VLSI Digital Design

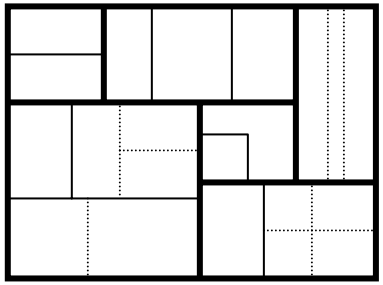
MODULE IV
ARITHMETIC AND LOGIC SUBSYSTEM DESIGN

4.1 Algorithmic systems and structured design
4.2 Datapath operators: adders
4.3 Datapath operators: multipliers
4.4 Other operators

4.1 Algorithmic systems and structured design

4.1.1 Structured design principles

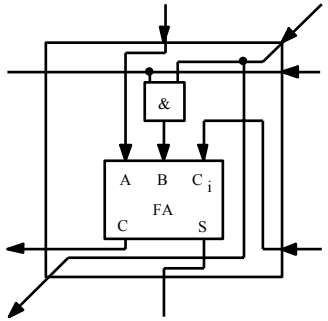
- Hierarchy (subdivision)



- Regularity (similar sub modules). Permits simple scaling

- Modularity (clearly defined function and interface)

- Locality (information encapsulated into modules)



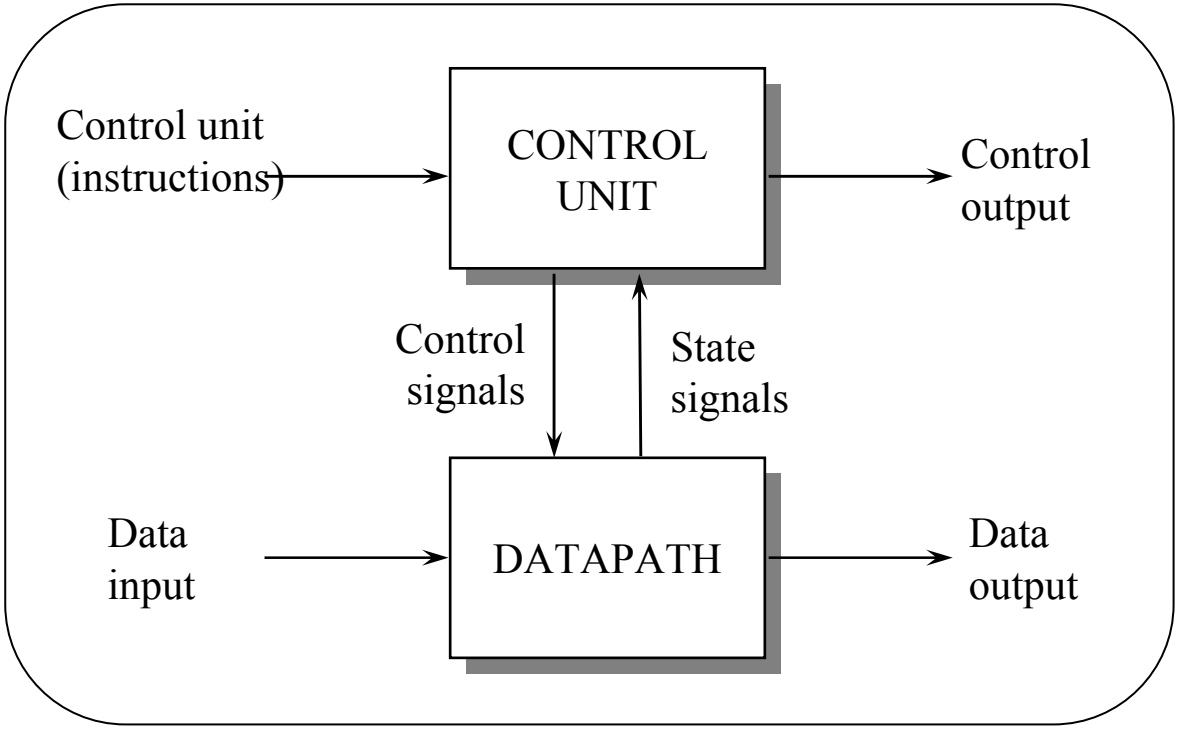
4.1.2 Algorithmic system review

Digital system **behavioral** descriptions (HDL, ASM,...) specify:

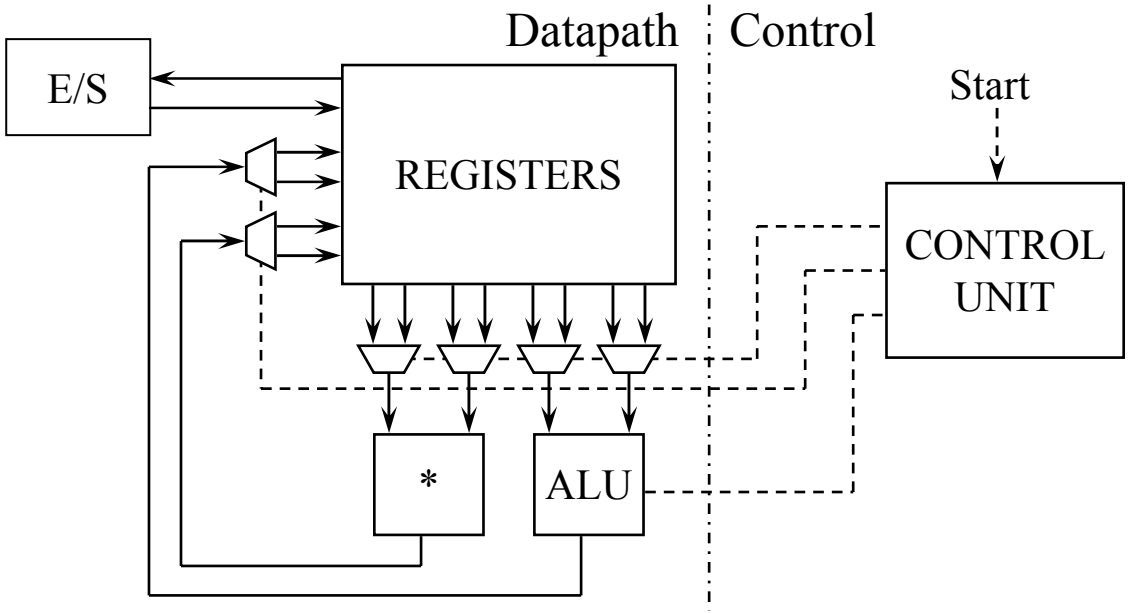
- Operations ➔ data process ➔ datapath
- Sequence ➔ process control ➔ control unit

Communication within units:

- Control signals
  (Control Unit ➔ Datapath)
- State signals
  (Datapath ➔ Control Unit)



Algorithmic system example:



D*atapath* consists of:

- Functional elements
- Registers (memory)
- Busses and connection elements

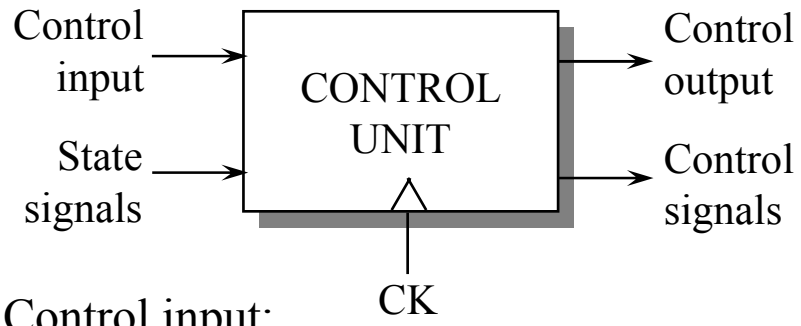
Input / Output resources are algorithmic subsystems

4.1.3 Control unit

The control unit generates the required signals to synchronize the datapath.

Each component is enabled at the suitable moment as a function of:

- The current operation.
- Datapath state (control unit inputs).

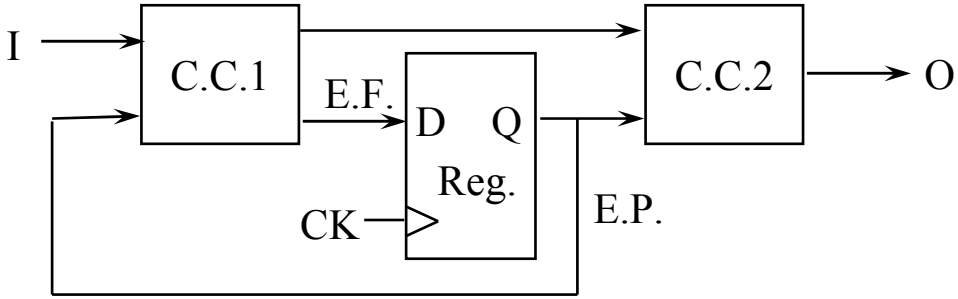


Control input:
Start command, etc.

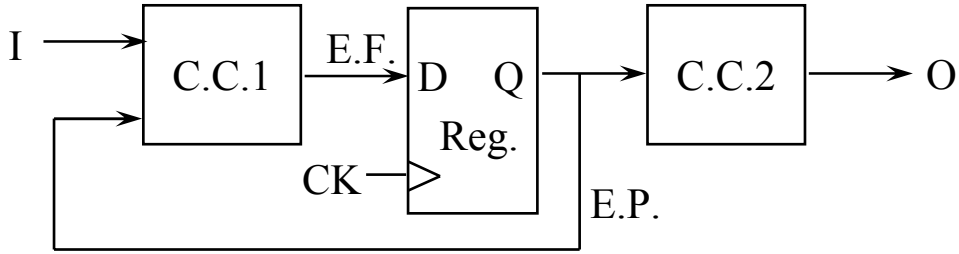
Control output:
Errors, end condition, etc.

It consists in a FSM (Finite State Machine):

Mealy machine



Moore machine



Control unit design

Methodologies:

- Classic FSM procedure with flip-flop minimization.
  - State diagram
  - State transition table
  - Manual or automated minimization...
- One-hot method. One flip-flop is assigned per state.
  - Only one simultaneously active flip-flop (current state).
  - Control signals are generated from the active flip-flop.
  - Straightforward implementation from a flow diagram or an algorithm.
- Microprogramming
  - Programmable logic /memory circuits
  - Control information is organized into microprogrammes.
- Ad-hoc methods.
  - Particular solutions for specific applications.
  - Example: sequence counter (SBC).

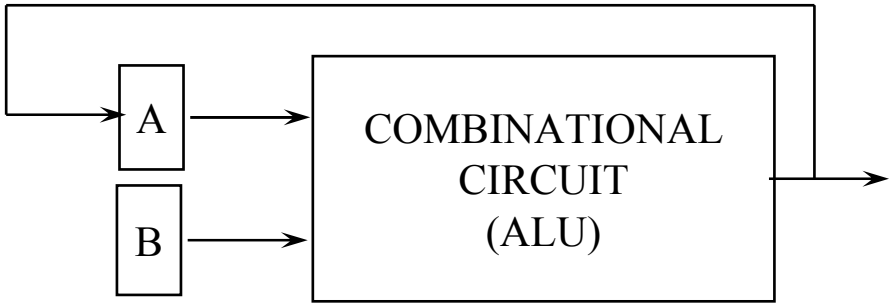
4.1.4. Datapath

Consists of:

1. Combinational circuits that perform operations (functional elements)
2. Registers for information storage
3. Busses for element interconnecting

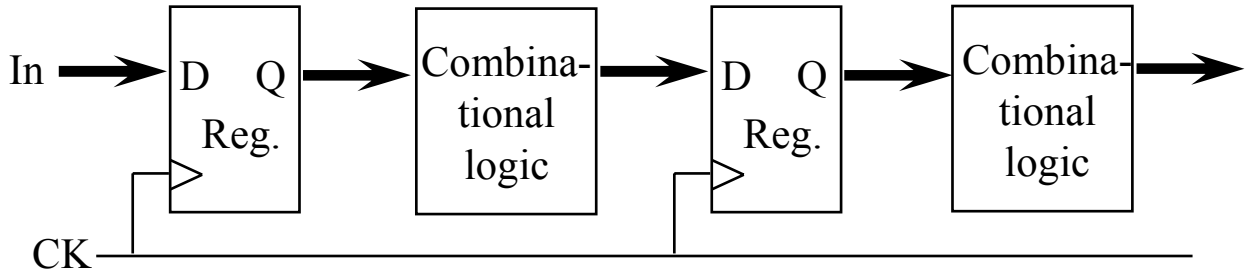
Datapath structures

a) Accumulator.
A register accumulates successive operation results: A = f(A,B).



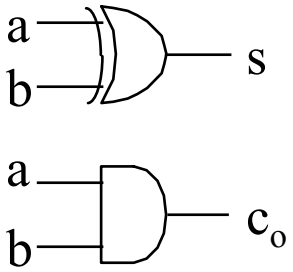
b) Pipeline. Data processing is divided in simpler stages.

- Ancillary registers between stages store partial results.
- All stages execute simultaneously.

