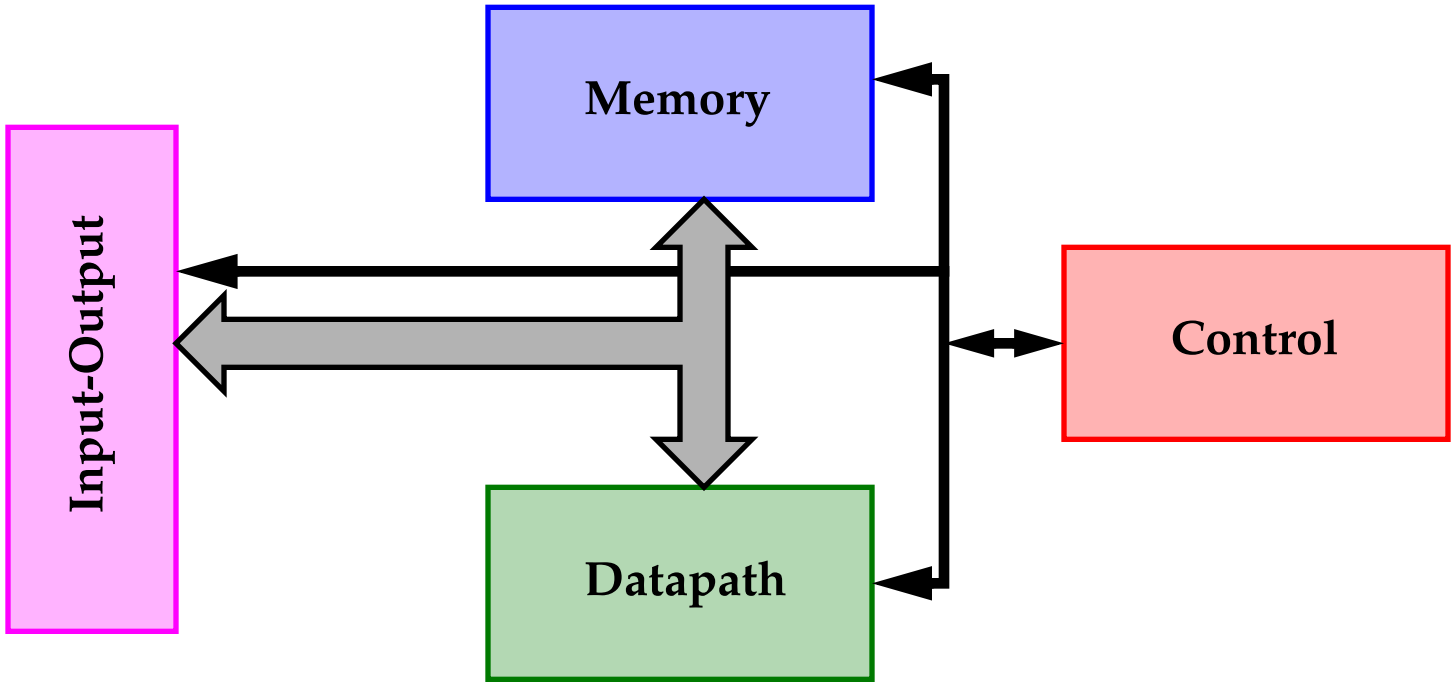
Digital Device Components

A simple processor illustrates many of the basic components used in any digital system:



- Datapath: The core -- all other components are support units that store either the results of the datapath or determine what happens in the next cycle.

Digital Device Components

- Memory:

  A broad range of classes exist determined by the way data is accessed:

  Read-Only vs. Read-Write

  Sequential vs. Random access

  Single-ported vs. Multi-ported access

  Or by their data retention characteristics:

  Dynamic vs. Static

  Stay tuned for a more extensive treatment of memories.

- Control:

  A FSM (sequential circuit) implemented using random logic, PLAs or memories.

- Interconnect and Input-Output:

  Parasitic resistance, capacitance and inductance affects performance of wires both on and off the chip.

  Growing die size increases the length of the on-chip interconnect, increasing the value of the parasitics.

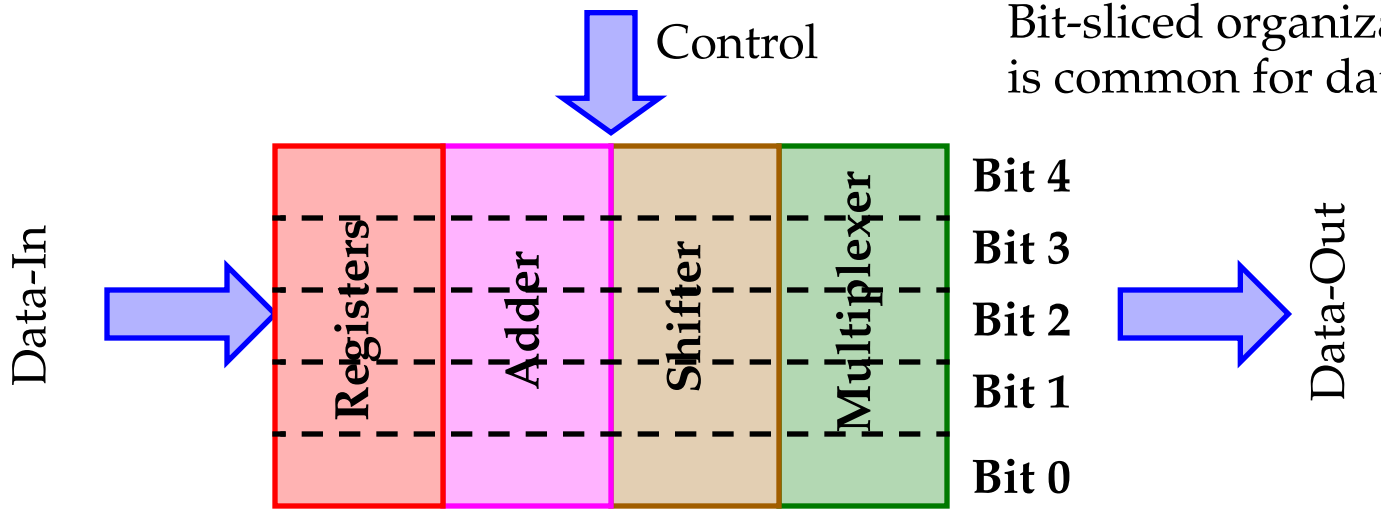
Digital Device Components

Datapath elements include adders, multipliers, shifters, BFUs, etc.

