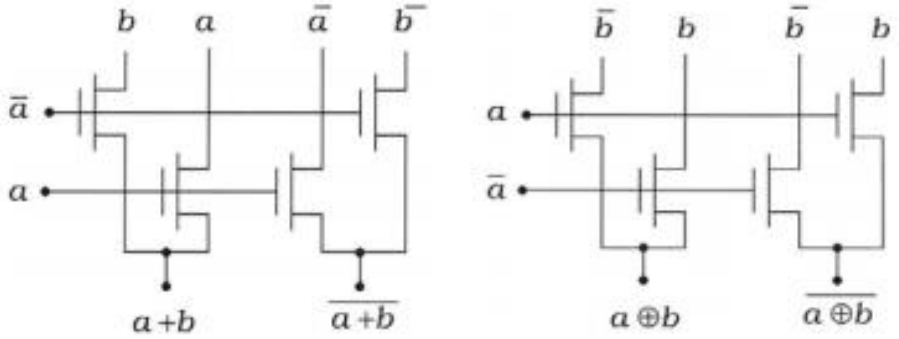
$$\Rightarrow a \cdot \bar{b} + \bar{a} = \bar{a} + \bar{b} = \overline{a \cdot b} \qquad (9.42)$$

- CPL has several 2-input gates that can be created by using the same transistor topology with different input sequences
  - » Less layout area
  - » However, threshold will be loss and the fact that an input variable may have to drive more than one FET terminal



(a) AND gate (b) AND/NAND array

Figure 9.32 CPL AND/NAND circuit



(a) OR/NOR (b) XOR/XNOR

Figure 9.33 2-input CPL arrays

Complementary Pass-Transistor Logic

- Dual-rail complementary pass-transistor logic (CPL)

$a_i \oplus b_i$ *and* $\overline{a_i \oplus b_i}$

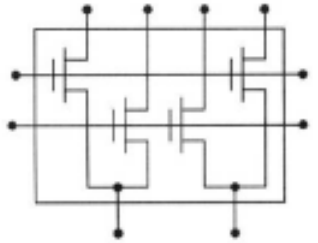
$s_n = \overline{(a_i \oplus b_i)} \cdot c_i + (a_i \oplus b_i) \cdot \overline{c_i}$

$\overline{a_i} \cdot b_i + \overline{b_i} \cdot \overline{c_i}$

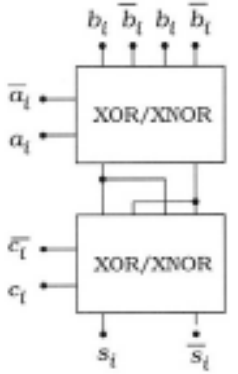
$\overline{b_i} \cdot c_i + a_i \cdot b_i$

$\overline{a_i} \cdot \overline{b_i} + b_i \cdot \overline{c_i}$

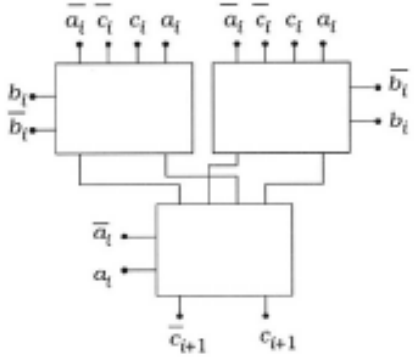
$b_i \cdot c_i + a_i \cdot \overline{b_i}$