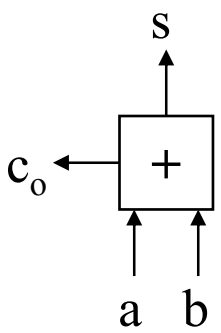


4.2 Datapath operators: adders

Half adder



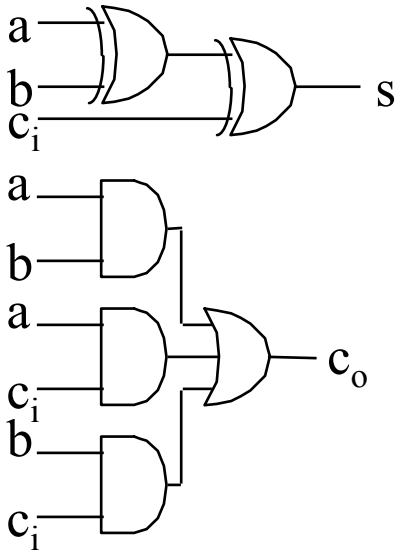
Symbol:



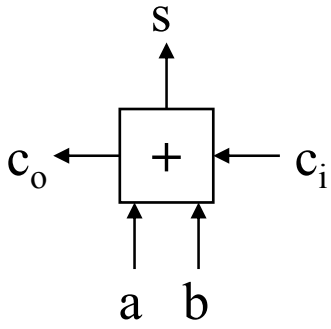
$$s = a \oplus b$$
$$c_o = a \cdot b$$

```
  o  a
+ o  b
-----
o  o
co s
```

One-bit full adder



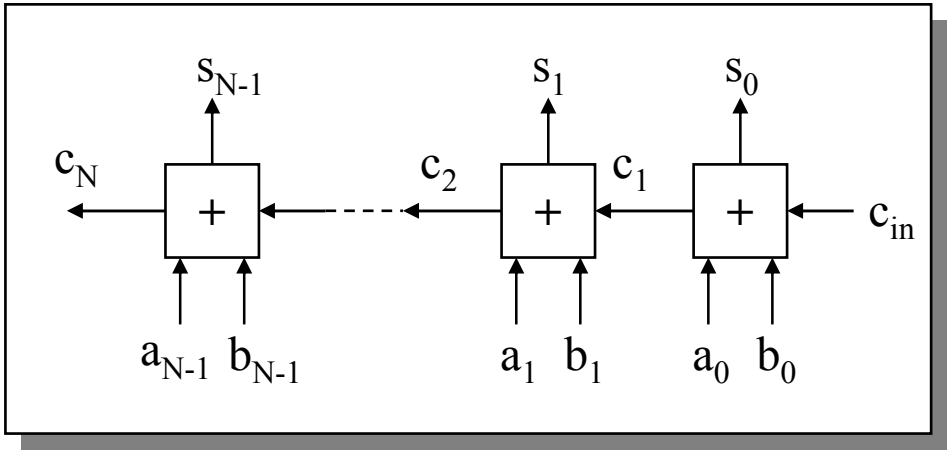
Symbol:



$$s = a \oplus b \oplus c_i$$
$$c_o = a \cdot b + a \cdot c_i + b \cdot c_i$$

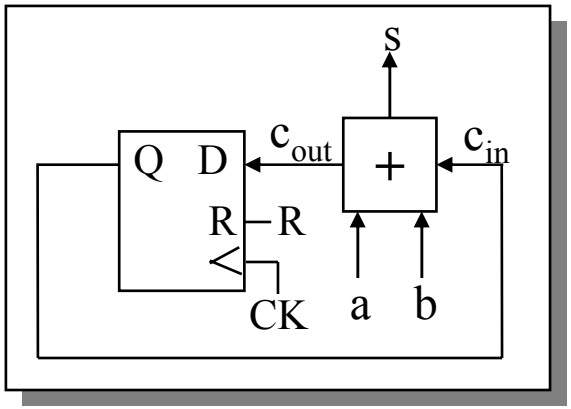
```
  o  cin
  o  a
+ o  b
-----
o  o
co s
```

N-bit binary Carry Propagate Adder (CPA)



$$t_d = (N-1)t_{ci} + t_s$$

Serial binary adder

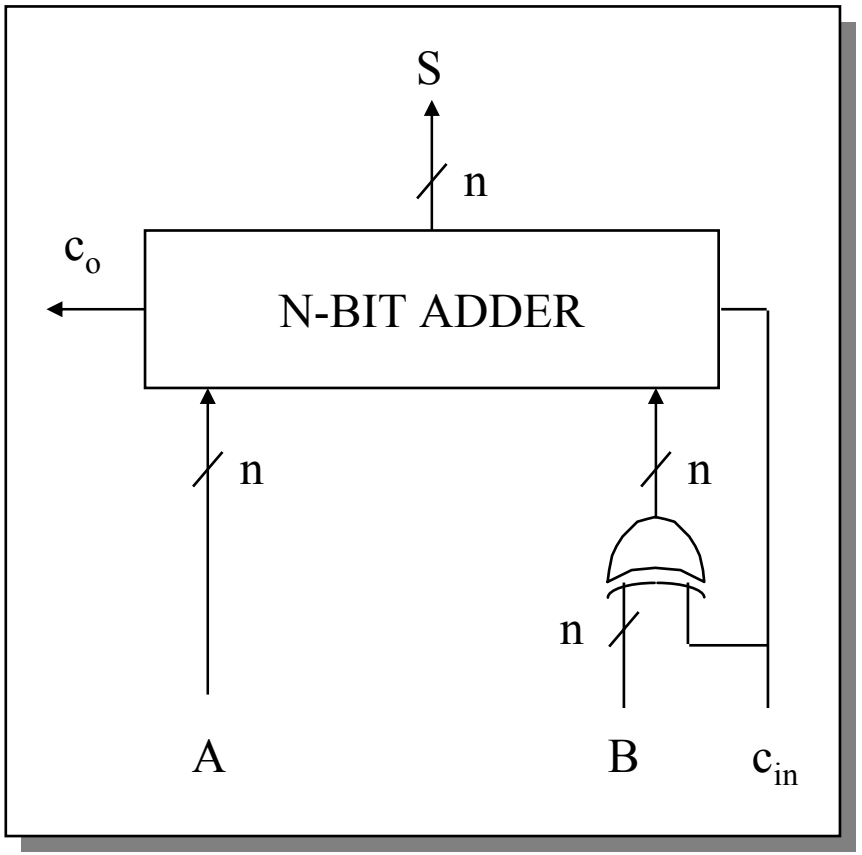


$t_d = t_{sum}$

N bits:

$t_d = N(t_{sum} + t_{setup} + t_q)$

N-bit 2's complement adder-subtracter



add : $c_{in} = 0$

subtract: $c_{in} = 1$

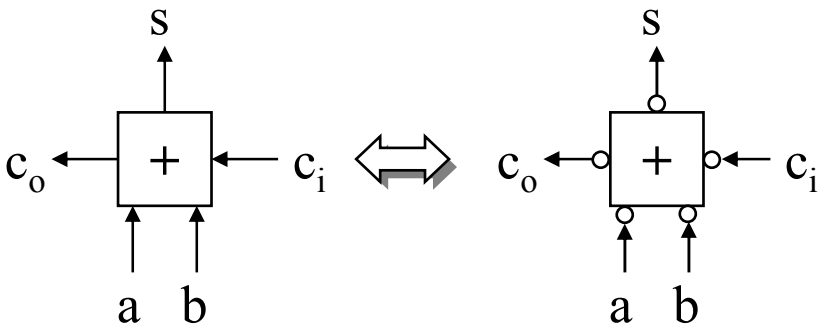
Binary adder implementation

a	b	ci	s	co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

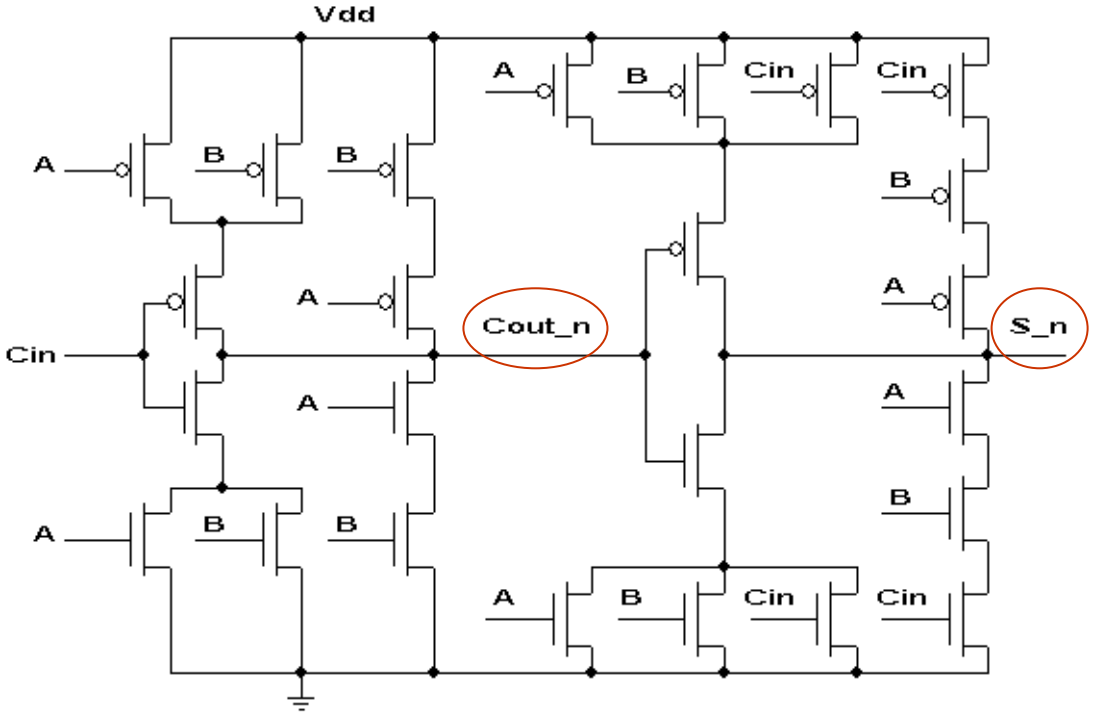
$\overline{p}\,\overline{g}$

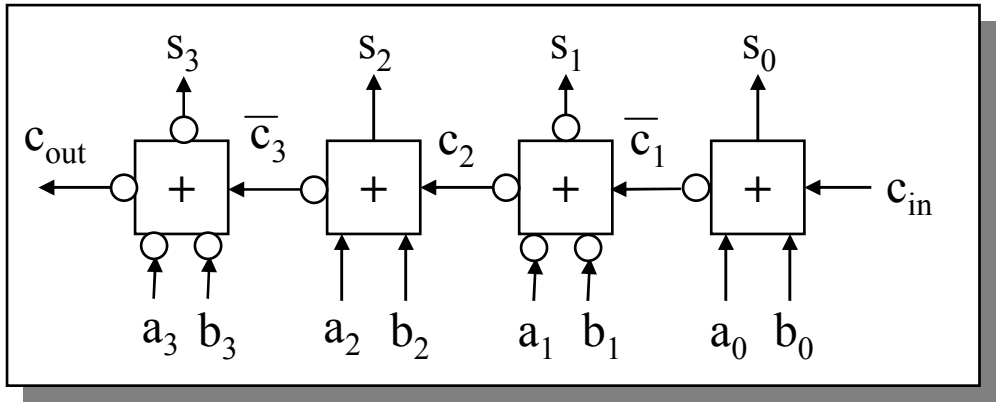
$p \equiv a \oplus b$

$g \equiv a \cdot b$



Transistor-level symmetrical implementation





Carry LookAhead Ader (CLA)

Definitions

Carry generation: $g_i \equiv a_i b_i$

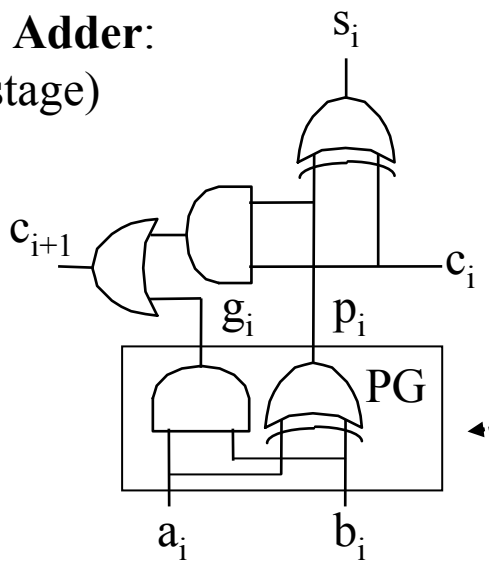
Carry propagation: $p_i \equiv a_i \oplus b_i$

(alternativelyt, $p_i \equiv a_i + b_i$)

Carries can be obtained as:

$c_{i+1} = g_i + p_i \cdot c_i$

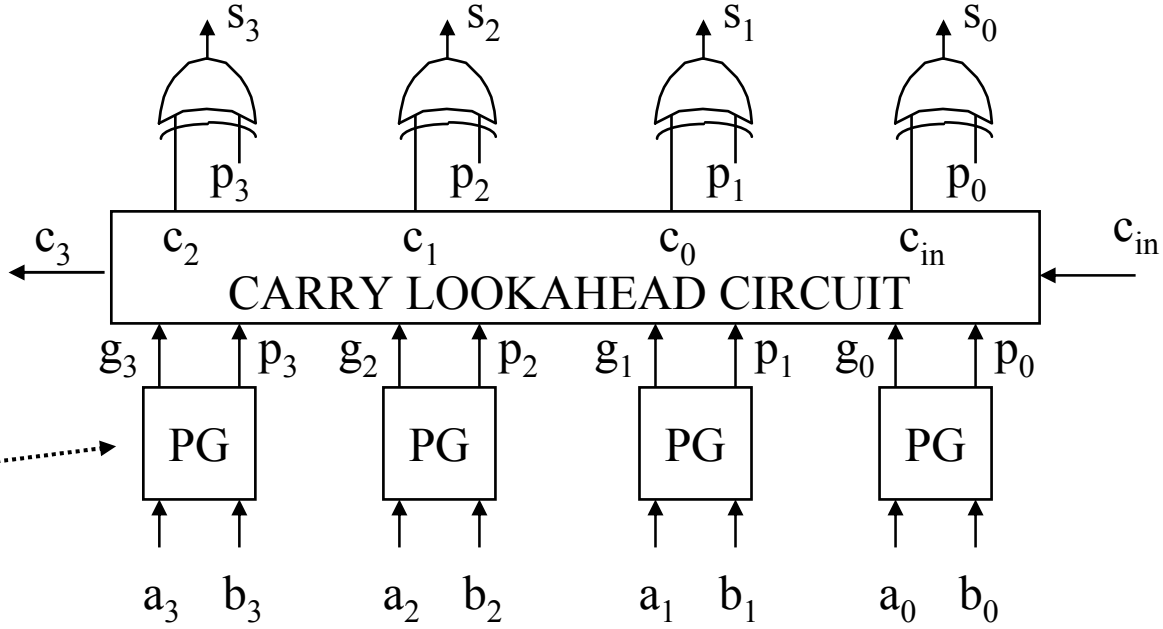
PG Adder:
(1 stage)



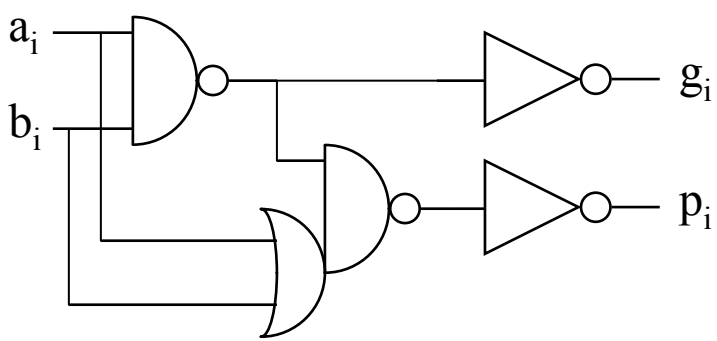
Developing the expression,

$$c_0 = g_0 + p_0 c_{in}$$
$$c_1 = g_1 + p_1 g_0 + p_1 p_0 c_{in}$$
$$c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{in}$$
$$c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{in}$$