The speed of these elements often dominates the overall system performance so optimization techniques are important.

However, as we will see, the task is non-trivial since there are multiple equivalent logic and circuit topologies to choose from, each with adv./disadv. in terms of speed, power and area.

Also, optimizations focused at one design level, e.g., sizing transistors, leads to inferior designs.



Datapath Operators: Addition/Subtraction

Let's start with addition, since it is a very common datapath element and often a speed-limiting element.

Optimizations can be applied at the logic or circuit level.

- Logic-level optimization try to rearrange the Boolean equations to produce a faster or smaller circuit, e.g. carry look-ahead adder.
- Circuit-level optimizations manipulate transistor sizes and circuit topology to optimize speed.

Let's start with some basic definitions before considering optimizations:

A	B	C_i	G(A.B)	P(A+B)	P'(A⊕B)	Sum	C_o	Carry status
0	0	0	0	0	0	0	0	delete
0	0	1	0	0	0	**1**	0	delete
0	1	0	0	1	1	**1**	0	propagate
0	1	1	0	1	1	0	**1**	propagate
1	0	0	0	1	1	**1**	0	propagate
1	0	1	0	1	1	0	**1**	propagate
1	1	0	1	1	0	0	**1**	generate
1	1	1	1	1	0	**1**	**1**	generate

Datapath Operators: Addition/Subtraction

G(A.B): (generate)

Occurs when a C_o is internally generated within the adder (occurs independent of C_i).

P(A+B): (propagate)

Indicates that C_i is *propagated* (passed) to C_o.

P'(A XOR B): (propagate)

Used in some adders for the P term since it can be reused to generate the sum term.

D($\overline{A}.\overline{B}$): (delete)

Ensures that a carry bit will be deleted at C_o.

The Boolean expressions for S and C_o are:

Sum = $A.B.C_i + A.\overline{B}.\overline{C}_i + \overline{A}.\overline{B}.C_i + \overline{A}.B.\overline{C}_i$ = A **XOR** B **XOR** C

Carry = $A.B + A.C_i + B.C_i$

Datapath Operators: Addition/Subtraction

But S and C_o can be written in terms of G and P′:

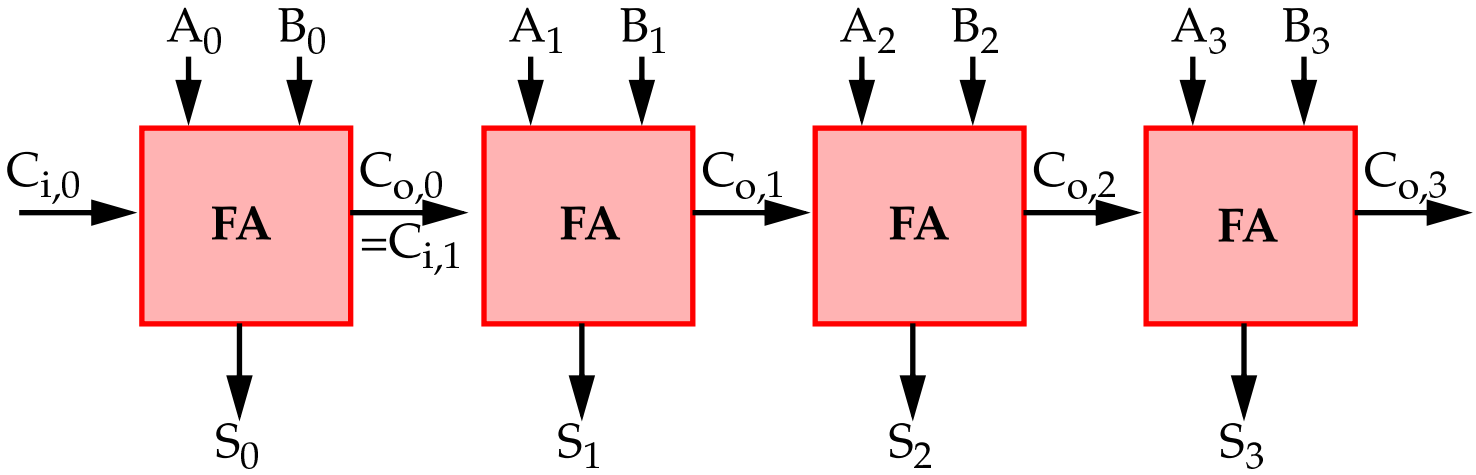
$C_o(G, P') = G + P'C_i$ (or P in this case).

$S(G, P') = P'$ XOR C_i

Note that G and P′ are INdependent of C_i.

(Also, C_o and S can be expressed in terms of delete (D)).

Ripple-carry adder:



The **critical path** (worst case delay over all possible inputs) is a ripple from *lsb* to *msb*.

Datapath Operators: Addition/Subtraction

The delay in this case is proportional to the number of bits, N, in the input words:

$$t_{adder} = (N - 1)t_{carry} + t_{sum}$$

where t_{carry} and t_{sum} are the propagation delays from C_i to C_o & S.

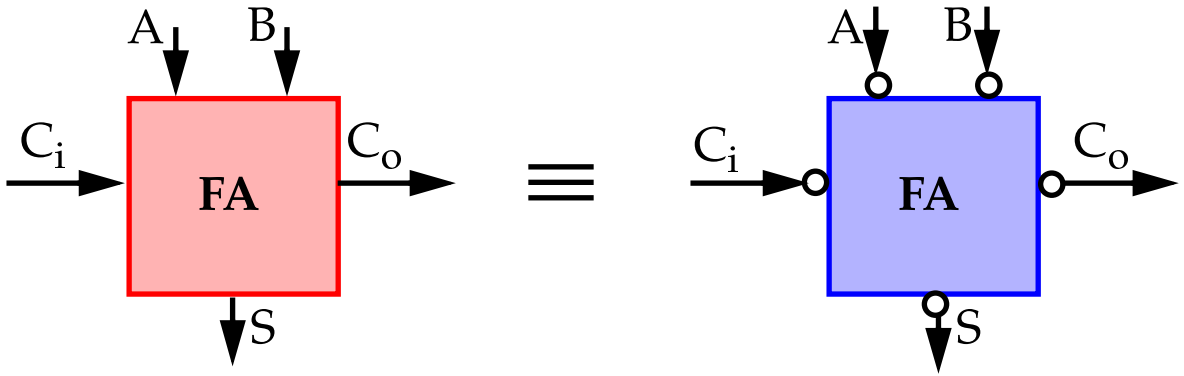
One possible worst case bit pattern (from *lsb* to *msb*) is:

A: 00000001; B: 01111111

Convince yourself that this is true.