(a) 2-input array

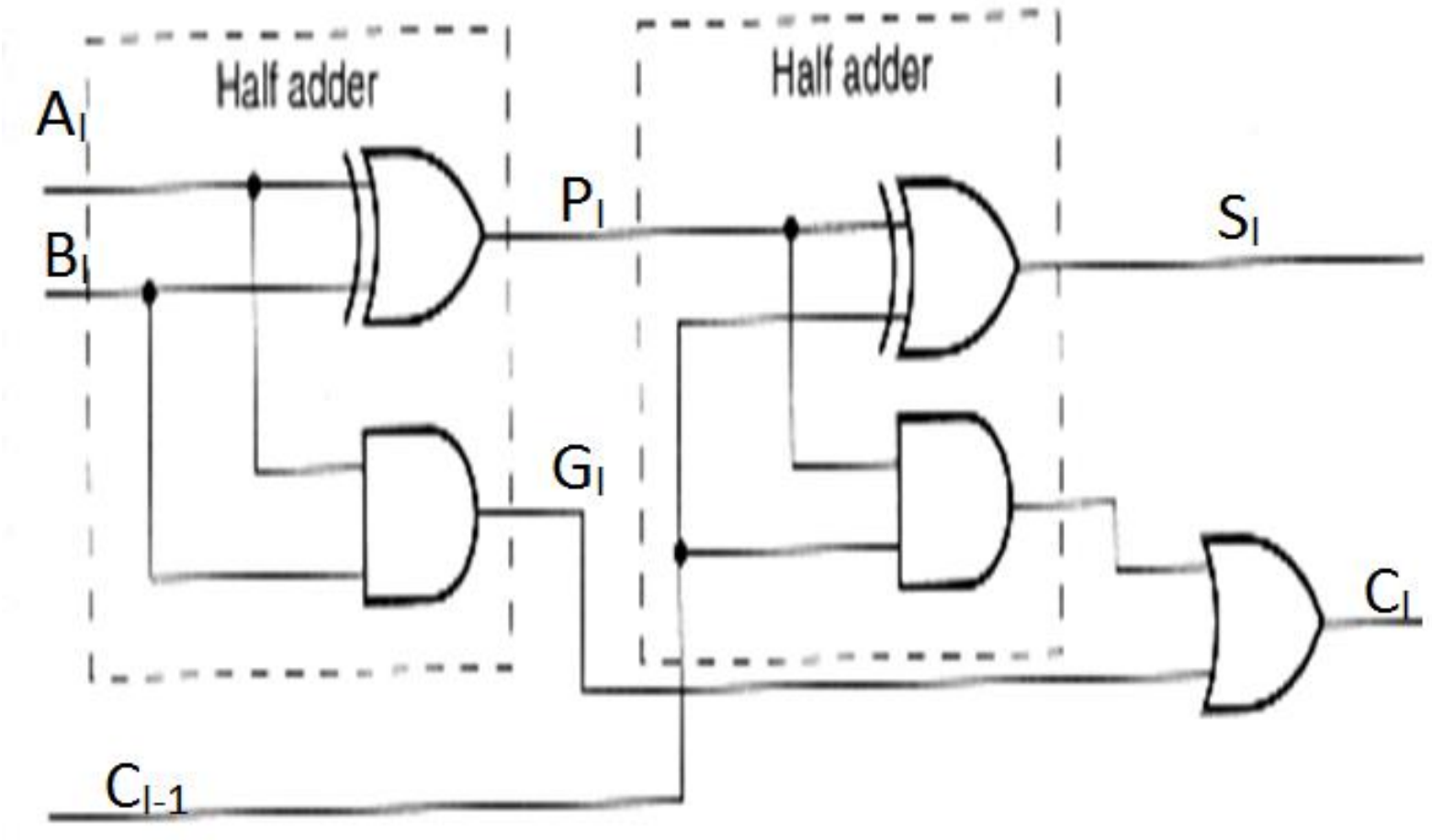


(b) Sum circuit



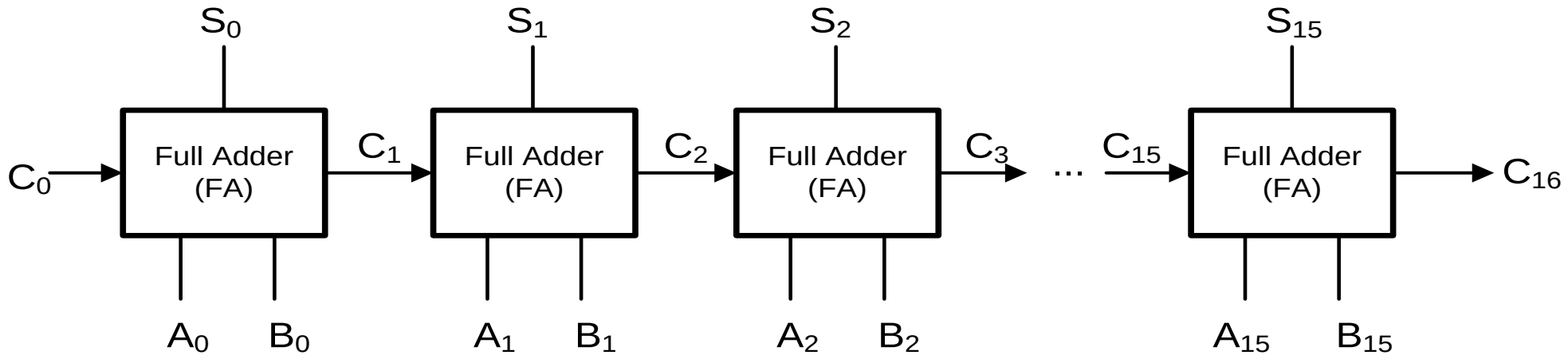
(c) Carry circuit

Full Adder using Half Adders



Ripple Carry Adders

- A carry ripple adder chain : 16-bit binary adder



- Cascade-connection
- Fast carry-out response is essential
- the delay will increase significantly when the number of bit is increased

Ripple-Carry Adders

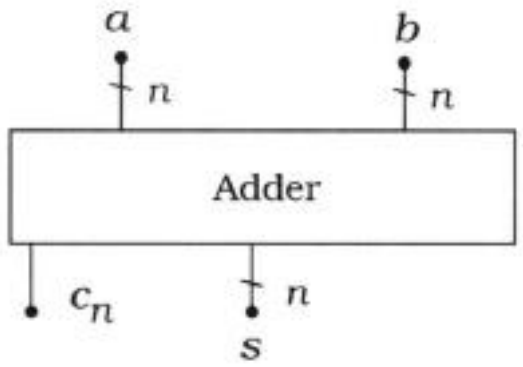


Figure 12.11 An n-bit adder

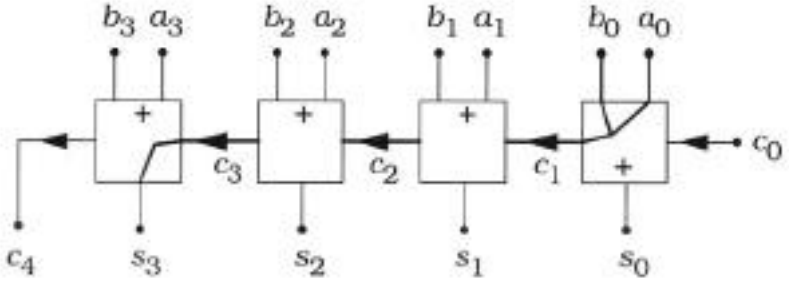


Figure 12.13 Worst-case delay through the 4-bit ripple adder

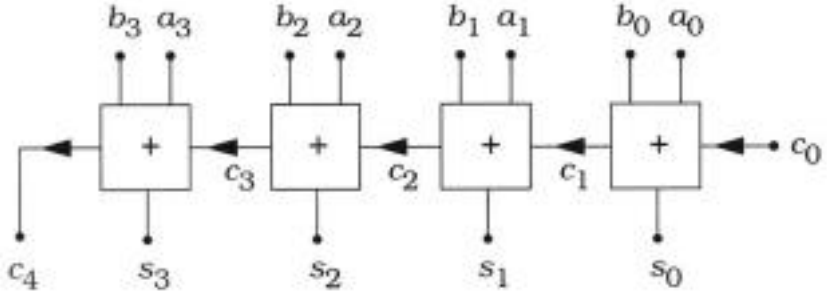


Figure 12.12 A 4-bit ripple-carry adder

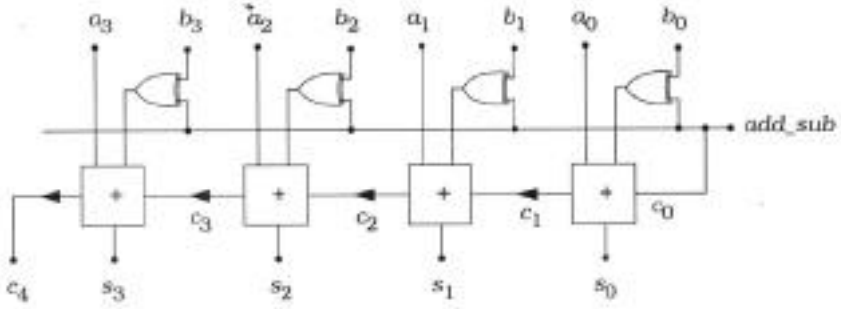
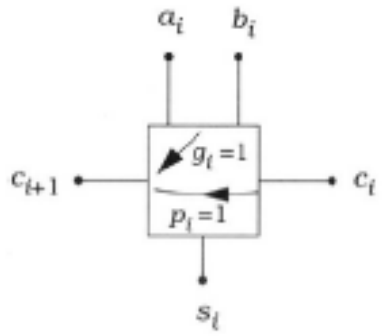