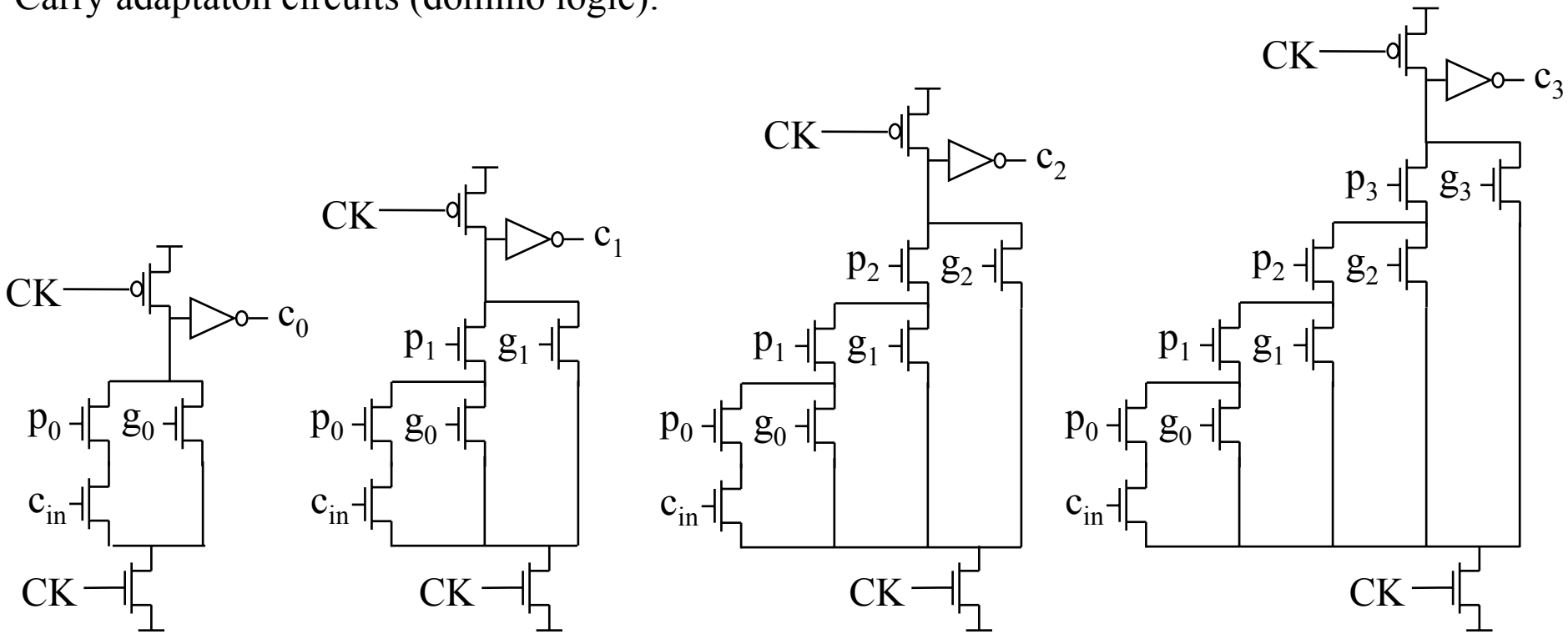
CLA Adder (4-bit example)



PG CMOS logic implementation:

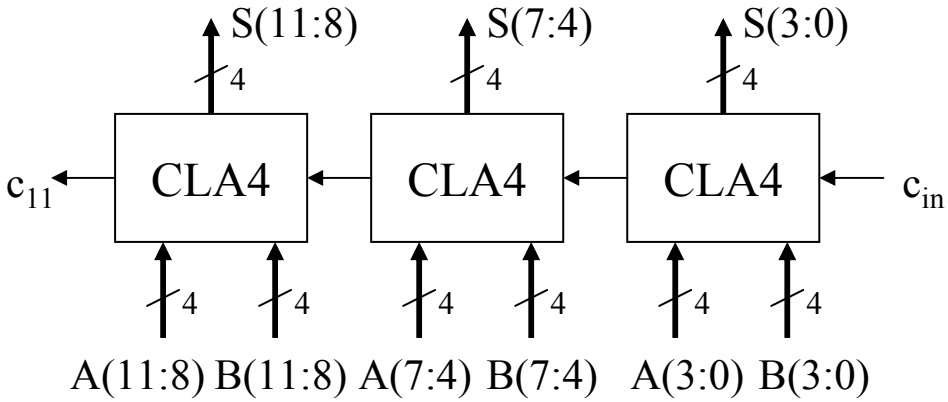


Carry adaptaton circuits (domino logic):



The number of gates and inputs of the carry lookahead circuit grows linearly with the number of stages in a CLA.
Because of this, CLAs are usually limited to 4 bits.

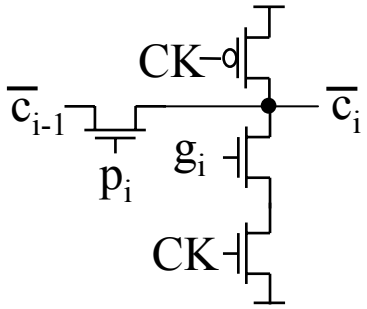
Example: 12-bit adder.



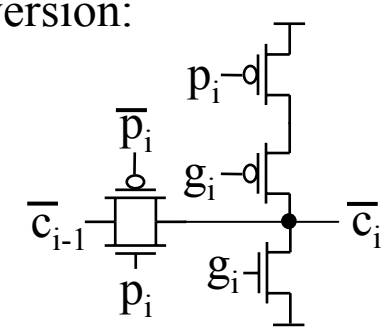
The carry lookahead scheme can be hierarchically applied to accelerate the computation.

Manchester adder

It is based on a a fast direct carry propagate circuit.
Dynamic version:



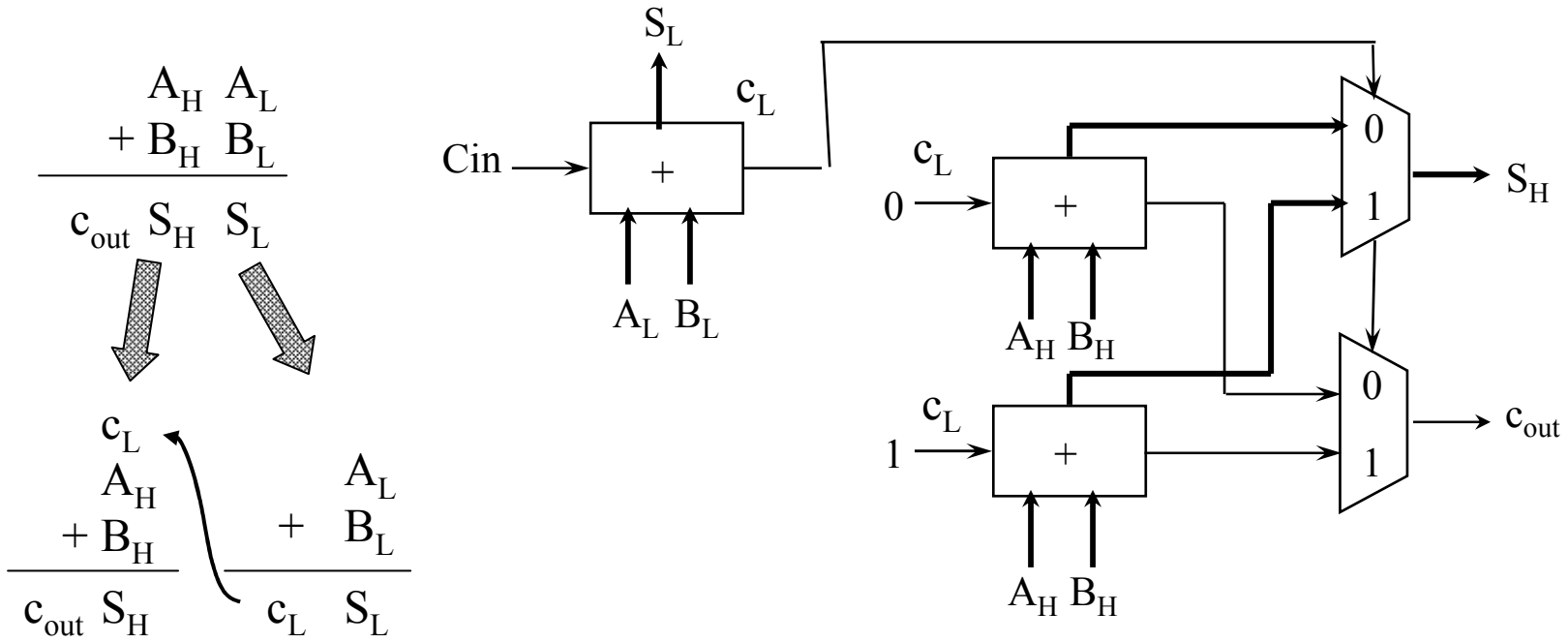
Static version:



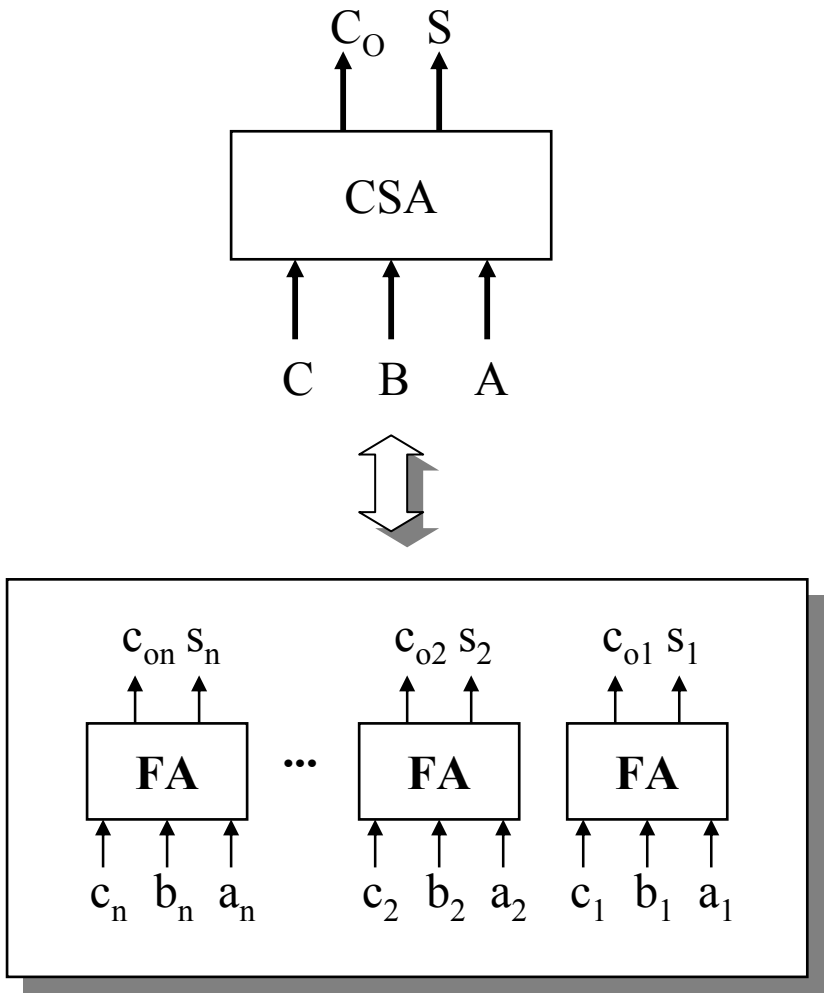
Carries are obtained by propagation through successive cells.

Carry Select Adder

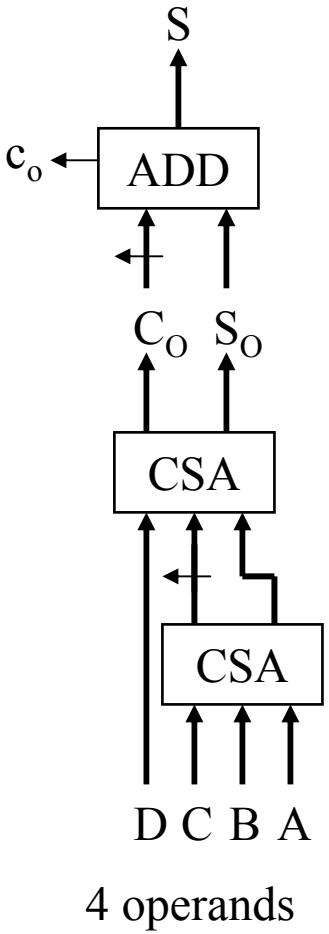
- The addition is divided into parts.
- Second stage: Speculative addition for $c_L = 0$ and $c_L = 1$.
- Correct result selection as a function of c_L .