Note that when optimizing this structure, it is far more important to optimize t_{carry} than t_{sum}.

The inverting property of a full adder can be used to achieve this goal:



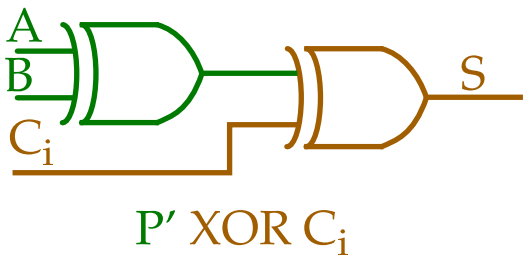
Datapath Operators: Addition/Subtraction

Thus,

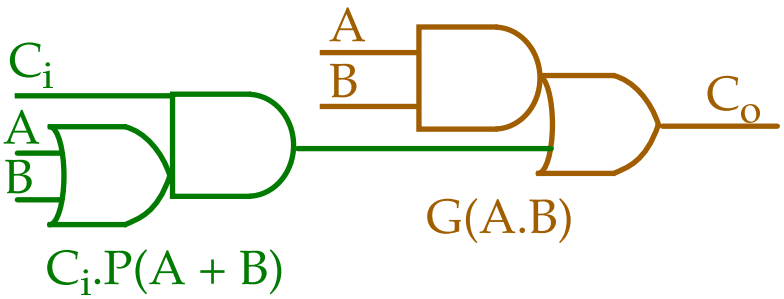
$\overline{S}(A, B, C_i) = S(\overline{A}, \overline{B}, \overline{C_i})$

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

One possible (un-optimized) implementation:



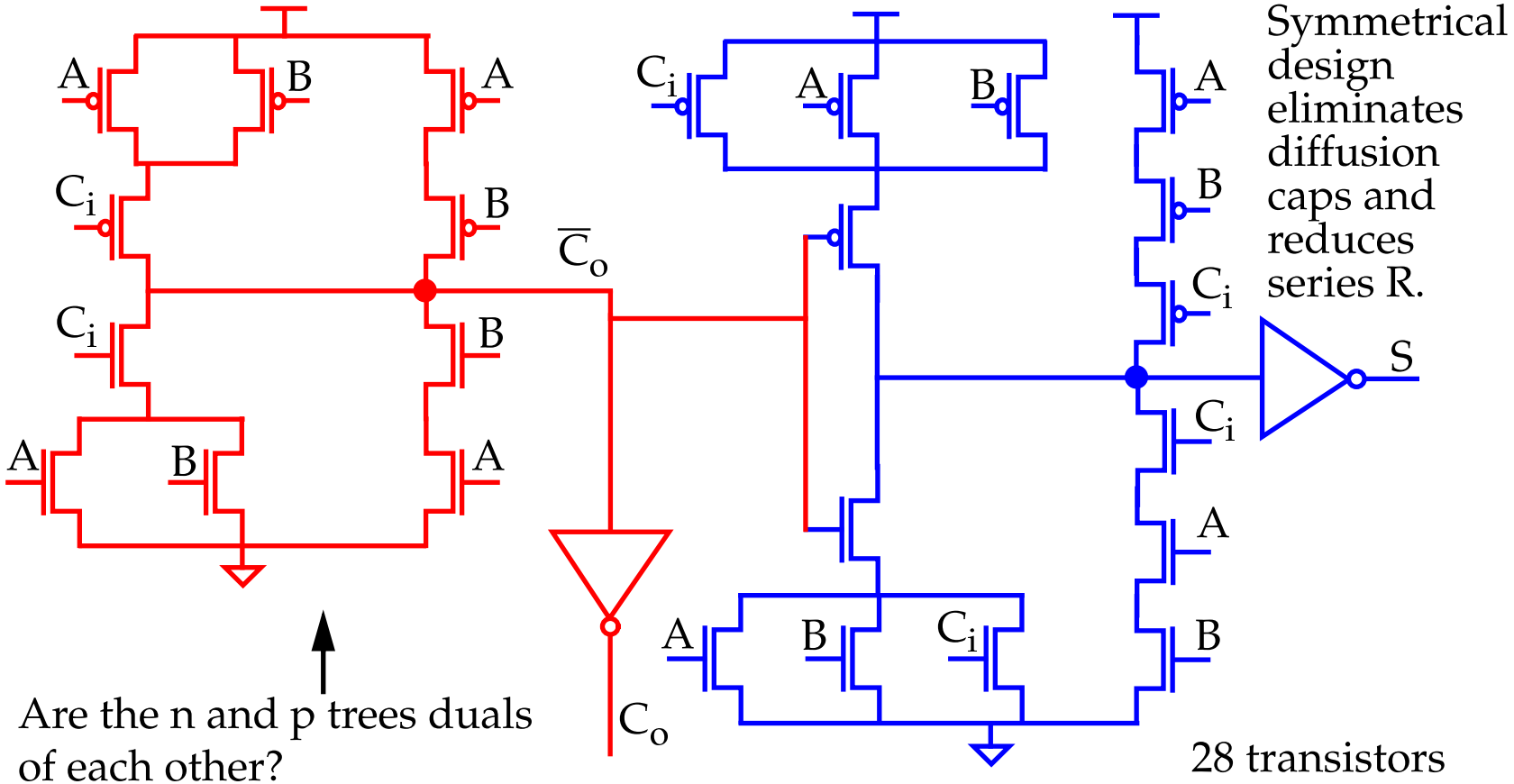
Transistor level diagram uses 32 transistors.