Figure 12.14 4-ibt adder-subtractor circuit

Carry Look-Ahead Adder



	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

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

Figure 12.15 Basis of the carry look-ahead algorithm

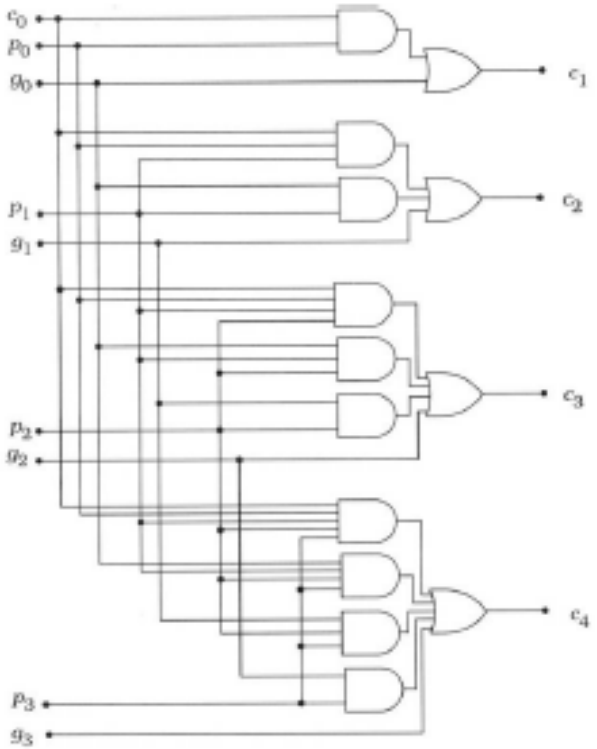


Figure 12.16 Logic network for 4-bit CLA carry bits

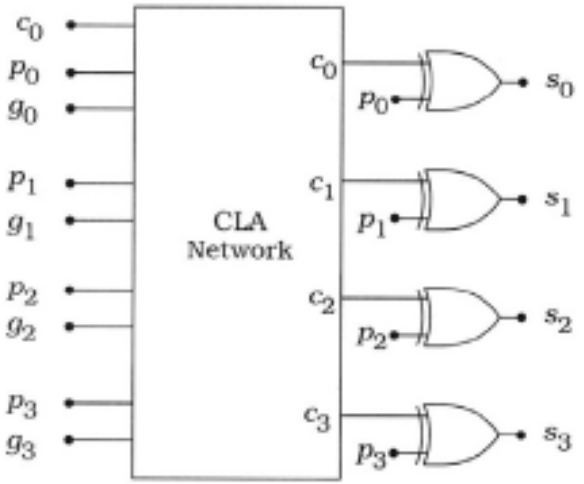
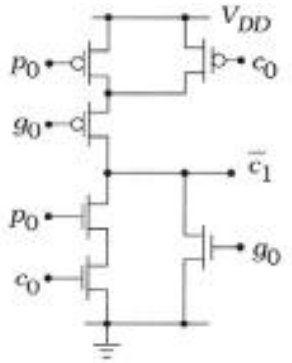
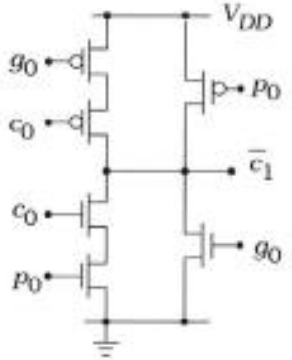


Figure 12.17 Sum calculation using the CLA network

Carry Look-Ahead Adders



(a) Series-parallel circuit



(b) Mirror equivalent

Figure 12.20 Static CLA mirror circuit

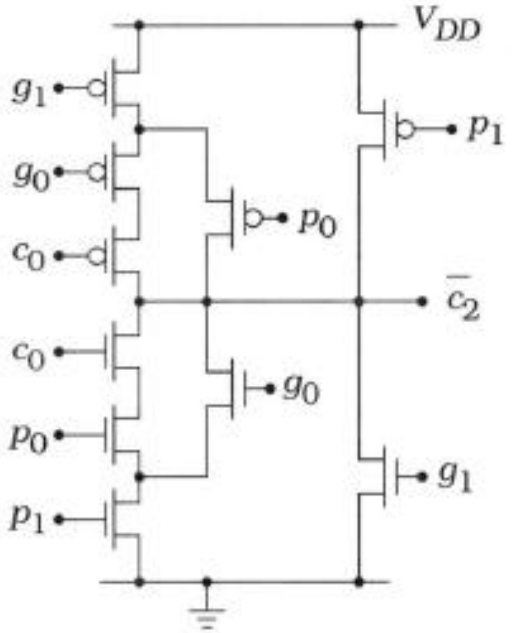


Figure 12.21 Static mirror circuit for C_2

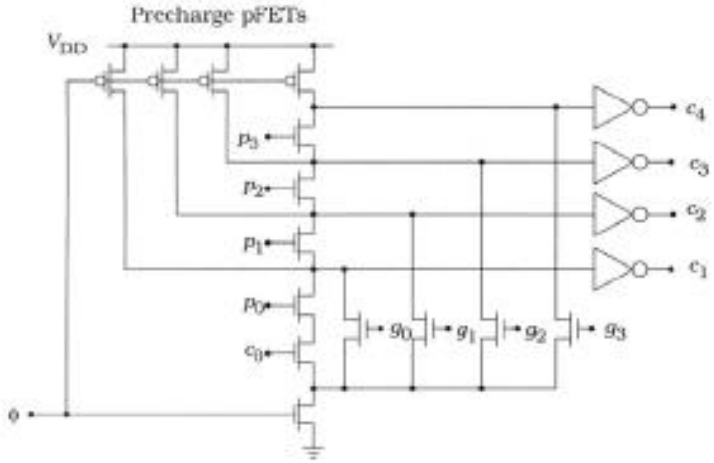
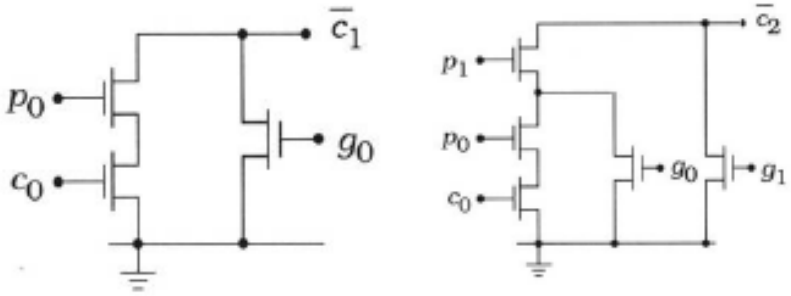


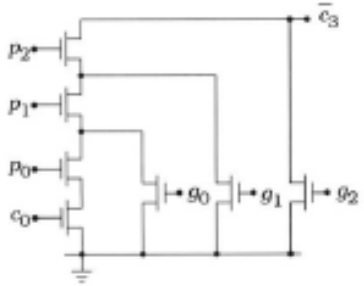
Figure 12.22 MODL carry circuit

Carry Look-Ahead Adders

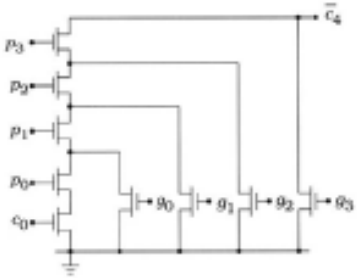


(a) C_1 logic

(b) C_2 logic

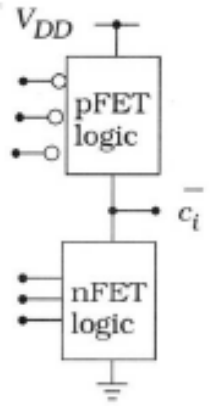


(c) C_3 logic

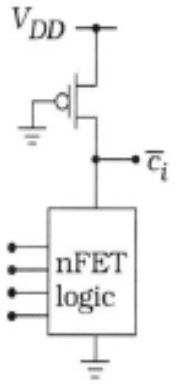


(d) C_4 logic

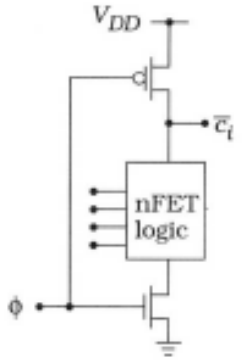
Figure 12.18 nFET logic arrays for the CLA terms