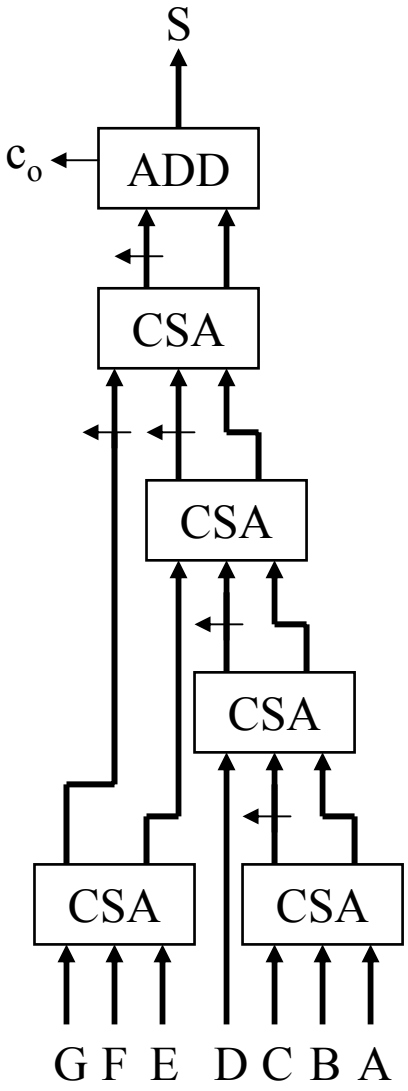
Multioperand adders: Carry Save Adder (CSA)



Examples:

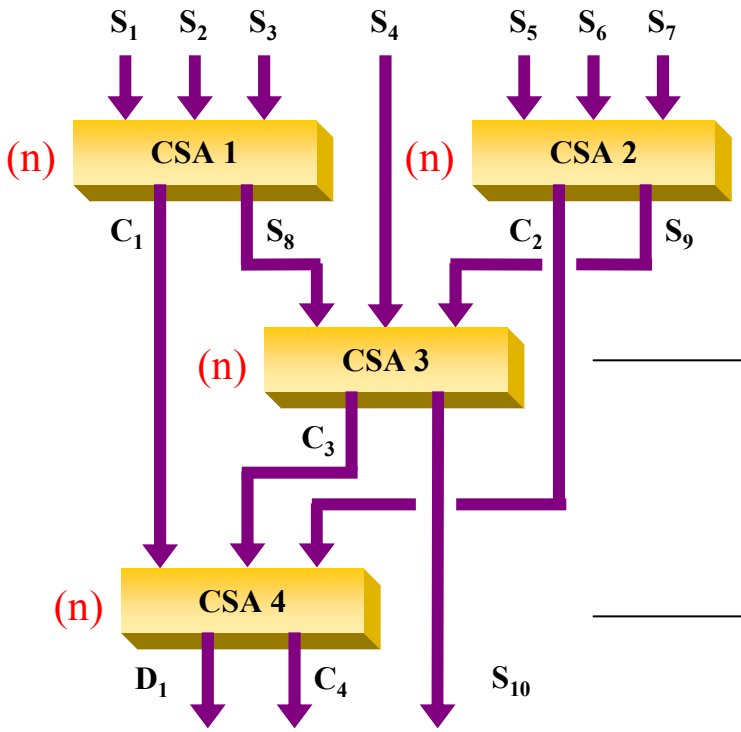


4 operands



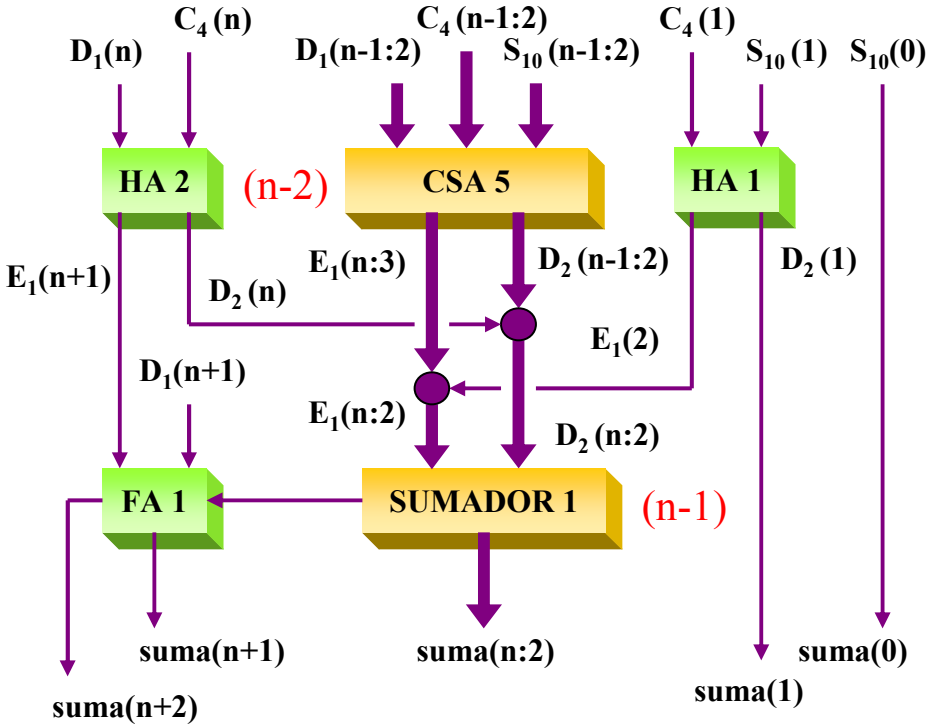
7 operands

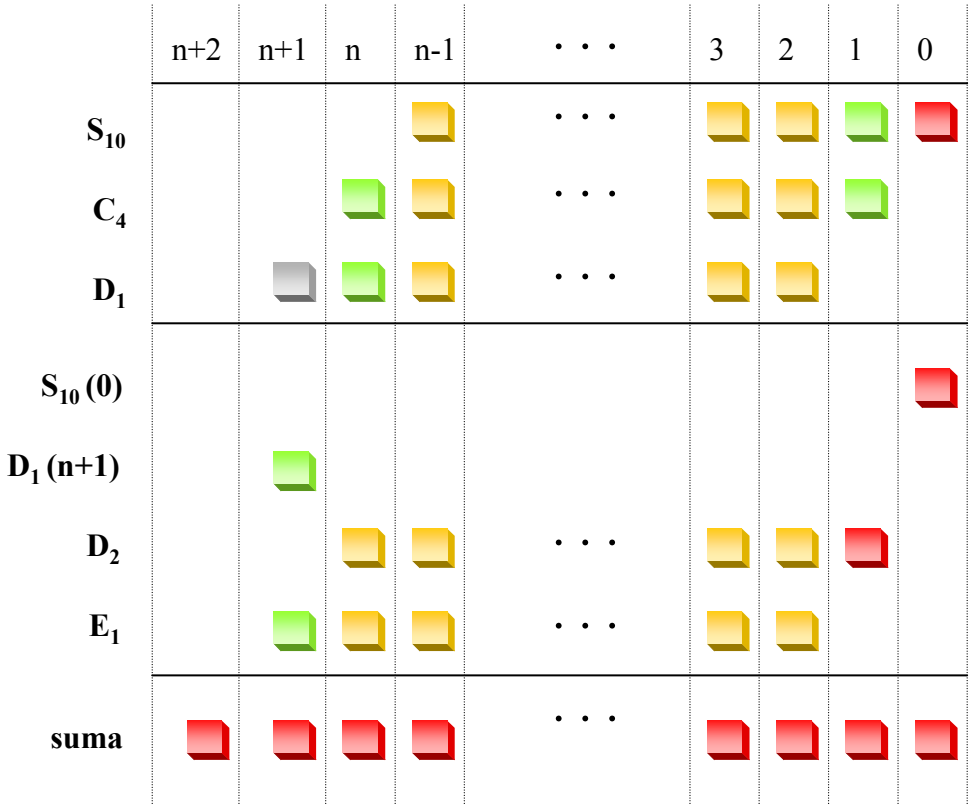
7-input Carry Save Adder design example (1)



	n+1	n	n-1	· · ·	3	2	1	0	
$S_1 \cdots S_7$			■	· · ·	■	■	■	■	× 7
S_4, S_8, S_9			■	· · ·	■	■	■	■	× 3
C_1, C_2		■	■	· · ·	■	■	■		× 2
S_{10}			■	· · ·	■	■	■	■	× 1
C_1, C_2, C_3		■	■	· · ·	■	■	■		× 3
S_{10}			■	· · ·	■	■	■	■	× 1
C_4		■	■	· · ·	■	■	■		× 1
D_1	■	■	■	· · ·	■	■			× 1

7-input Carry Save Adder design example (2)



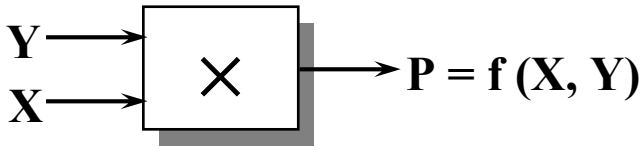


Total number of adders:
Full Adders: 6n –2
Half Adders: 2

4.3 Datapath operators: multipliers

Used in:

- General-purpose architectures
- Specific architectures (DSP, neural nets, etc.)



Definitions:

Multiplicand	$Y = y_{n-1}y_{n-2}\cdots y_0$
Multiplier	$X = x_{n-1}x_{n-2}\cdots x_0$
Product	$P = p_{m-1}p_{m-2}\cdots p_0$

Length of P: m = 2n bits. Demonstration (assuming n > 1):

$X_{MAX} = 2^n-1$; $Y_{MAX} = 2^n-1$;

$P_{MAX} = X_{MAX} * Y_{MAX} = 2^{2n}-2^{n+1}+1 \begin{cases} < 2^{2n}-1 \\ > 2^{2n-1}-1 \end{cases}$

2n bits to encode P_{MAX}

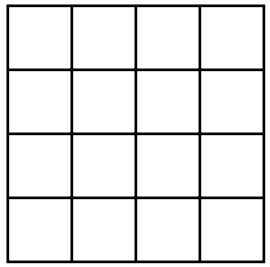
		← n →	
$10\cdots0$	$00\cdots$	00	2^{2n}
$-$	$10\cdots$	00	-2^{n+1}
	$+$	01	1
$01\cdots1$	$00\cdots$	01	P_{MAX}
	← 2n →		

Implementation alternatives

- Combinational function
  - Karnaugh tables
  - Automated simplification

  ⇒ High complexity, Irregular circuits

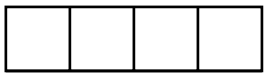
- Modular approximation. Product algorithm



Area-delay tradeoff

Area reduction: *Product sequentialization* (logic multiplexing)

- Parallel/series, series/parallel, series/series multipliers



Design based on the product algorithm

Regular architecture ➔ scalable
➔ time multiplexable

Binary product algorithm
(example for 3 bits):

```
                        y2    y1    y0
                        x2    x1    x0
                    ------------------
                        x0y2  x0y1  x0y0
                  x1y2  x1y1  x1y0
            x2y2  x2y1  x2y0
    ------------------------------------
     p5    p4    p3    p2    p1    p0
```

Resources:
- 1-bit multipliers (2-input logic AND)
- 1-bit full adders

Alternatives:
- Combinational product
  - 1 to 1 mapping of sums and products
  - Parallel-parallel multiplier
- Sequential product
  - Time multiplexing logic products and sums.

Classification as a function of data flow

Multiplicand	Multiplier	Clock cycles	Adders (1 bit)	AND gates	Control
Parallel	Parallel	1	n^2	n^2	no
Parallel	Serial	n	n	n	sí
Serial	Parallel	2n	n	n	sí
Serial	Serial	n^2	1	1	sí

Parallel-parallel (or array) product

- Combinational circuit
- Logic delay can be high
- No need of control logic

Sequential product

- Sequential circuit
- Area savings
- More clock cycles
- Control logic needed to synchronize the product cycles

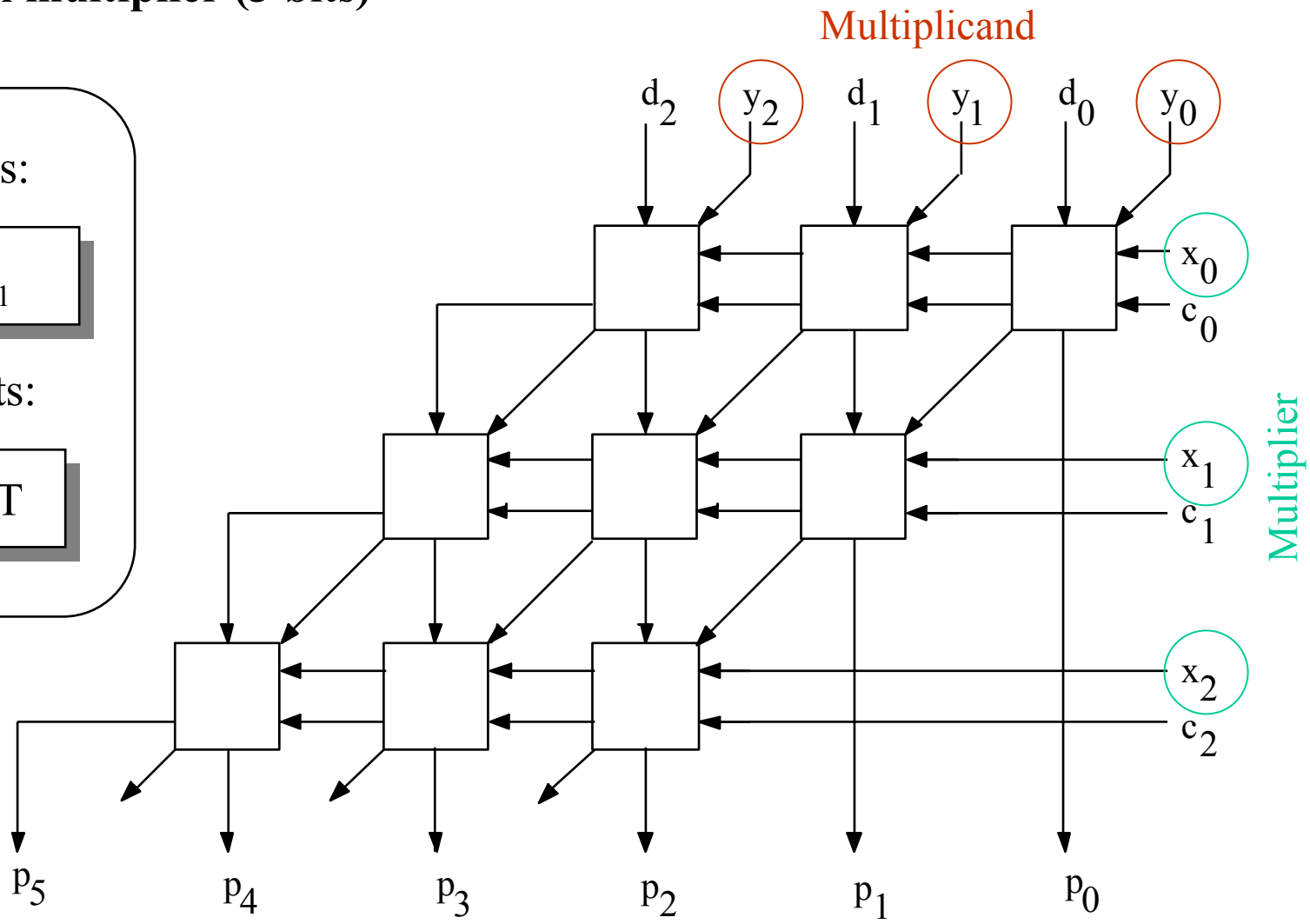
Parallel-parallel multiplier (3 bits)

Area for n bits:

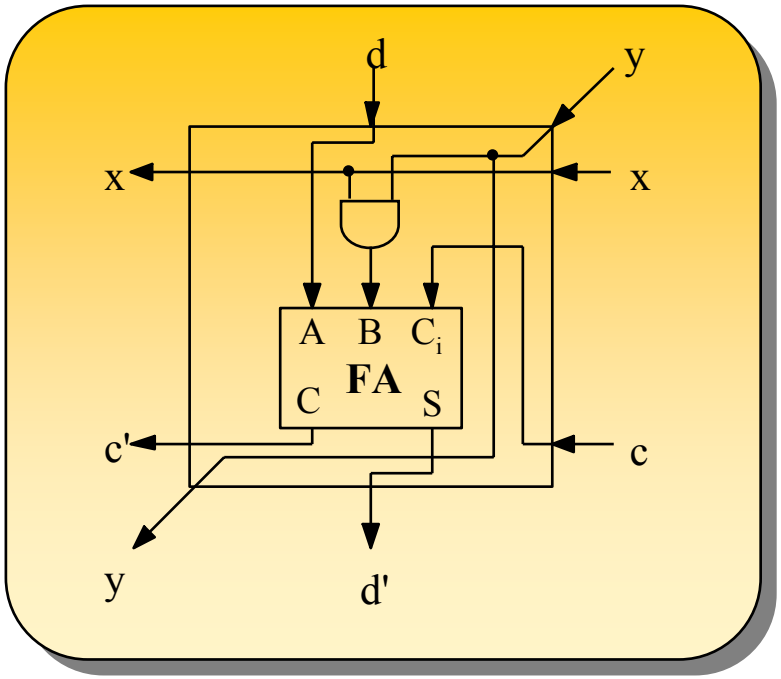
$$A \cong n^2 A_1$$

Delay for n bits:

$$t_d = (3n-2)T$$



Parallel-parallel multiplier : Basic cell



$$c' = xyc + cd + xyd$$
$$d' = d \oplus (xy) \oplus c$$

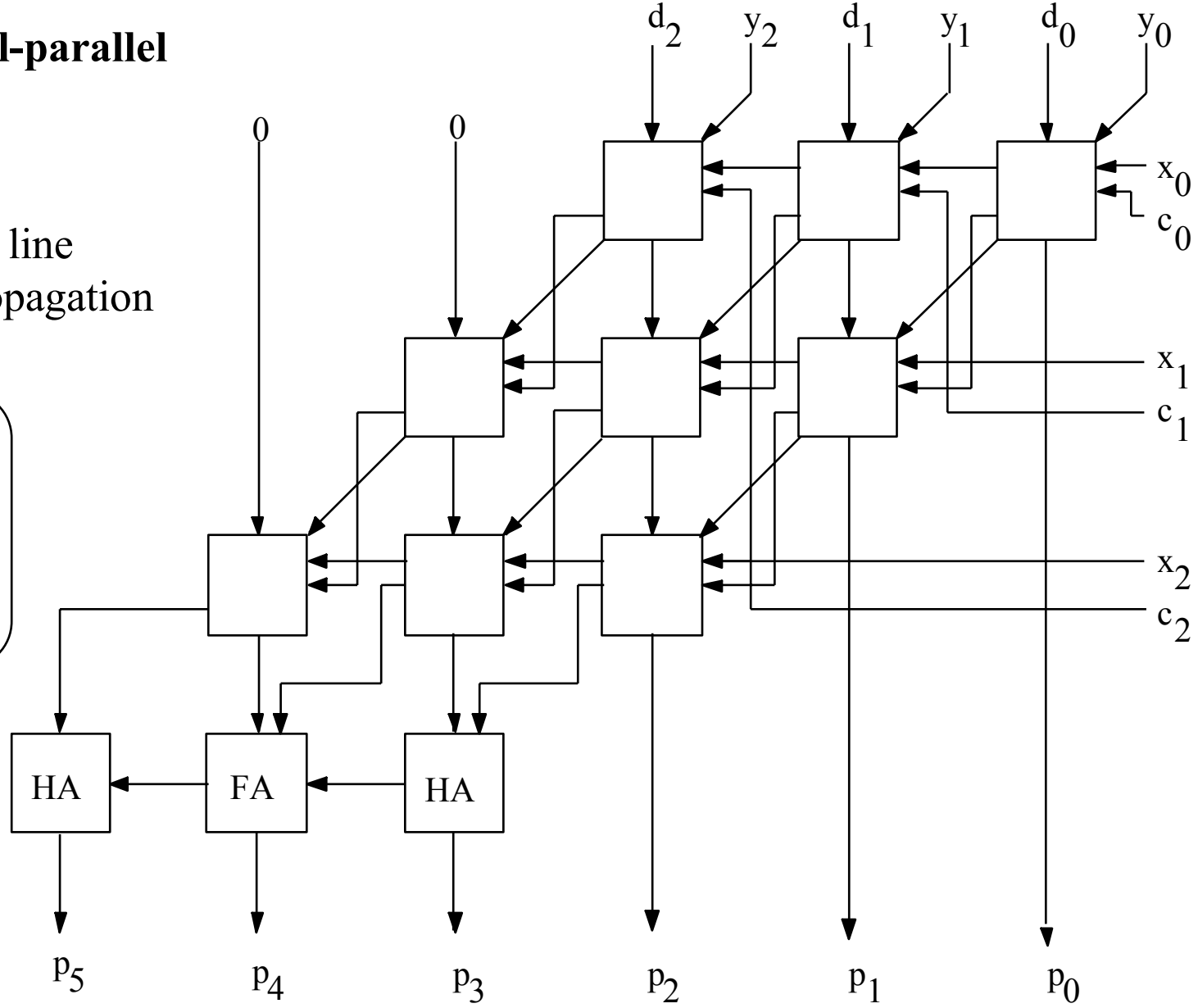
Two extra terms can be added to the product

Carry-save parallel-parallel multiplier (3 bits)

- More speed
- Additional adding line
- Diagonal carry propagation

$A \cong n(n+1)A_1$

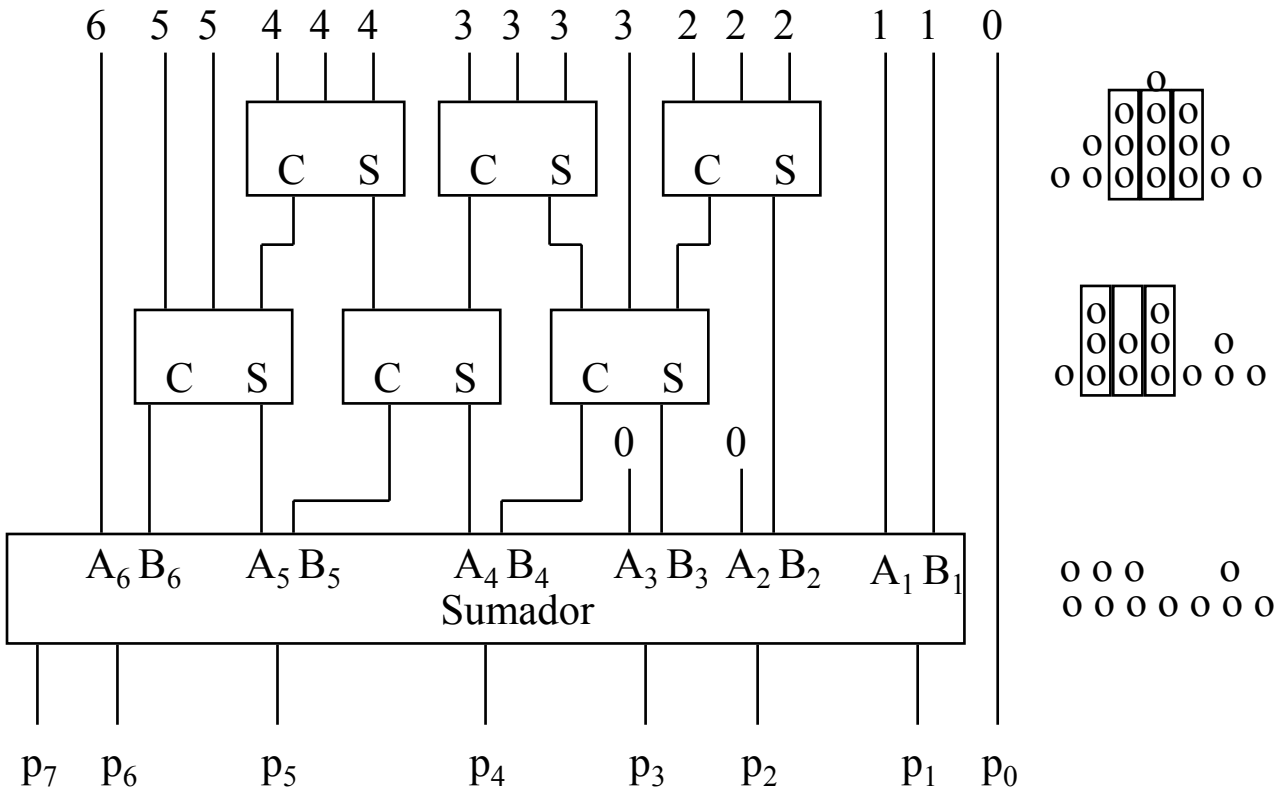
$t_d = 2nT$



Wallace trees

- Another kind of combinational (parallel-parallel) multiplier.
- Principle: Partial product addition using a carry-save scheme.

Example (4x4 bits):



Parallel-serial multipliers : Robertson multiplier

Example:
3-bit product

			1	0	1	Y
			1	1	0	X
			0	0	0	$P_1 \leftarrow x_0 \times Y \times 2^0$
		1	0	1		$P_2 \leftarrow x_1 \times Y \times 2^1 + P_1$
	1	0	1			$P_3 \leftarrow x_2 \times Y \times 2^2 + P_2$
0	1	1	1	1	0	$P \leftarrow X \times Y$

Algorithm:

$P \leftarrow 0$

repeat (i=0..n-1)

$P \leftarrow (P + x_i \times Y) \times 2^{-1}$

$P \leftarrow P \times 2^n$

$$P \leftarrow \begin{cases} P \times 2^{-1} & \text{si } x_i = 0 \\ (P + Y) \times 2^{-1} & \text{si } x_i = 1 \end{cases}$$

Requirements: ANDs, adders and shifts

Elementary operations for a 3-bit product

	y_2 y_1 y_0		
	1 1 1,		Y = 7
	1 0 1,		X = 5
	0 0 0, 0 0 0		$P \leftarrow 0$
0	1 1 1, 0 0 0		$P \leftarrow P + Y \quad (x_0 = 1)$
	0 1 1, 1 0 0		$P \leftarrow P \times 2^{-1}$
0	0 1 1, 1 0 0		$P \leftarrow P \quad (x_1 = 0)$
	0 0 1, 1 1 0		$P \leftarrow P \times 2^{-1}$
1	0 0 0, 1 1 0		$P \leftarrow P + Y \quad (x_2 = 1)$
	1 0 0, 0 1 1		$P \leftarrow P \times 2^{-1}$
	1 0 0 0 1 1,		**P** $\leftarrow P \times 2^3$ P = 35
	p_5 p_4 p_3 p_2 p_1 p_0		