Datapath Operators: Addition/Subtraction

C_o is reused in the S term as:

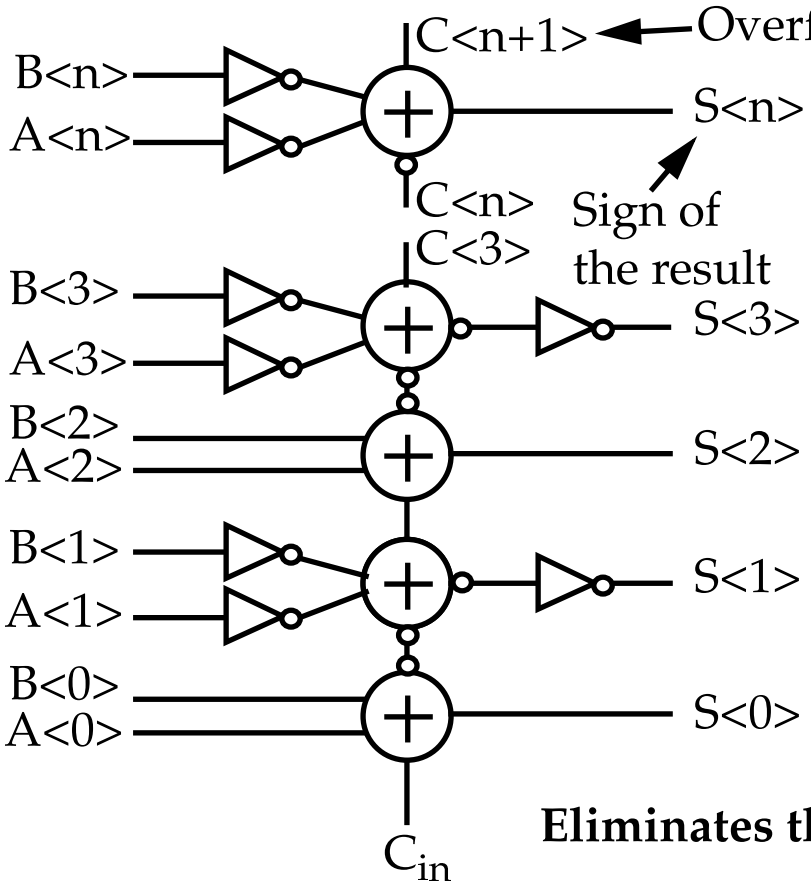
$$Sum = A.B.C_i + (A + B + C_i)\overline{C}_o$$

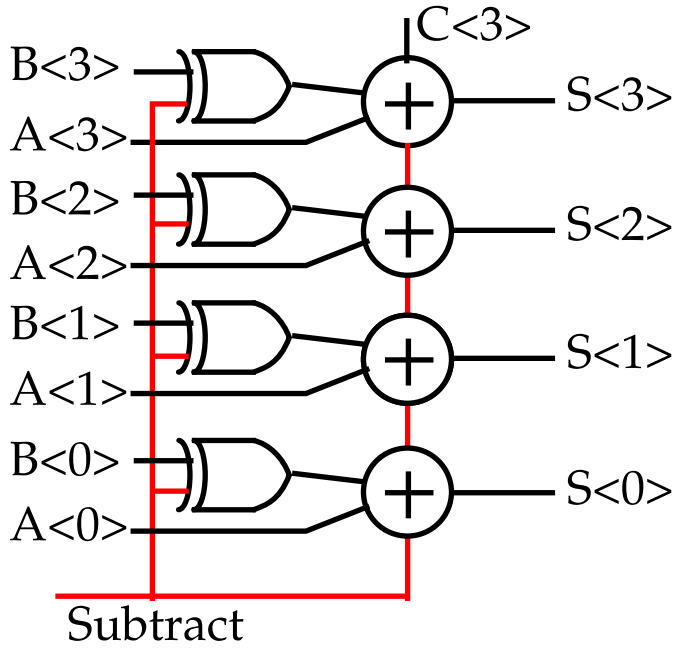


Even with some design tricks, e.g., transistors on the critical path, C_i placed closest to the output and symmetrical design, this implementation is slow.

Datapath Operators: Addition/Subtraction

The load capacitance in previous version on C_o consists of 2 diffusion capacitances (inverter) and 6 (next bit) gate capacitances:



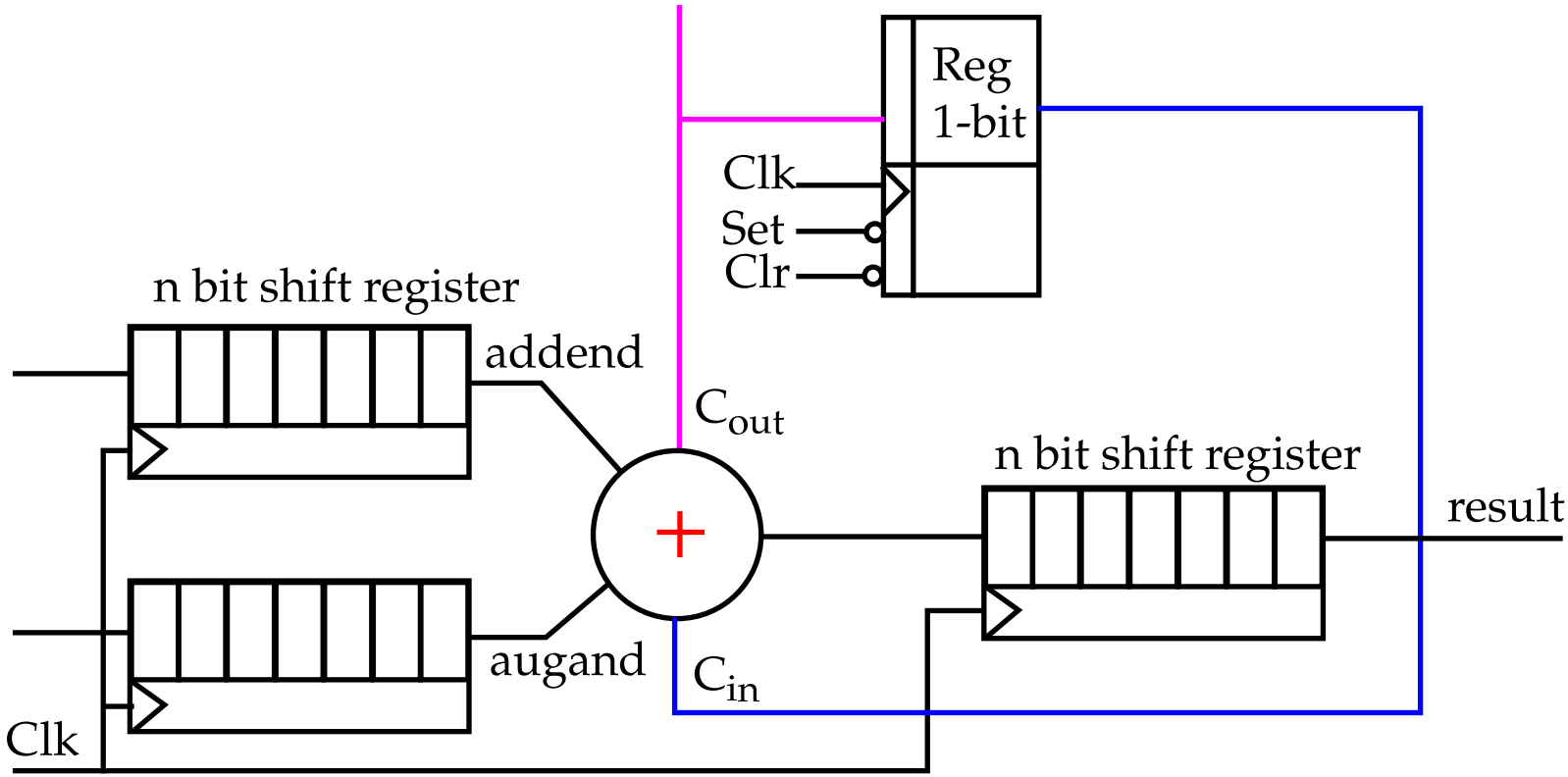


Eliminates the inverter delay per bit for carry!

This version increases $\overline{C}_o$'s load to 4 diffusion caps, 2 internal (sum) gate caps plus the 6 (next bit) gate caps.

Datapath Operators: Addition/Subtraction

Serial addition can be used if area is a concern:

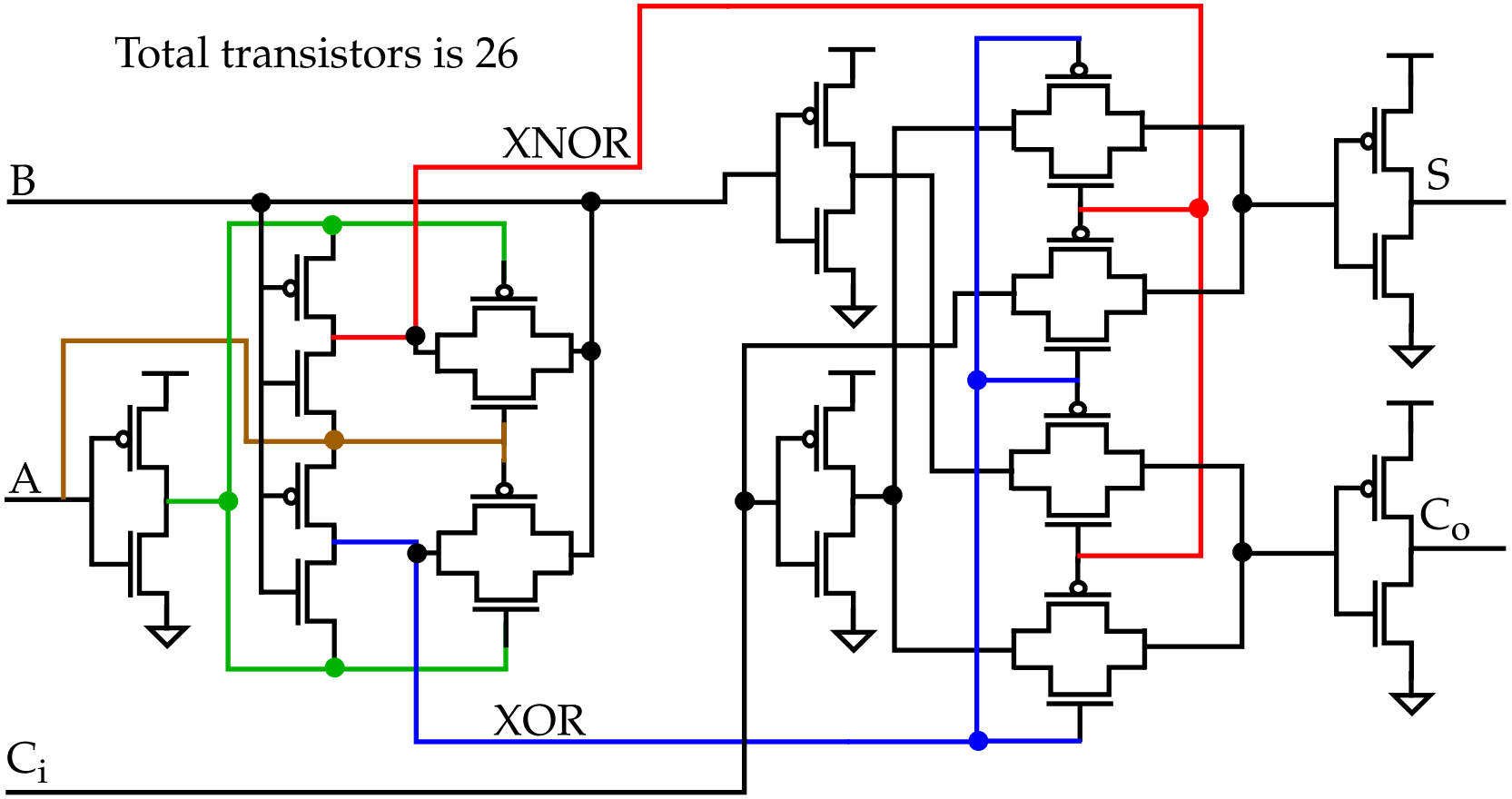


In this case, you want equal Sum and Carry delays in order to minimize clock cycle time.

Bit-level pipelining can be used to break the dependency between addition time and the number of bits by inserting FAs between each register bit.