(a) Complementary



(b) Pseudo nMOS



(c) Dynamic

Figure 12.19 Possible uses of the nFET logic arrays in Figure 12.18

Manchester Carry Chains

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

Figure 12.23 Propagate, generate, and carry-kill values

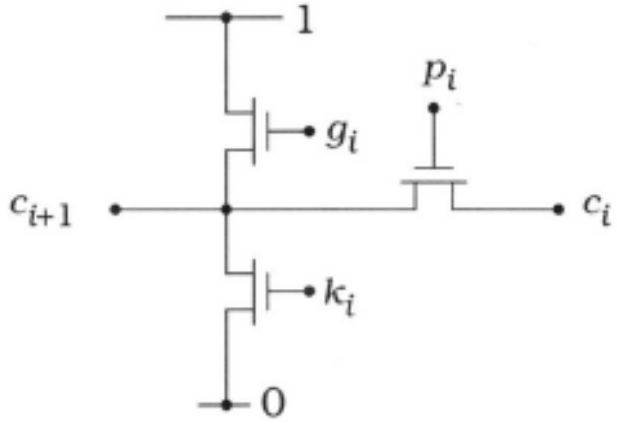
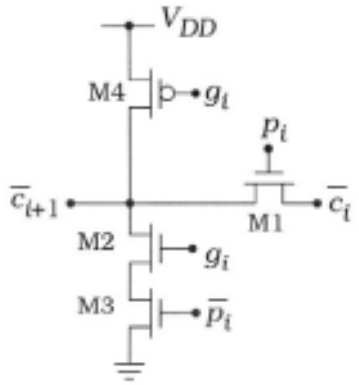
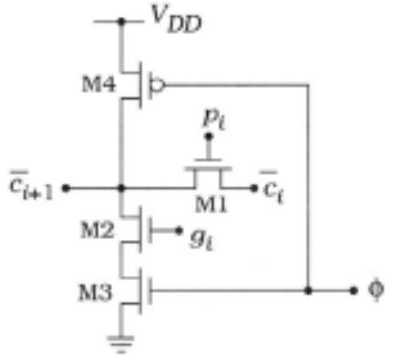


Figure 12.24 Switching network for the carry-out equation



(a) Static circuit



(b) Dynamic circuit

Figure 12.25 Manchester circuit styles

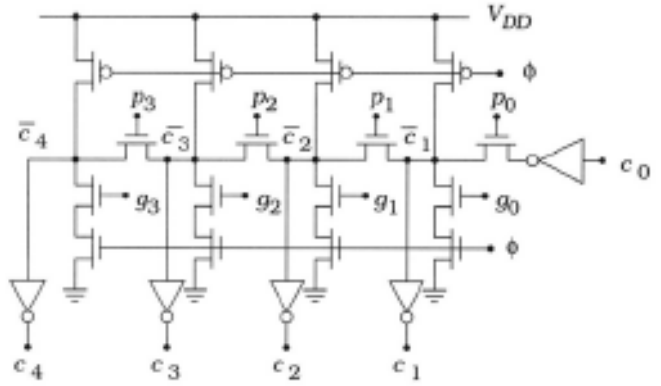


Figure 12.26 Dynamic Manchester carry chain

Multipliers

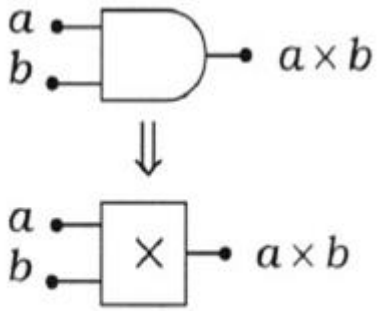


Figure 12.39 Bit-level multiplier

			a_3	a_2	a_1	a_0		multiplicand
			× b_3	b_2	b_1	b_0		multiplier
			$a_3 b_0$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$		
		+ $a_3 b_1$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$			
	+ $a_3 b_2$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$				
+ $a_3 b_3$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$					
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product

Figure 12.40 Multiplication of two 4-bit words

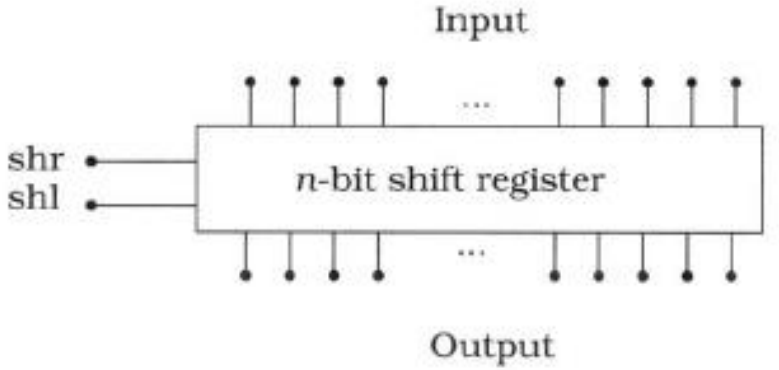
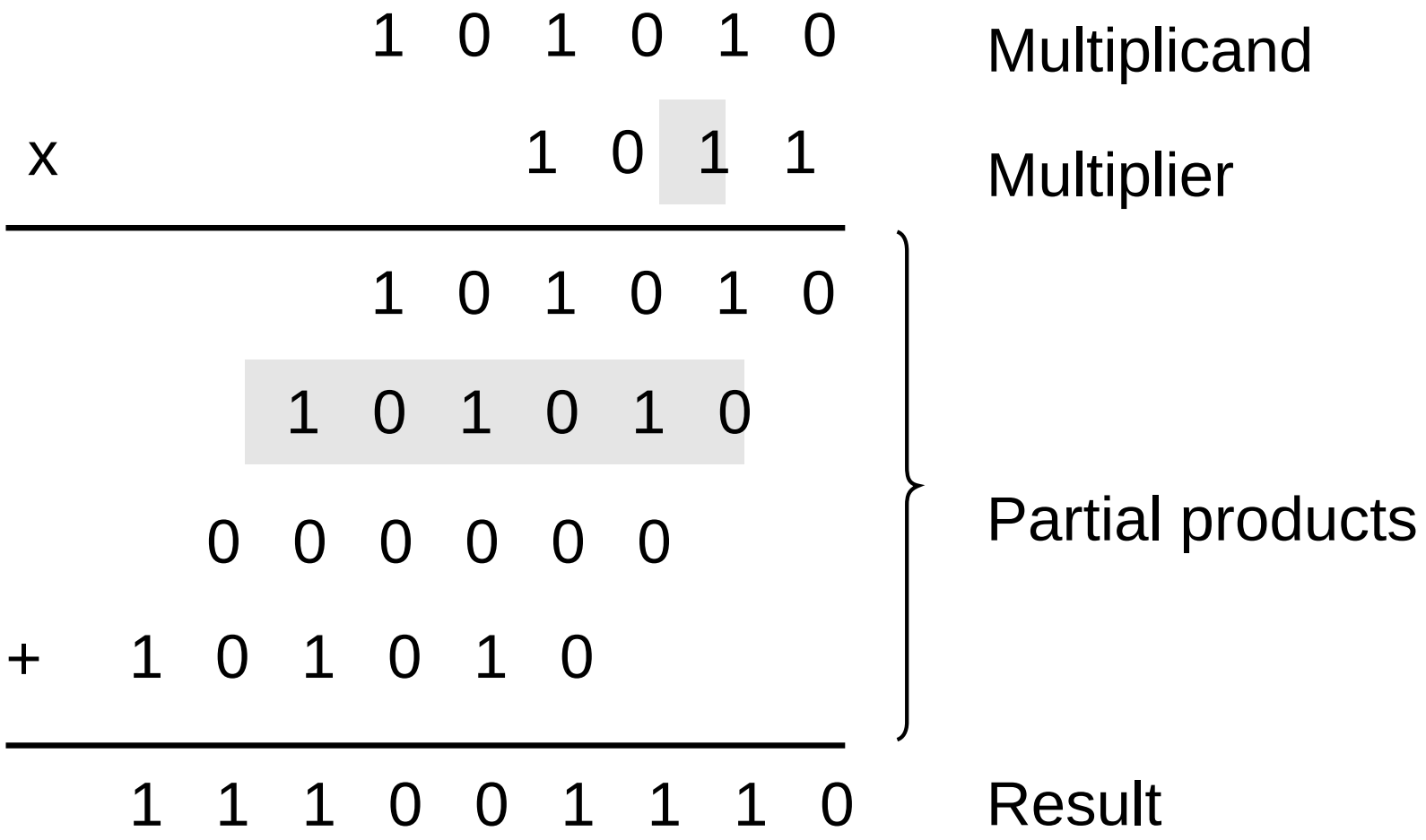