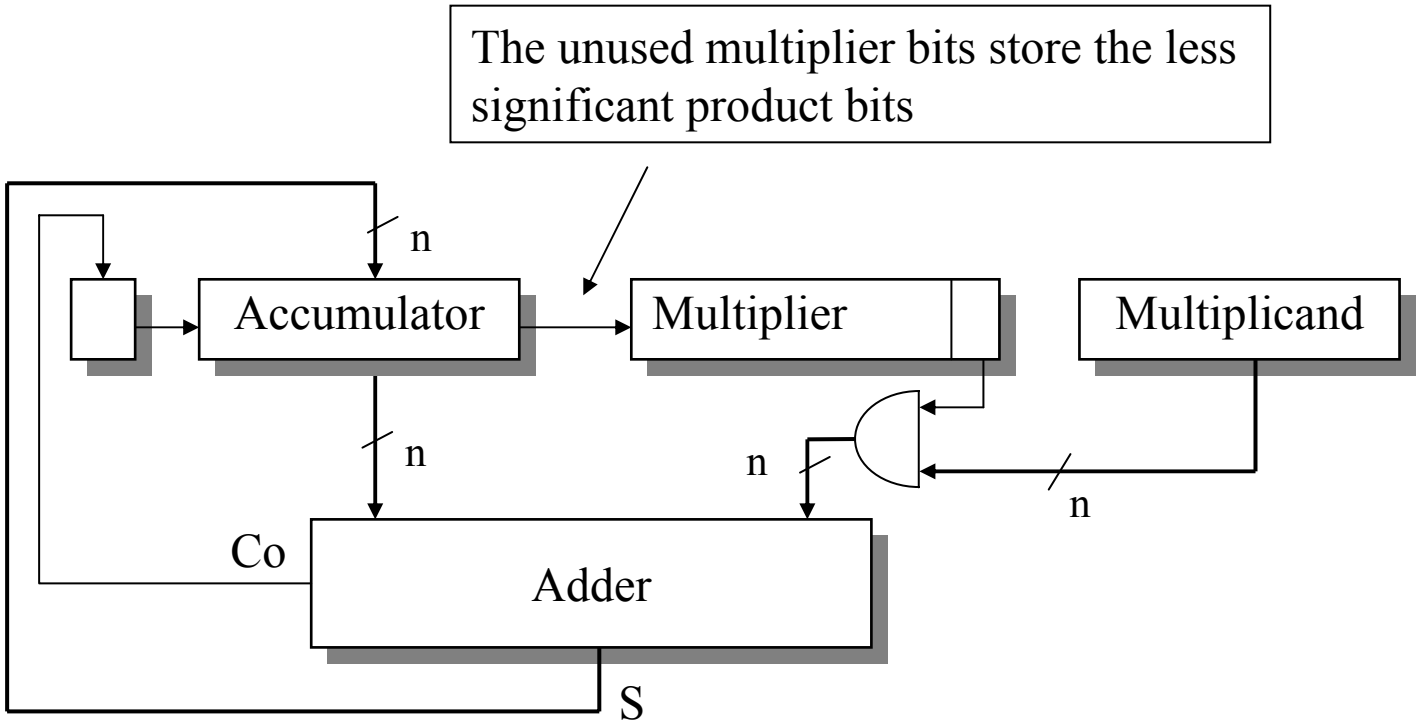
$i \leftarrow 0$

$i \leftarrow i + 1$

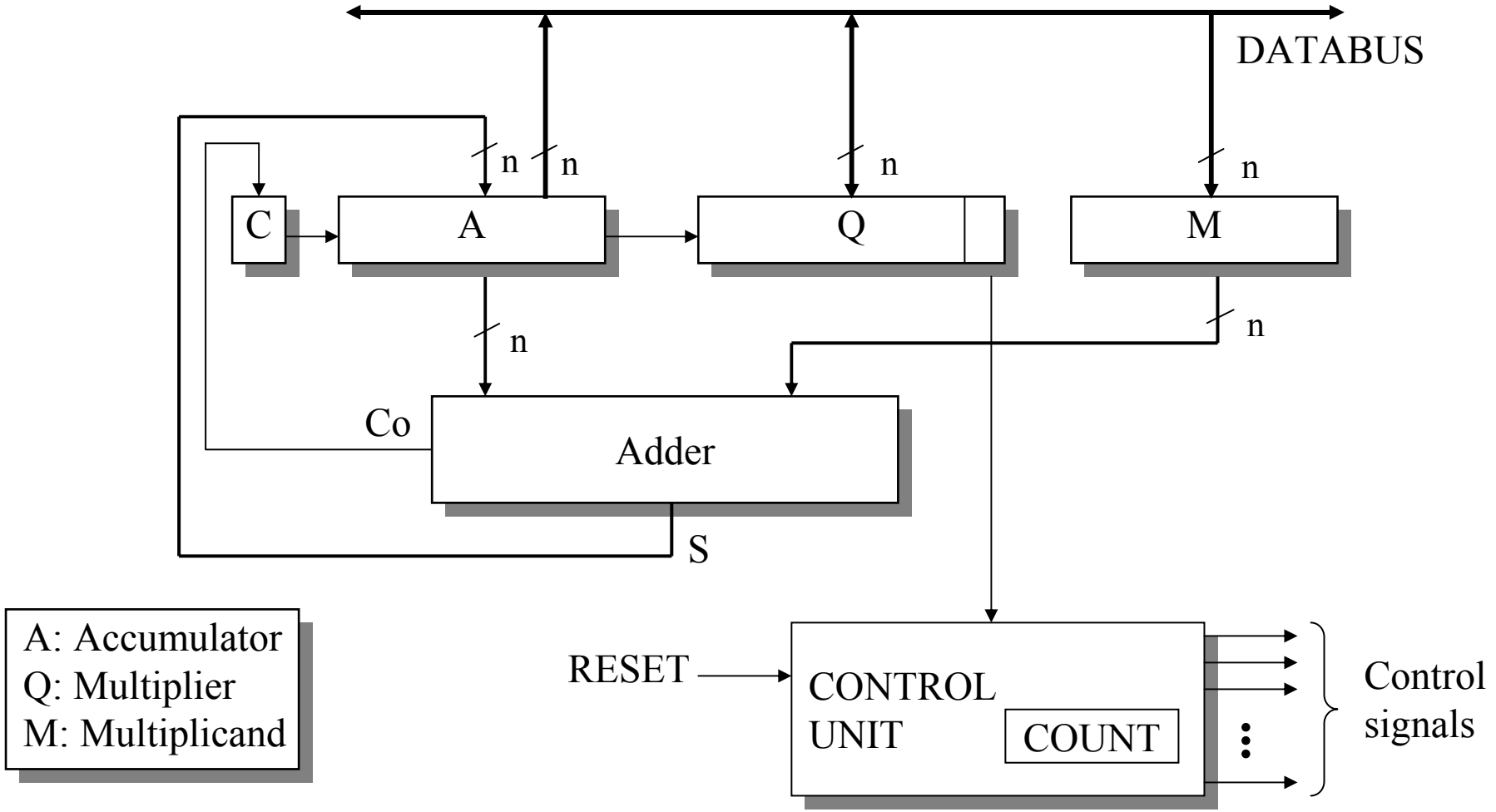
$i \leftarrow i + 1$

STOP (i = 2)

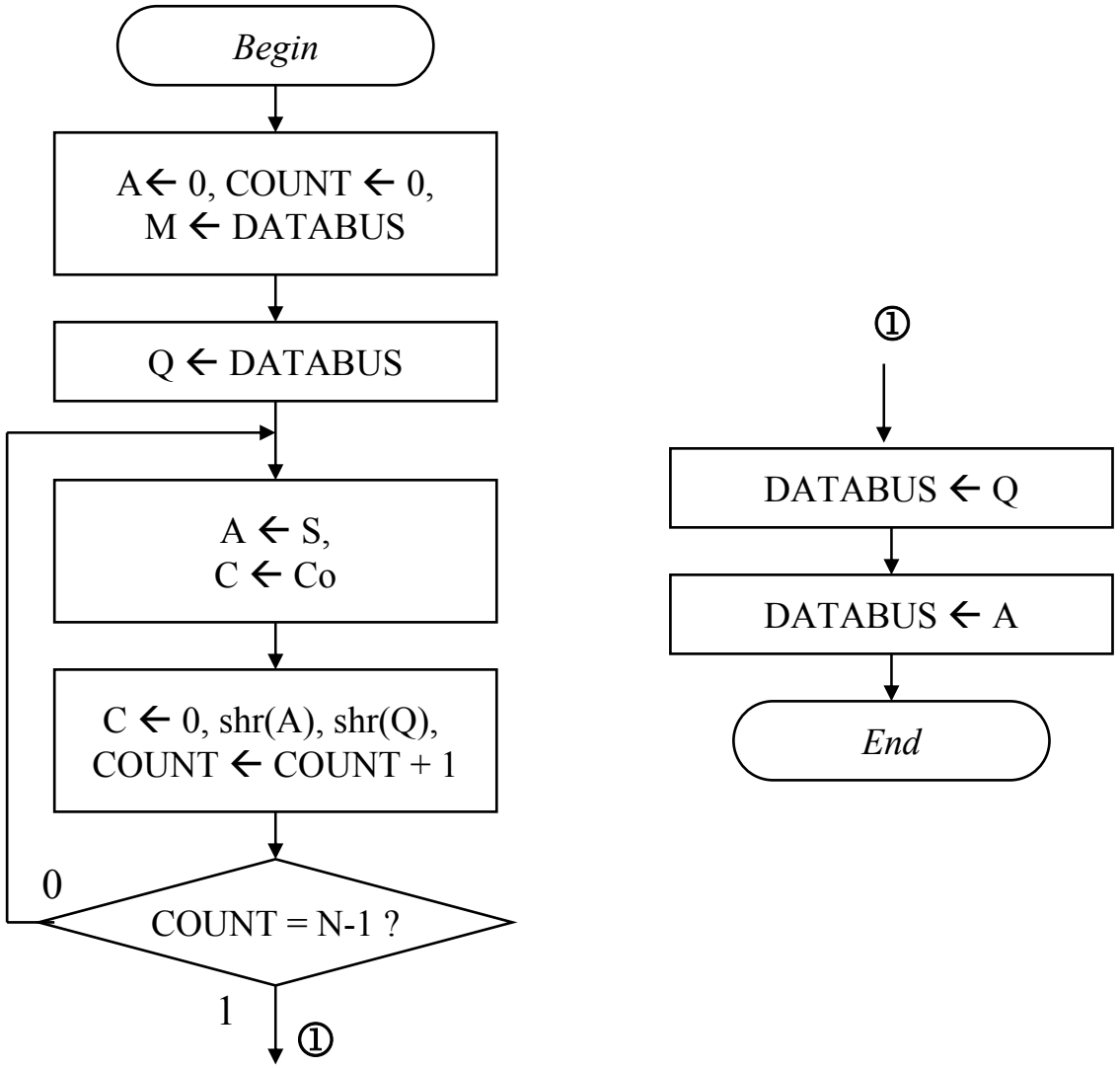
Unsigned multiplier structure (control not included)



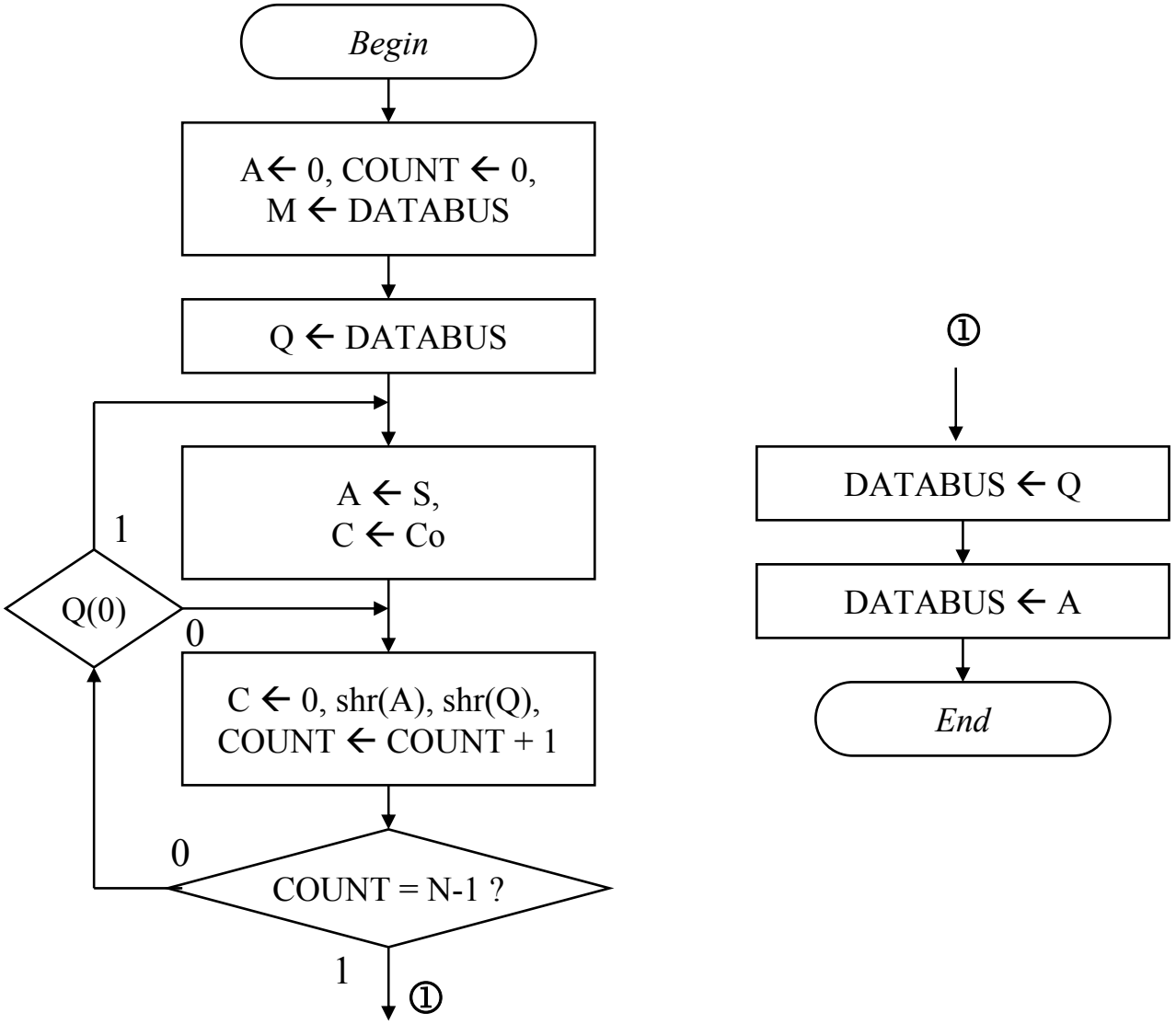
Unsigned multiplier structure (AND by control)



Unsigned multiplier ASM diagram



Unsigned multiplier ASM diagram (AND by control)



Operacions elementals per a un producte de 3 bits

y_2	y_1	y_0	q_2	q_1	q_0		
1	1	1,				Y = 7	
1	0	1,				X = 5	
0	0	0,	1	0	1	$P \leftarrow 0$	$i \leftarrow 0$
1	1	1,	1	0	1	$P \leftarrow P + Y$ $(x_0 = 1)$	
0	1	1,	1	1	0	$P \leftarrow P \times 2^{-1}$	$i \leftarrow i + 1$
0	1	1,	1	1	0	$P \leftarrow P$ $(x_1 = 0)$	
0	0	1,	1	1	1	$P \leftarrow P \times 2^{-1}$	$i \leftarrow i + 1$
0	0	0,	1	1	1	$P \leftarrow P + Y$ $(x_2 = 1)$	
1	0	0,	0	1	1	$P \leftarrow P \times 2^{-1}$	STOP (i = 2)
1	**0**	**0**	**0**	**1**	**1**,	**P** $\leftarrow P \times 2^3$ P = 35	
p_5	p_4	p_3	p_2	p_1	p_0		

2's-complement product

If $A = a_{n-1} \cdot\cdot a_0 = \sum_{i=0}^{n-1} a_i 2^i$ is a 2's-complement number (sign bit: a_{n-1}),

- For A > 0, $a_{n-1} = 0$, it can be represented as

$$A = a_{n-2} \cdot\cdot a_0 = \sum_{i=0}^{n-2} a_i 2^i$$

- When A < 0, $a_{n-1} = 1$ and

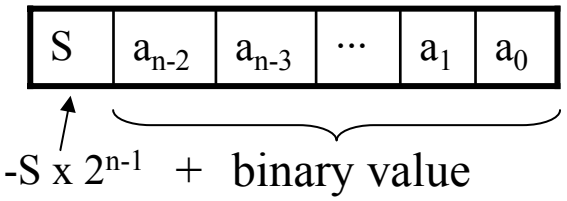
$$-A = C\,a\,2(A) = \overline{A} + 1 = \underbrace{11\cdot\cdot1}_{\text{n bits}} - a_{n-1} \cdot\cdot a_0 + 1 = \underbrace{10\cdot\cdot0}_{\text{n bits}} - a_{n-2} \cdot\cdot a_0$$

thus,

$$\boldsymbol{A = -10\cdot\cdot0 + a_{n-2} \cdot\cdot a_0 = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i}$$

Since for A > 0 $\Rightarrow$ $a_{n-1} = 0$, the previous expression becomes general:

$$\boldsymbol{A = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i}$$



The product for multiplier Q < 0 can be done as follows:

$$M \times Q = M \times \left(-q_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} q_i 2^i \right) = q_0 M + q_1 2M + \cdots + q_{n-2} 2^{n-2} M - q_{n-1} 2^{n-1} M =$$

$$= q_0 M + q_1 2M + \cdots + q_{n-2} 2^{n-2} M + q_{n-1} 2^{n-1} Ca2(M)$$

- **For the first n-1 multiplier bits: conventional product**
- **Sign bit: multiplicand is 2's complemented**
- Expression also valid for Q ≥ 0, because $q_{n-1} = 0$

For M < 0,

- When the accumulator is positive or zero, **zeroes** are introduced from the left.
- When the accumulator is negative, **ones** are introduced from the left (negative numbers shift)

For M ≥ 0, always **zeroes** are introduced from the left.