Datapath Operators: Addition/Subtraction

Transmission-gate Adder:

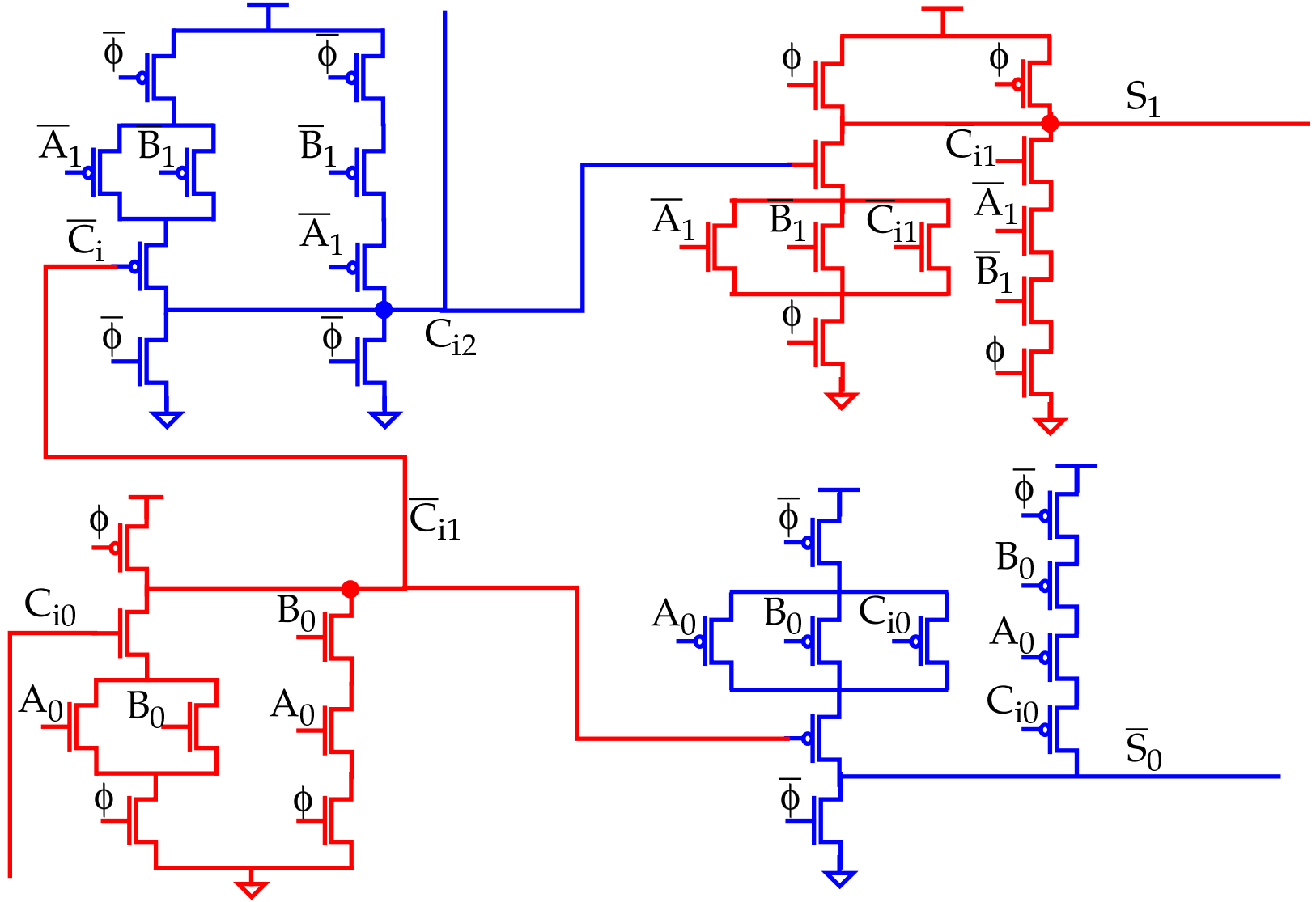


Note: S and C_o delay times are approximately equal -- good for multipliers.

See Weste and Eshraghian for an 18 transistor implementation.

Datapath Operators: Addition/Subtraction

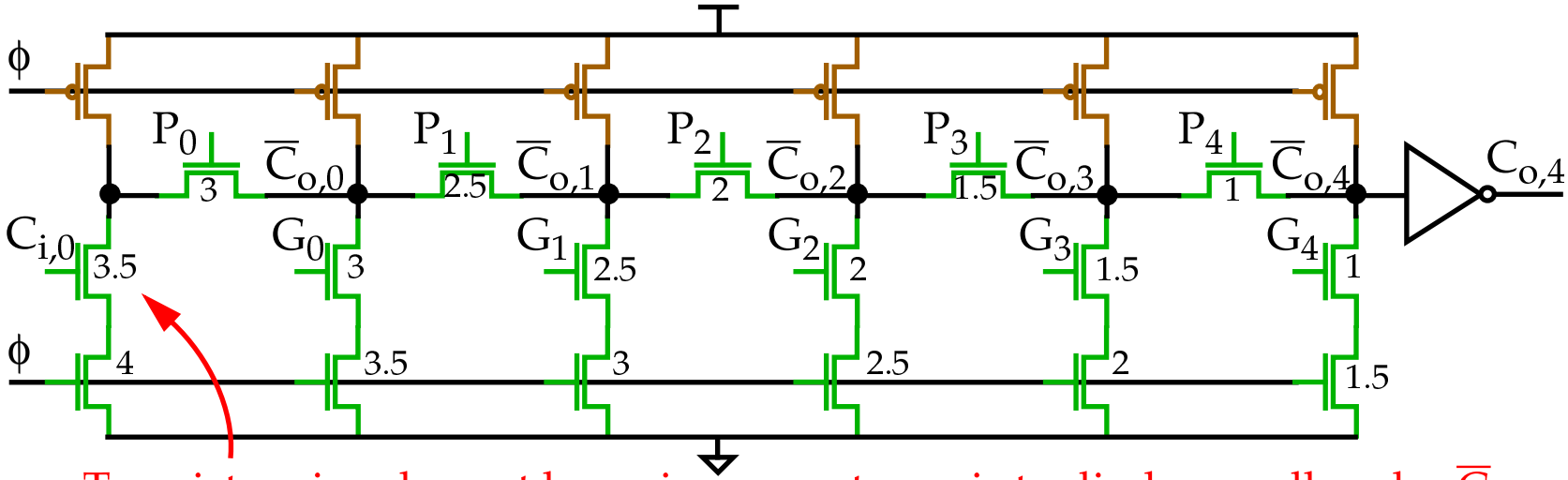
Dynamic Adder Design: *np-CMOS* adder



Datapath Operators: Addition/Subtraction

Dynamic Adder Design: *Manchester Carry-Chain* adder.

A chain of pass-transistors are used to implement the carry chain.



Transistor sizes largest here since worst case is to discharge all nodes $\overline{C}_{o,k}$.

Precharge: All intermediate nodes, e.g. $\overline{C}_{o,0}$, charged to V_{DD}.