Figure 12.41 Shift register for multiplication or division by a factor of 2

Binary Multiplication



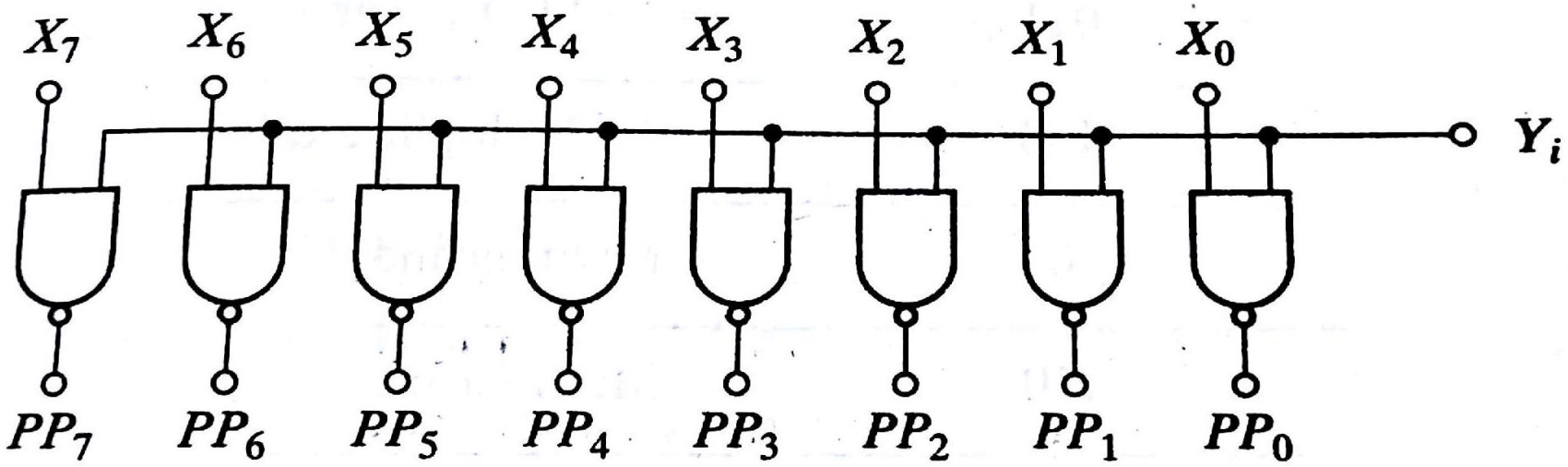
Multipliers

		a_3	a_2	a_1	a_0	multiplicand
	$\times$	b_3	b_2	b_1	b_0	multiplier
		$(a_3$	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	$a_0) \times b_3$			$(a \times b_3)\,2^3$
p_7	p_6	p_5	p_4	p_3	p_2	p_1 p_0	product