Therefore, **in general the sign bit is introduced from the left, but the overflow condition has to be taken into account.**

Example (1/2): 3-bit 2's-complement product

m_2 m_1 m_0	
1 0 1, 1 1 0,	M = -3 Q = -2
0 0 0, 0 0 0	P ← 0
0 0 0, 0 0 0 0 0 0, 0 0 0	P ← P ($q_0 = 0$) P ← P × 2^{-1}
1 0 1, 0 0 0 1 1 0, 1 0 0	P ← P + M ($q_1 = 1$) P ← P × 2^{-1}
0 0 1, 1 0 0 0 0 0, 1 1 0	P ← P - M ($q_2 = 1$) P ← P × 2^{-1}
0 0 0 1 1 0	**P** ← P × 2^3 P = +6
p_5 p_4 p_3 p_2 p_1 p_0	

Sign bit is kept for shifting

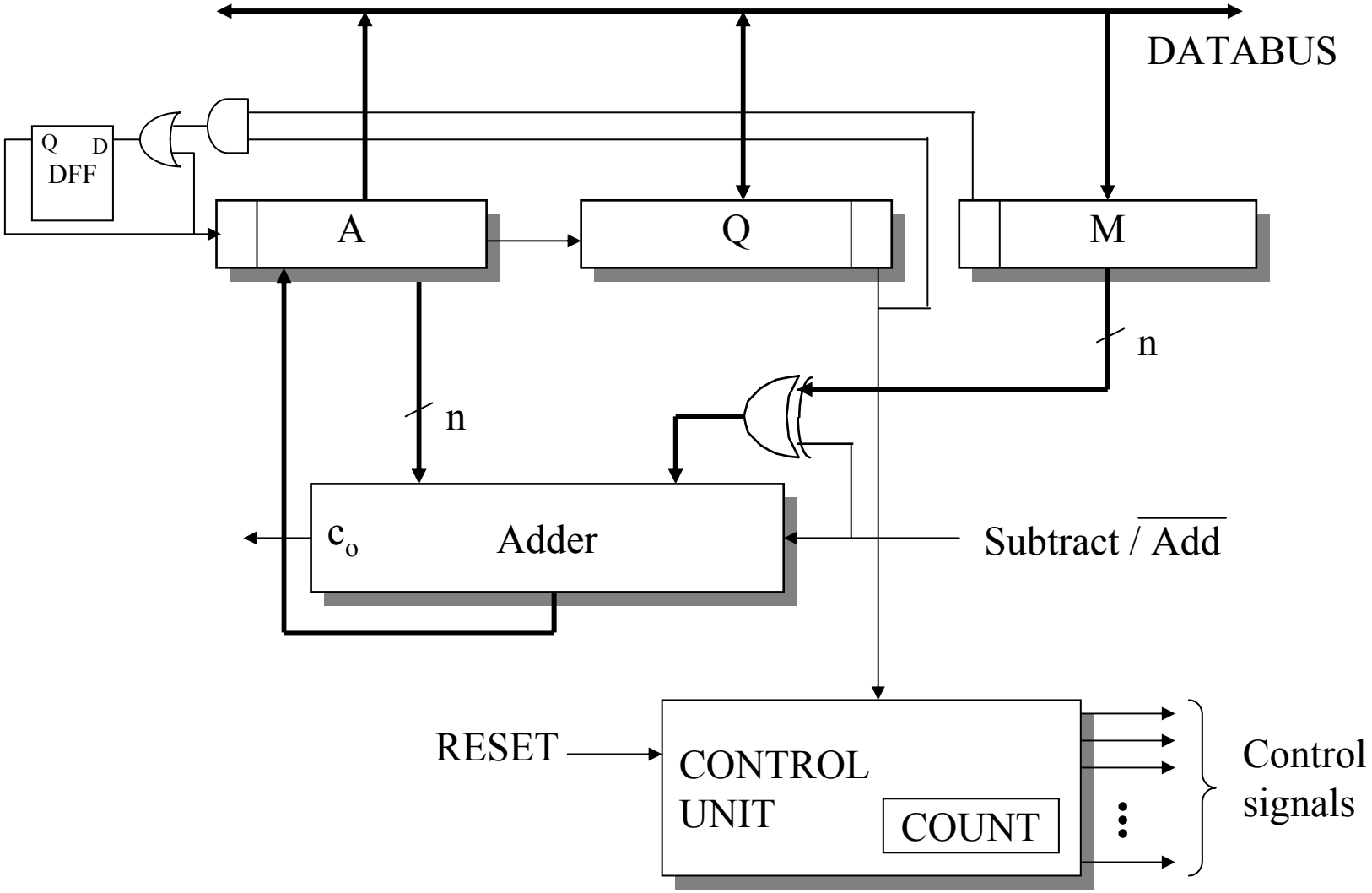
At the last step, M 2's-complement is added

Example (2/2): 3-bit 2's-complement product

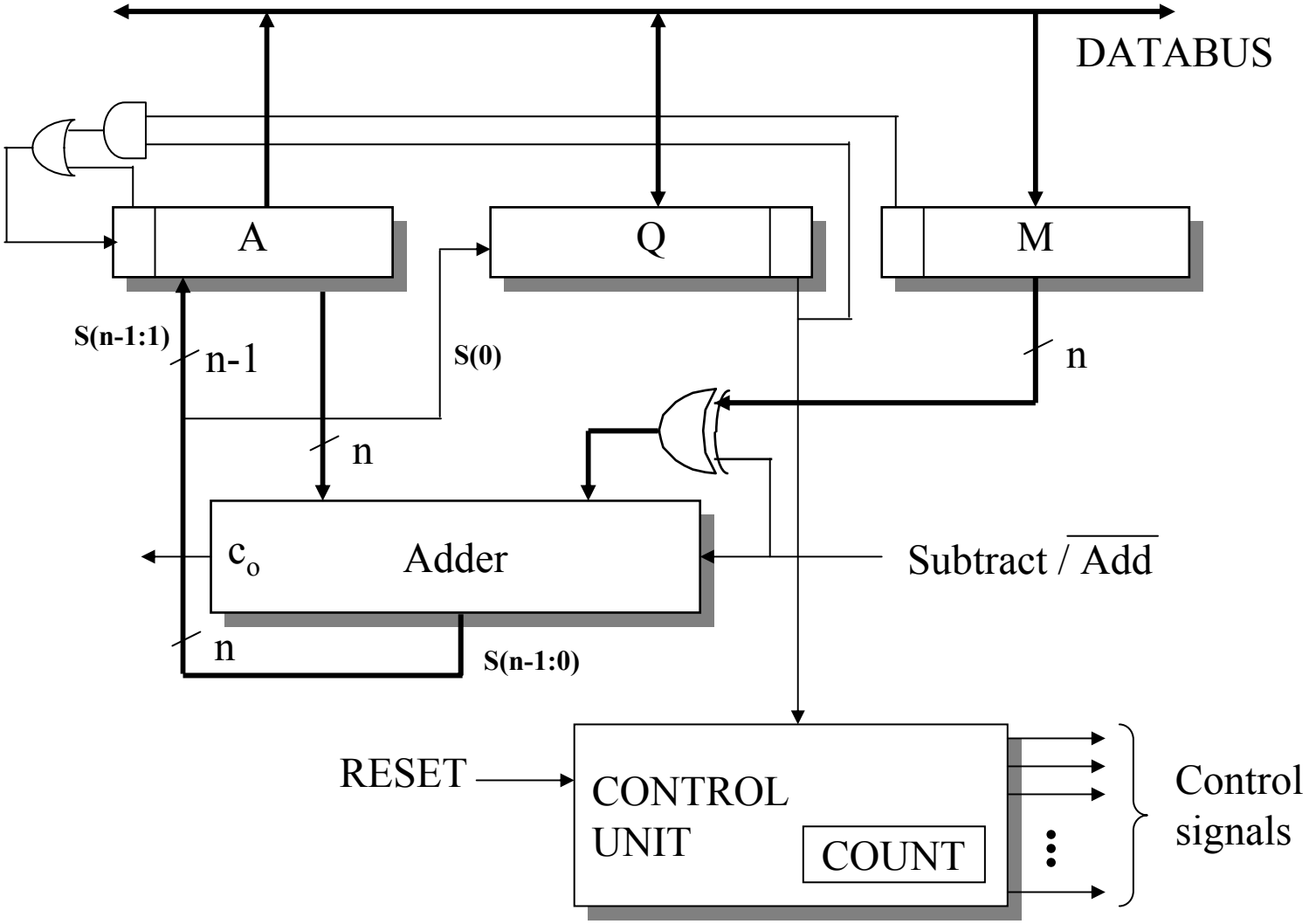
m_2 m_1 m_0	
1 0 1, 0 1 1,	M = -3 Q = 3
0 0 0, 0 0 0	P ← 0
1 0 1, 0 0 0 1 1 0, 1 0 0	P ← P + M (q_0 = 1) P ← P × 2^{-1}
1 0 1, 1 0 0 1 0 1, 1 1 0	P ← P + M (q_1 = 1) P ← P × 2^{-1}
1 0 1, 1 1 0 1 1 0, 1 1 1	P ← P (q_2 = 0) P ← P × 2^{-1}
1 1 0 1 1 1	**P** ← P × 2^3 P = -9
p_5 p_4 p_3 p_2 p_1 p_0	

Overflow in sign bit

2's-complement multiplier structure

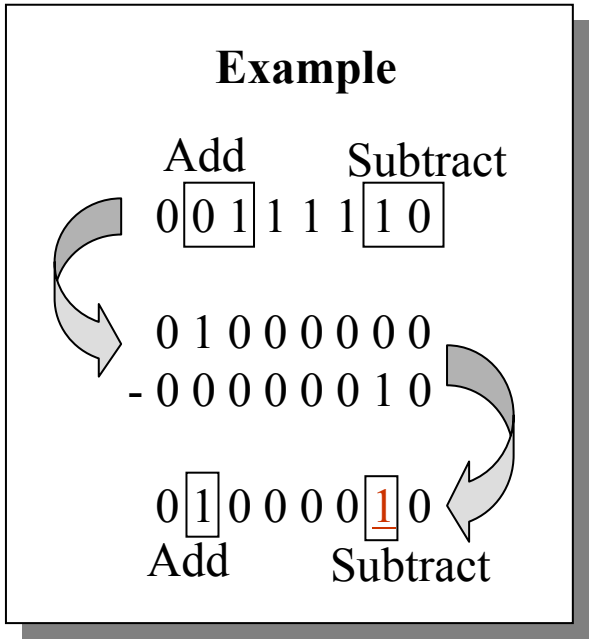


2's-complement multiplier structure. Single cycle add-shift.



Booth's algorithm

Consists in encoding the **multiplier** bits:



x_i	x_{i-1}	Code	Operation
0	0	0	----------
0	1	1	Add multiplicand
1	0	$\underline{1}$	Subtract multiplicand
1	1	0	----------

Evaluated dígit

- For long chains of ones or zeroes, the number of operations is reduced.
- No improvement is achieved if ones and zeroes alternate

Modified Booth : The pattern $1\underline{1} \rightarrow 01$ is recoded

Radix-4 recoding

Allows processing two dígits per cycle.

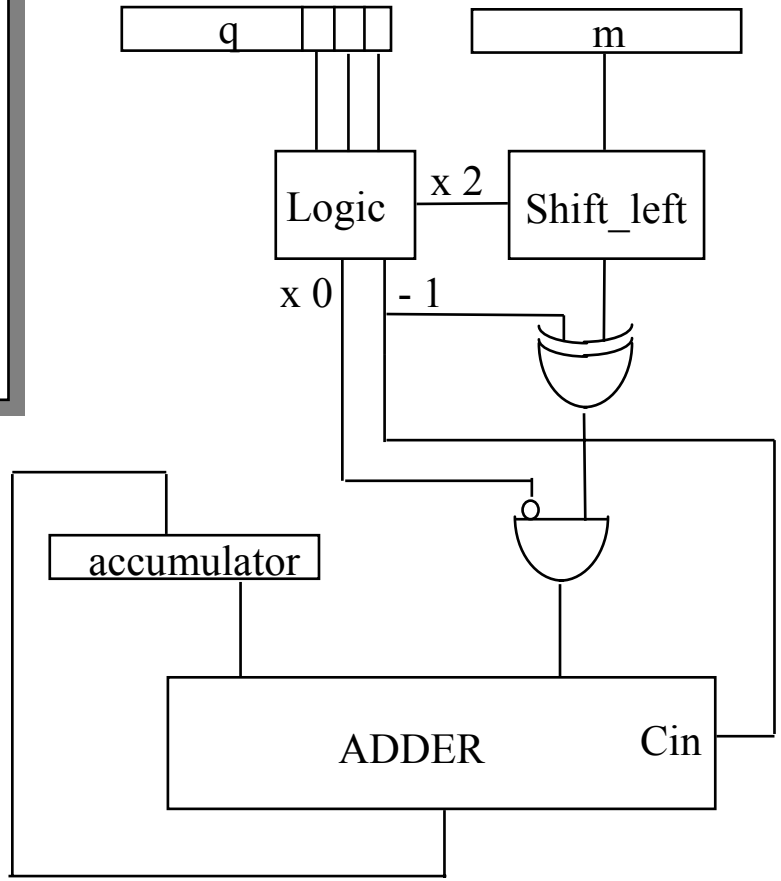
Multiplier	Booth code	Digit	Multiplicand op.
0 0 0	00	0	----------
0 0 1	01	1	Add
0 1 0	11	1	Add
0 1 1	10	2	Shift & add
1 0 0	10	2	Shift & sub
1 0 1	11	1	Sub
1 1 0	01	1	Sub
1 1 1	00	0	----------

Evaluated digits

Only one operation is needed for each two digits.

A combinacional shifter is required. Multiplier bits are shifted twice per clock.

Functional diagram:



Serial/parallel multiplier

Uses flip-flops as multiplicand delay elements, that is serially-input.

AND gates outputs: Same weight partial products. The partial products are accumulated by the carry-save adder tree.