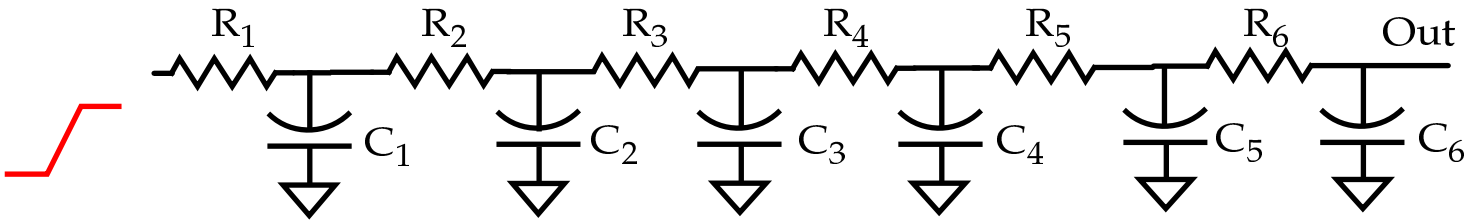
Evaluate: Node $\overline{C}_{o,k}$ is discharged, for example, if there is an incoming carry, $C_{i,0}$ and the previous propagate signals are high, P_0 to P_{k-1}.

Only 4 diffusion capacitances are present per node but the distributed RC-nature of the chain results in delay that is quadratic with number of bits.

Buffers and/or transistor sizing can be used to improve performance.

Datapath Operators: Addition/Subtraction

Consider the worst case delay of the carry chain:



Elmore delay is given by:

$$t_p = 0.69\left(\sum_{i=1}^{N} C_i\right)\left(\sum_{j=1}^{i} R_j\right)$$

The delay of the RC network is then:

$t_p = 0.69(C_1R_1 + C_2(R_1 + R_2) + C_3(R_1 + R_2 + R_3) + C_4(R_1 + R_2 + R_3 + R_4) +$
$C_5(R_1 + R_2 + R_3 + R_4 + R_5) + C_6(R_1 + R_2 + R_3 + R_4 + R_5 + R_6)$

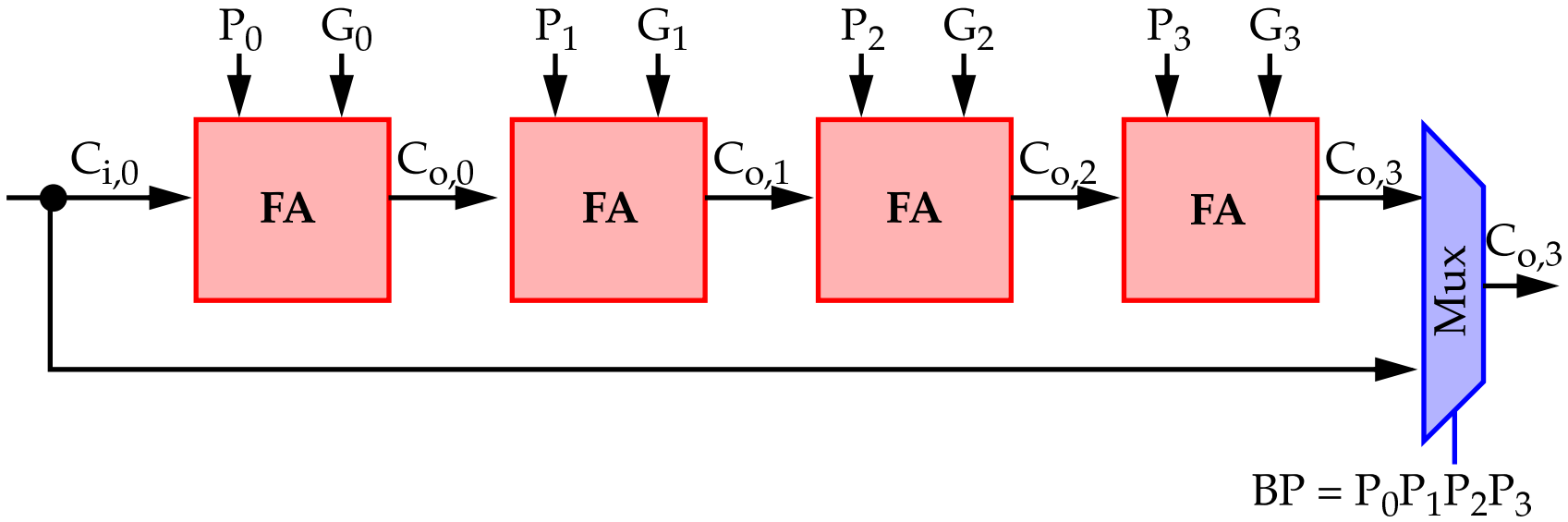
Since R_1 appears **6 times** in the expression, it makes sense to minimize its contribution.

Note that reducing R by a factor, e.g. *k*, at each stage increases the capacitance by a factor *k* and increases area.

A *k-factor* of 1.5, reduces delay by 40% and increases area by 3.5X.

Datapath Operators: Addition/Subtraction

Carry-Bypass adder:



Assume A_k and B_k (for k = 1...3) are set such that all P_k (propagate) are high.

In this case, an incoming carry $C_{i,0} = 1$, propagates along the complete chain and $C_{o,3} = 1$.

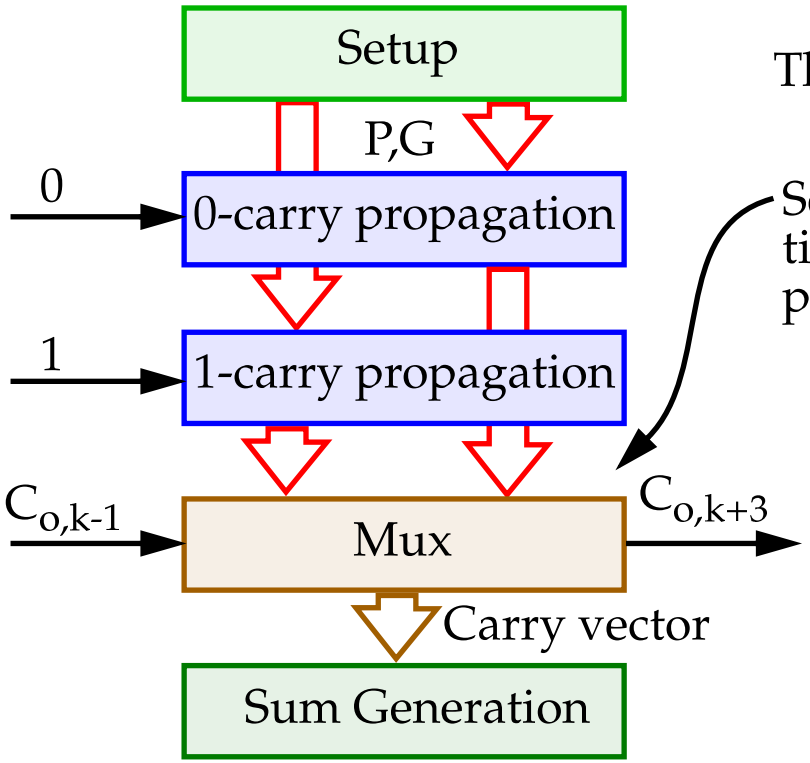
In other words:

if ($P_0P_1P_2P_3$ == 1) then $C_{o,3} = C_{i,0}$ else either DELETE or GENERATE occurred.