Figure 12.42 Alternate view of multiplication process

Figure 12.43 Using a product register for multiplication

Partial product generation logic



Shift-right multiplication sequence

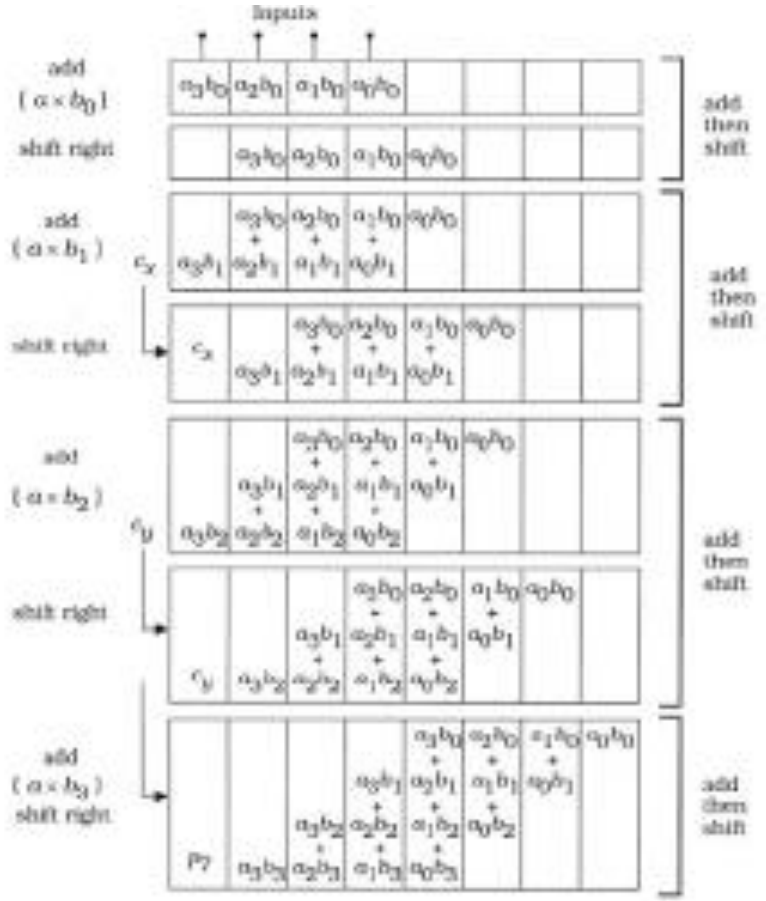


Figure 12.44 Shift-right multiplication sequence

Register based multiplier network

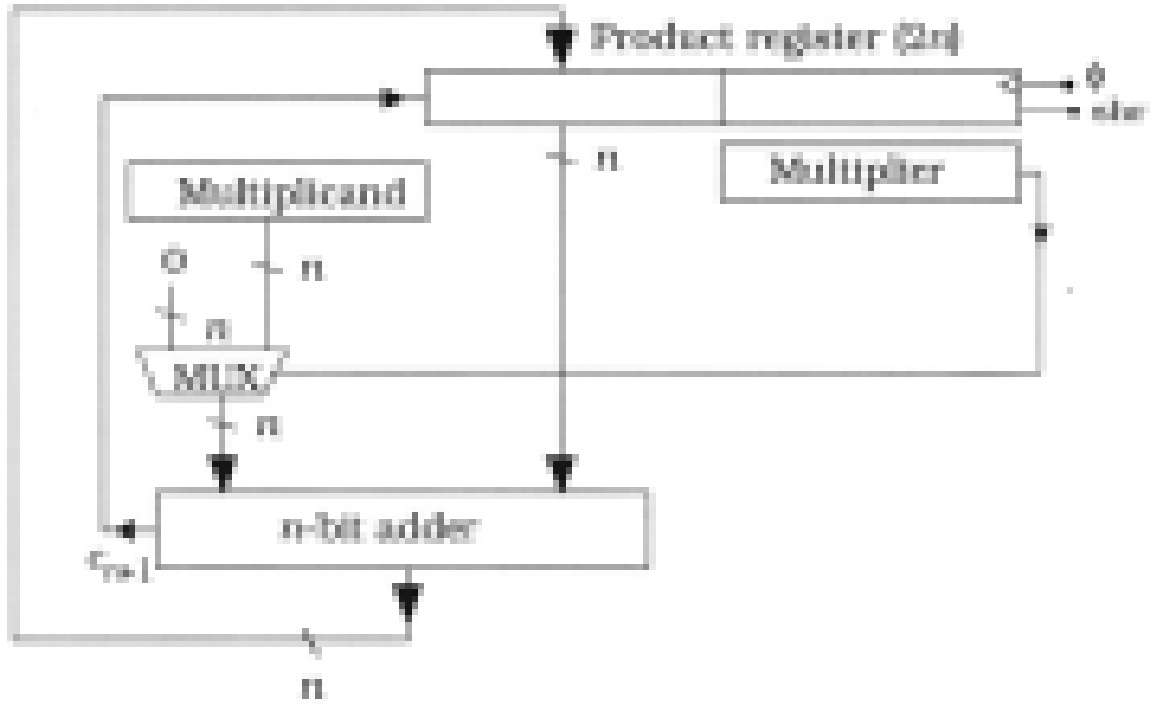