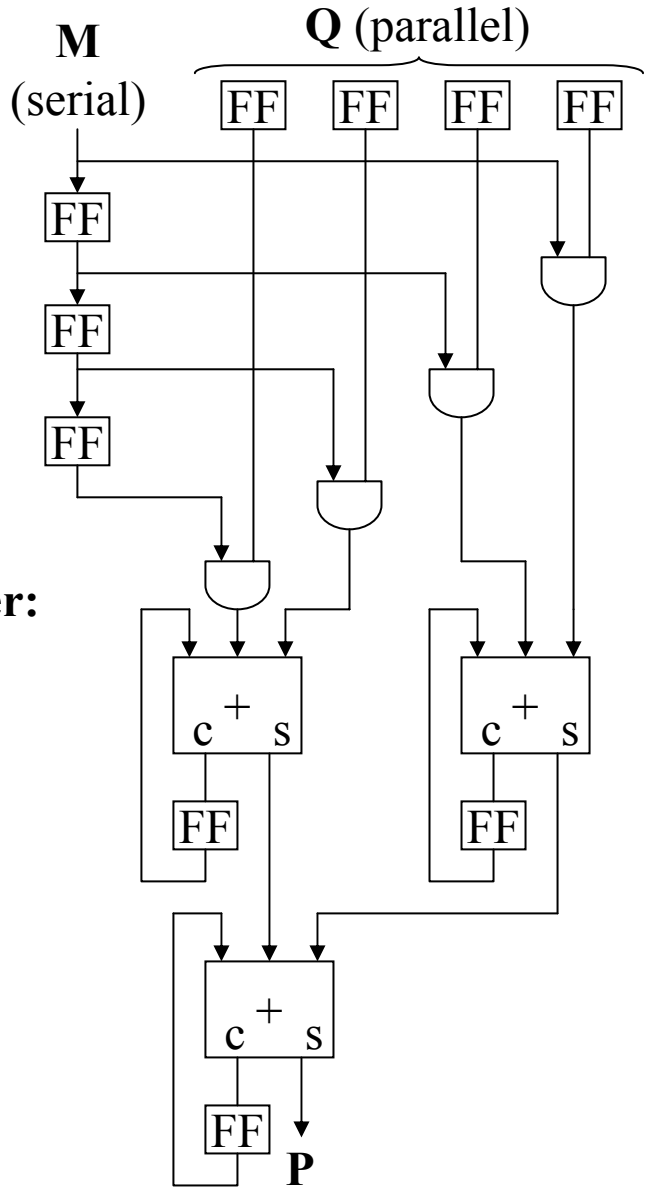
Serial output:

m_0q_0, $m_0q_1+m_1q_0$, $m_0q_2+m_1q_1+m_2q_0$, ...

Initial reset is required for all carry and multiplicand flip-flops.

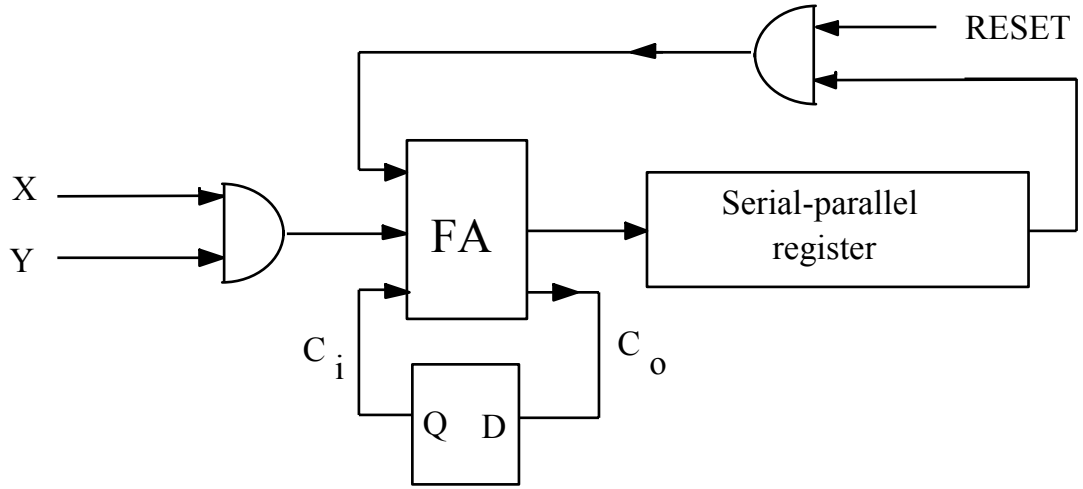
Number of cycles : 2n

4×4 Multiplier:



Serial/serial multiplier

- Simple hardware
- Very slow: n^2



Summary of multiplier first-order characteristics

- T, A: Full-adder unit delay and area.
- N: Multiplier number of bits.
- Δ includes *setup* time and flip-flop delay in sequential multipliers.
- Control circuits can increase area significantly, especially in serial-serial multiplier.

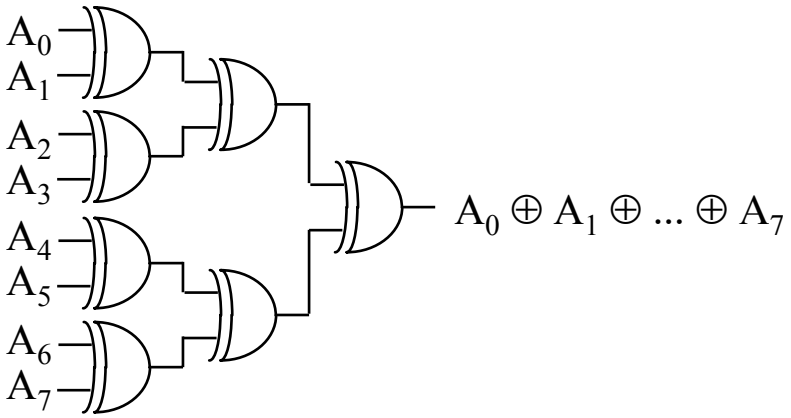
Multiplier	Clock cycles	t_d/cycle	Total delay	Area
Parallel- Parallel (array)	1	(3n-2)T	(3n-2)T + Δ	n^2A
Parallel- Parallel (carry-save)	1	2nT	2nT + Δ	n(n+1)A
Parallel -Serial	n	nT + Δ	n (nT + Δ)	$nA + control_{PS}$
Parallel –Serial (radix-4)	n/2	nT + Δ	n (nT + Δ)/2	$nA + control_{PSR4}$
Serial - Parallel	2n	$(\log_2 n)T + \Delta$	$2n[(\log_2 n)T + \Delta]$	$nA + control_{SP}$
Serial - Serial	n^2	T + Δ	$n^2(T + \Delta)$	$1 + control_{SS}$

4.4 Other operators

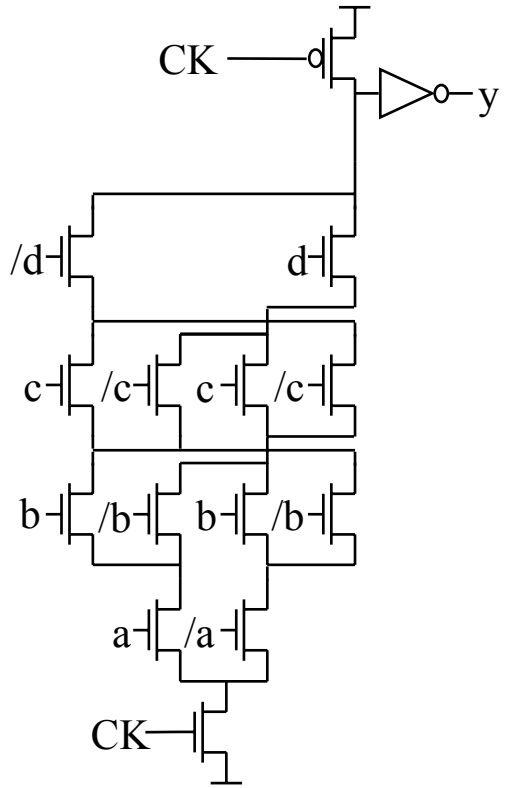
Parity generators

$$P(n) = A_0 \oplus A_1 \oplus \cdots \oplus A_{n-1}$$

Tree implementation

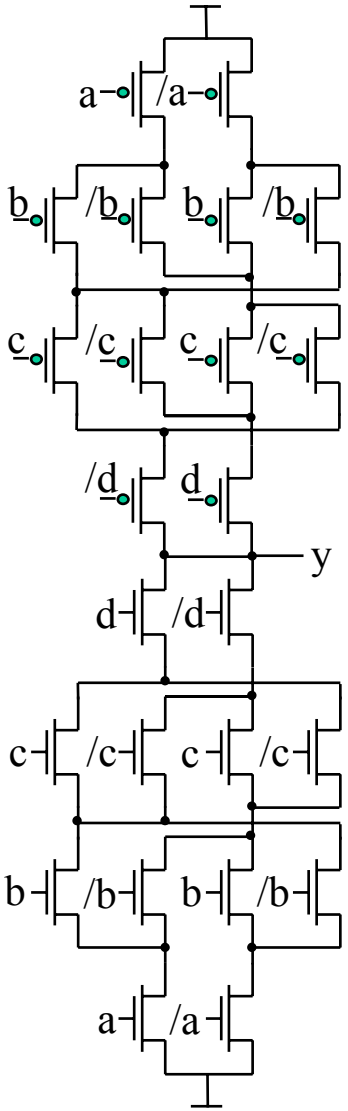


Transmission gate XOR: dynamic version



Static CMOS implementation

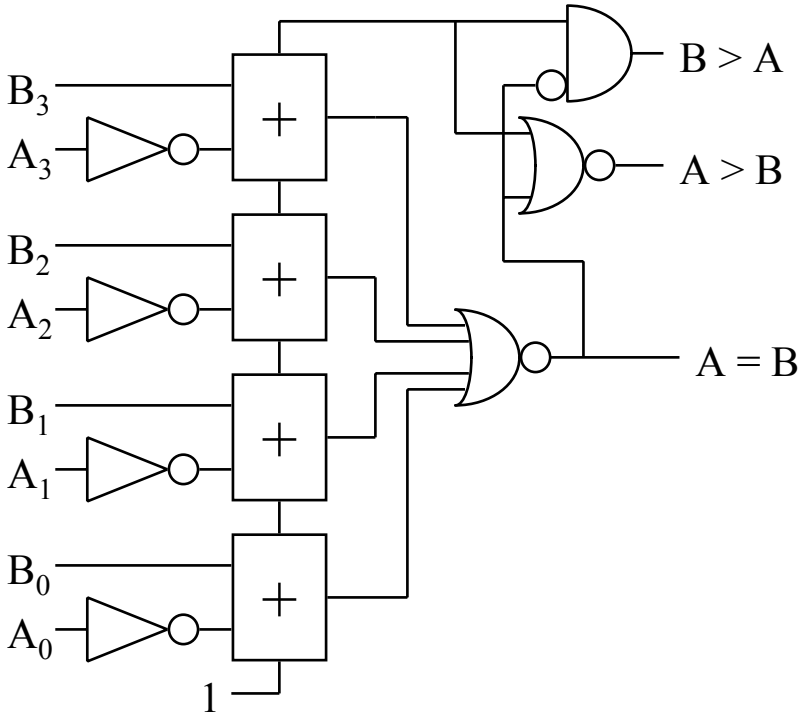
Complementary PMOS network is added.



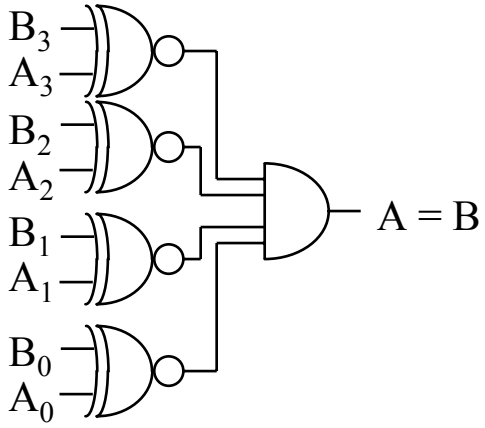
Comparators

Built with inverters and an adder.

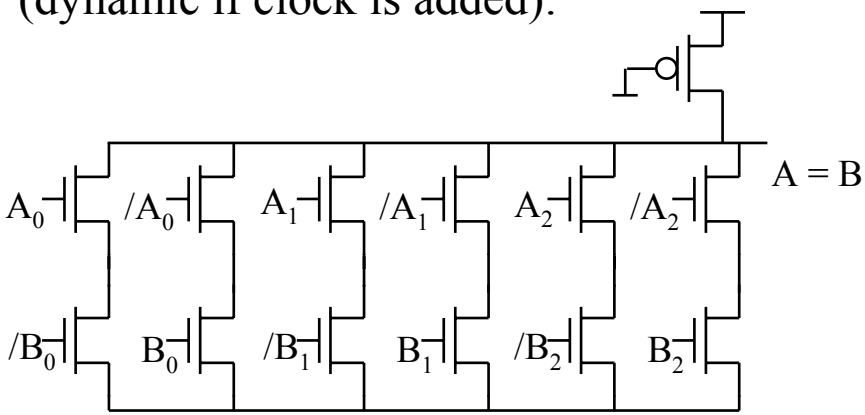
- Outputs NOR: A = B
- Carry: B > A (except A = B)
- None of the previous: A > B



Detection of only equality A = B is simpler:

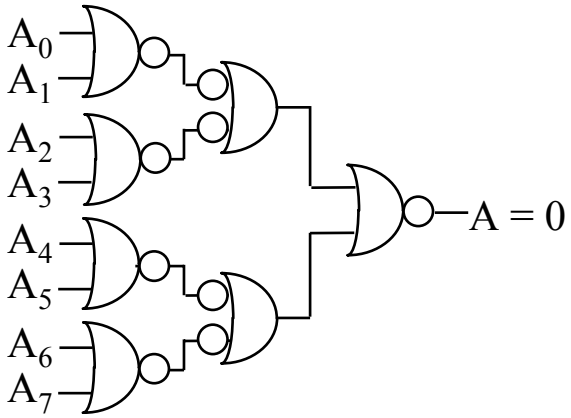


Pseudo-NMOS solution
(dynamic if clock is added):



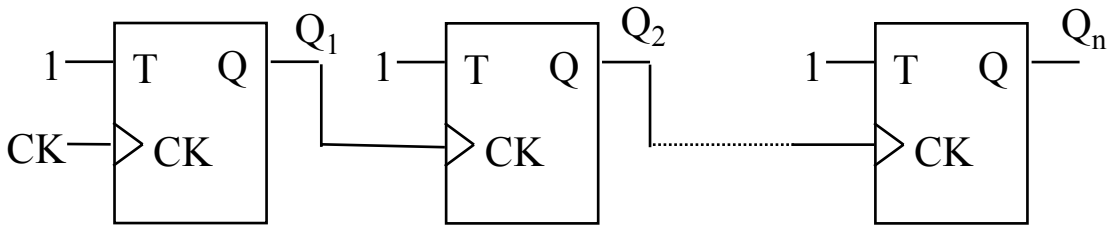
Detectors of 0 (1)

Gate trees AND (OR).
Delay proportional to log (N).



Counters

• Asynchronous
  Carry propagated.
  Based on registre T (single-stage counter).
  Clock period linearly increases with # of stages



• Synchronous: SBC