Figure 12.45 Register-based multiplier network

Partial Product Accumulation- Array Multipliers

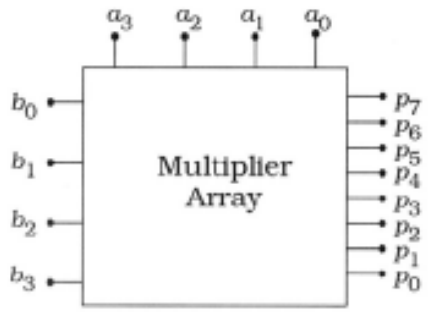


Figure 12.46 An array multiplier

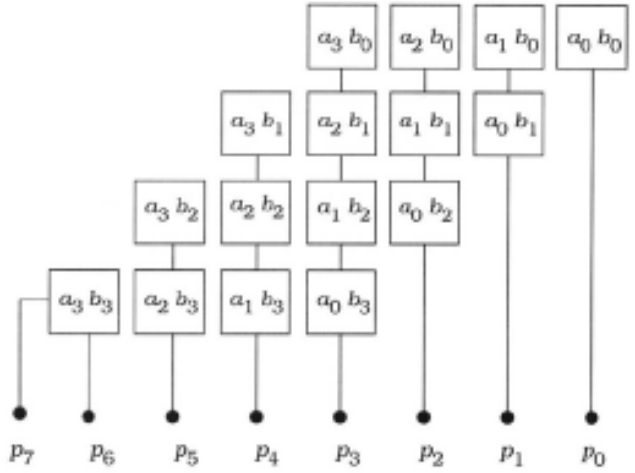


Figure 12.47 Modularized view of the multiplication sequence

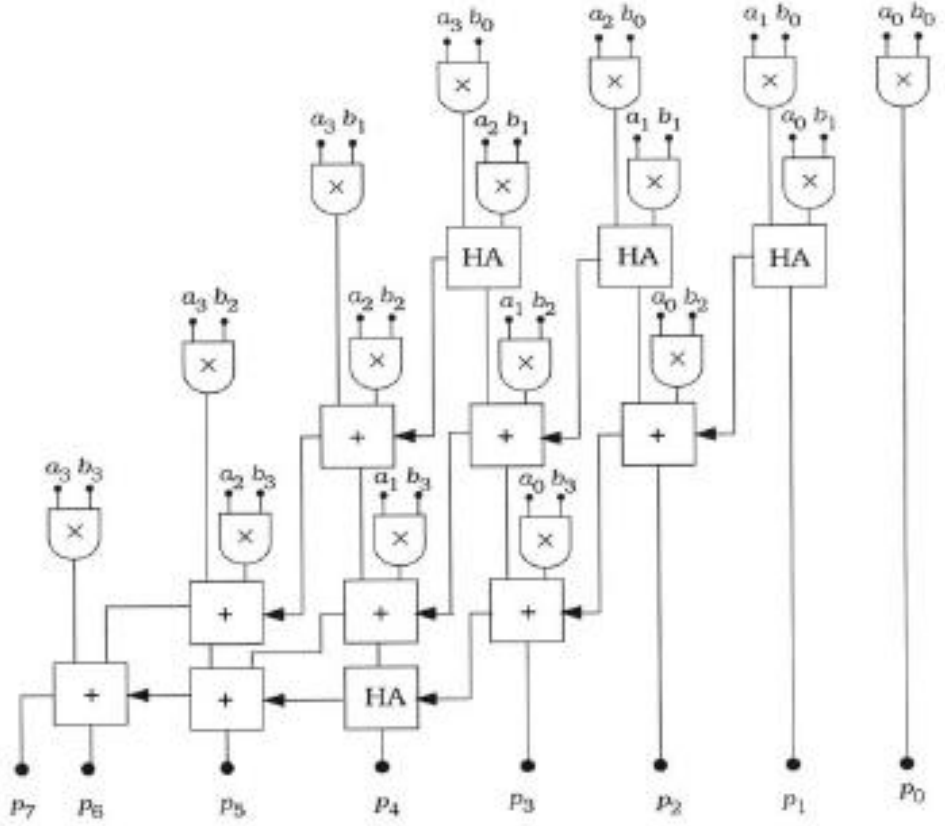
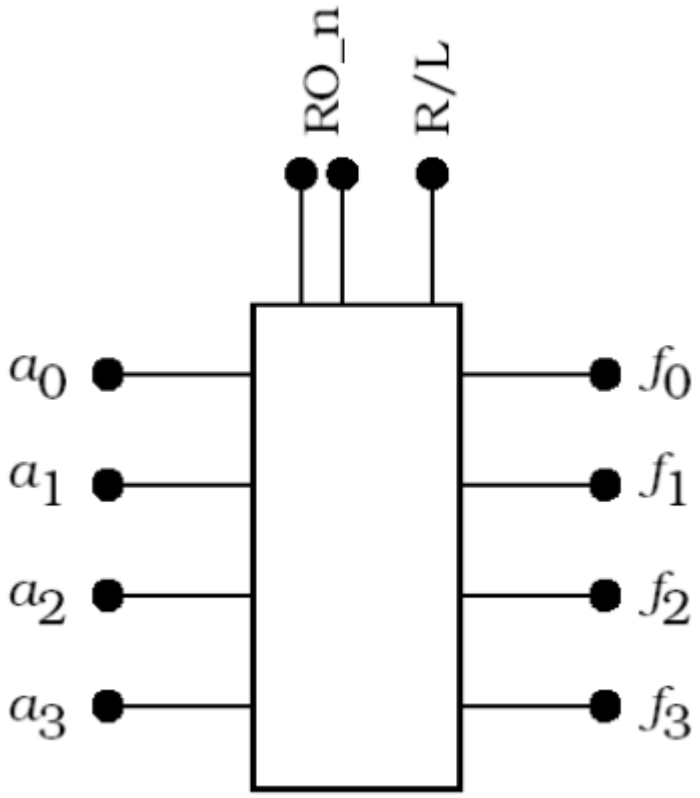
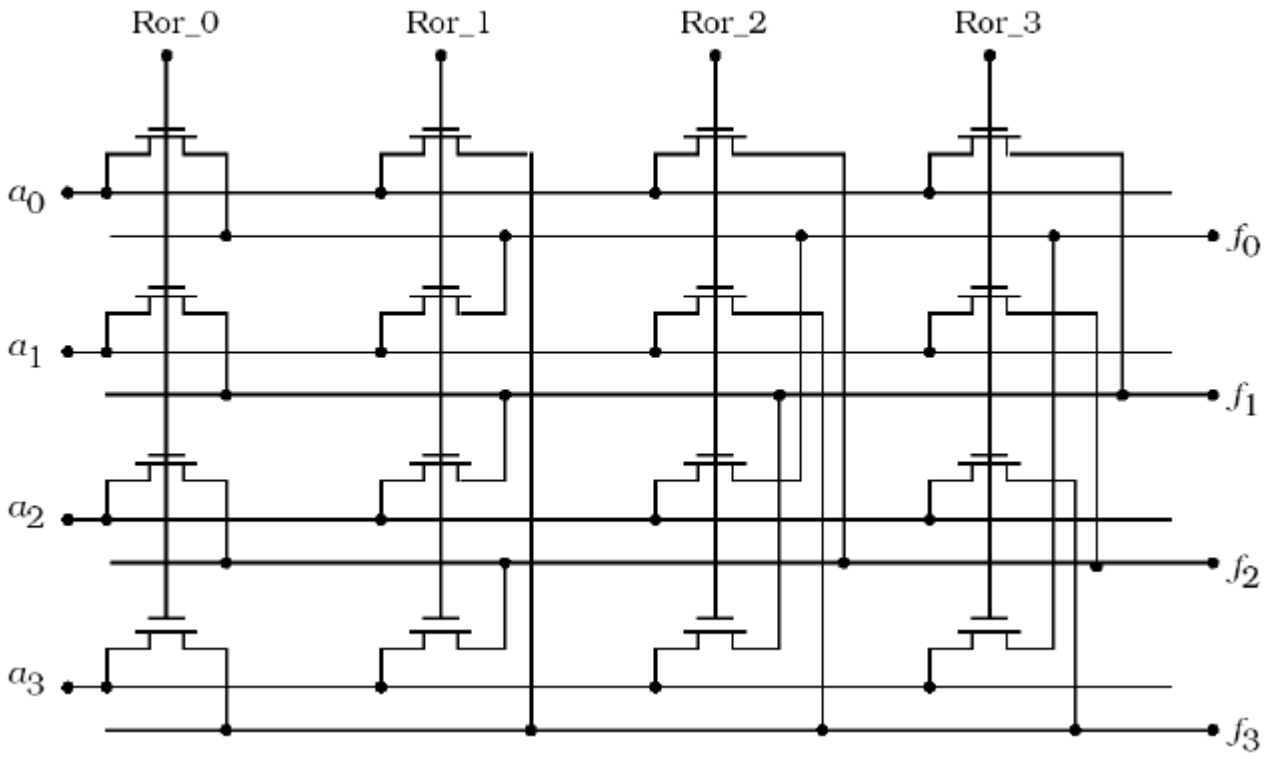


Figure 12.48 Details for a 4 X 4 array multiplier

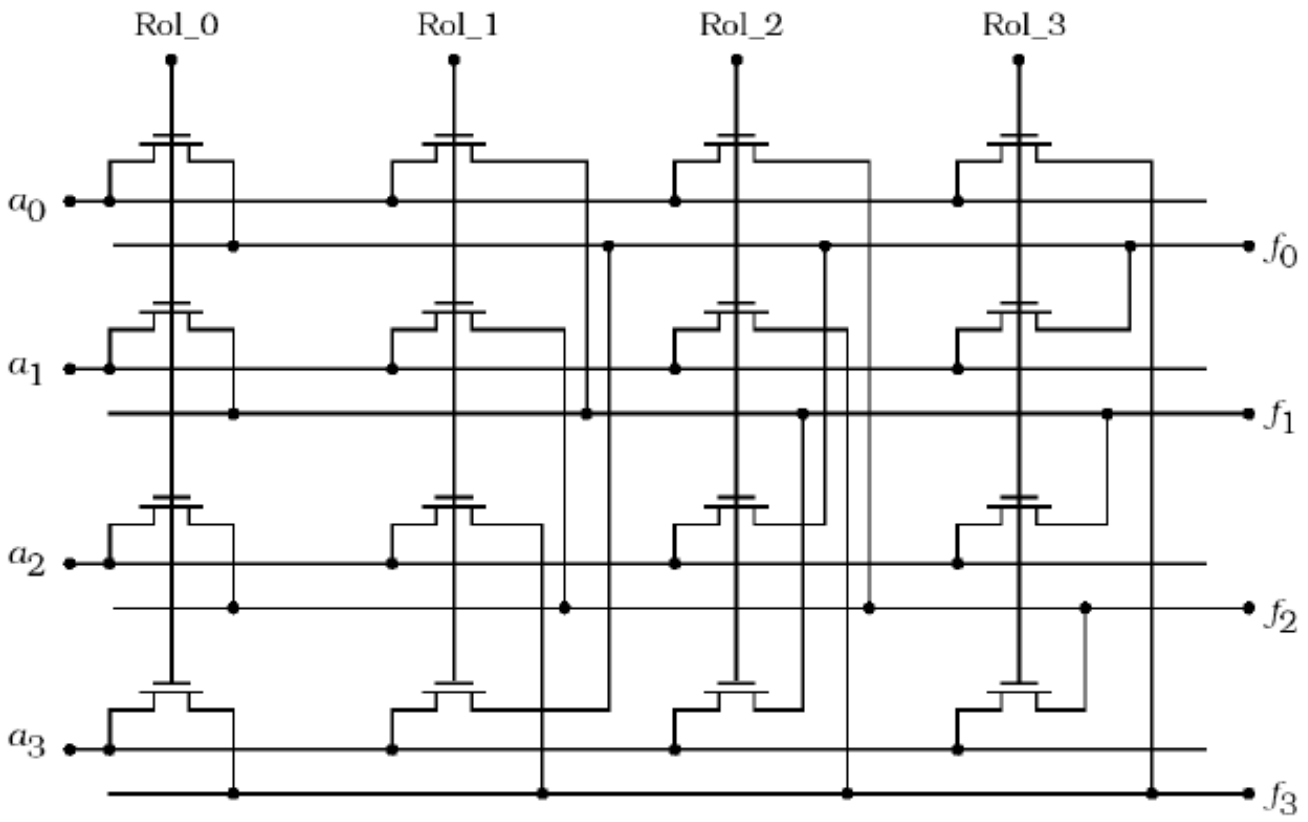
General Rotator



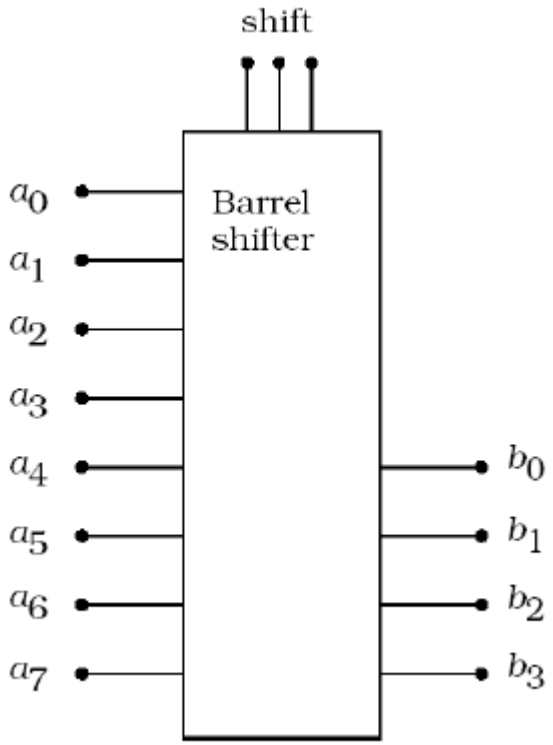
4-bit rotate-right network



Left-rotate switching array

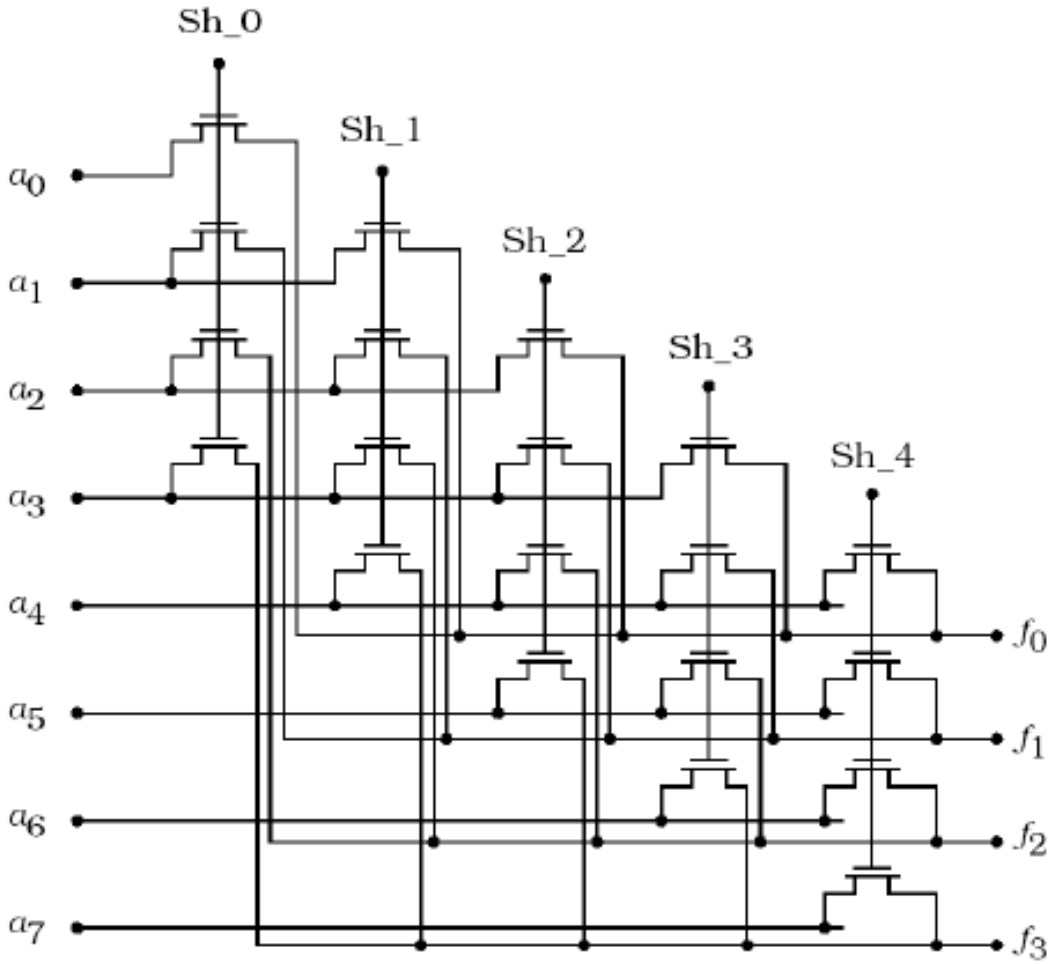


8 x 4 Barrel Shifter



shift	$b_0b_1b_2b_3$
0	$a_0a_1a_2a_3$
1	$a_1a_2a_3a_4$
2	$a_2a_3a_4a_5$
3	$a_3a_4a_5a_6$
4	$a_4a_5a_6a_7$

FET Array Barrel Shifter



Thank You