Datapath Operators: Addition/Subtraction

Linear Carry-Select adder:

One way around waiting for the incoming carry is to compute the result of **both** possible values in advance and let the incoming carry **select** the correct result.



This block adds bits k to k+3.

Select operation is much faster than time to compute either of the two possible carry vectors.

For Square-Root Carry-Select, higher order blocks take more operand bits than lower order blocks.

A **Square-Root Carry-Select** Adder (delay = $O(N^{1/2})$) is constructed by increasing the number of input bits in each block from *lsb* to *msb*.

Datapath Operators: Addition/Subtraction

Carry look-ahead adder (avoiding the ripple altogether):

Compute the carries to each stage in parallel.

The carry out of the k_{th} stage is computed as:

$$C_{o,k} = G_k + P_k \cdot C_{o,k-1} \quad \text{where} \quad G_k = A_k \cdot B_k$$
$$P_k = A_k + B_k$$

The dependency between $C_{o,k}$ and $C_{o,k-1}$ can be eliminated by expanding $C_{o,k-1}$.

$$C_{o,k} = G_k + P_k \cdot (G_{k-1} + P_{k-1}.C_{o,k-2})$$

For example, for 4 stages of look-ahead:

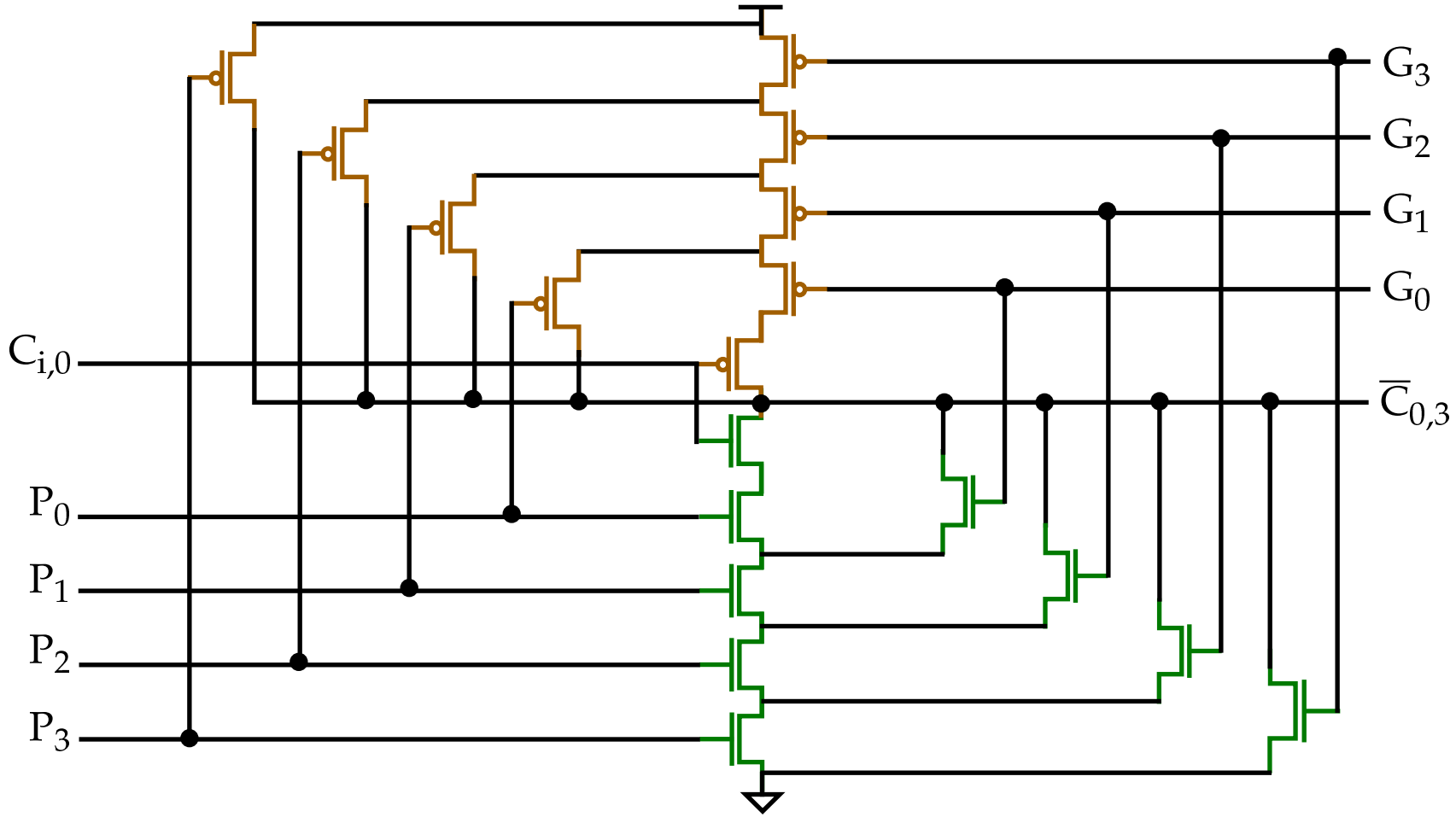
$$C_0 = G_0 + P_0C_i$$
$$C_1 = G_1 + P_1G_0 + P_1P_0C_i$$
$$C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_i$$
$$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_i$$

Note that the low-order terms, e.g., P_0 and G_0, appear in the expression for every bit, making the fanout load large.

Datapath Operators: Addition/Subtraction

Carry look-ahead adder:

One possible implementation without using simple logic gates.



Size and fan-in of the gates limit the size to about four.

Datapath Operators: Addition/Subtraction

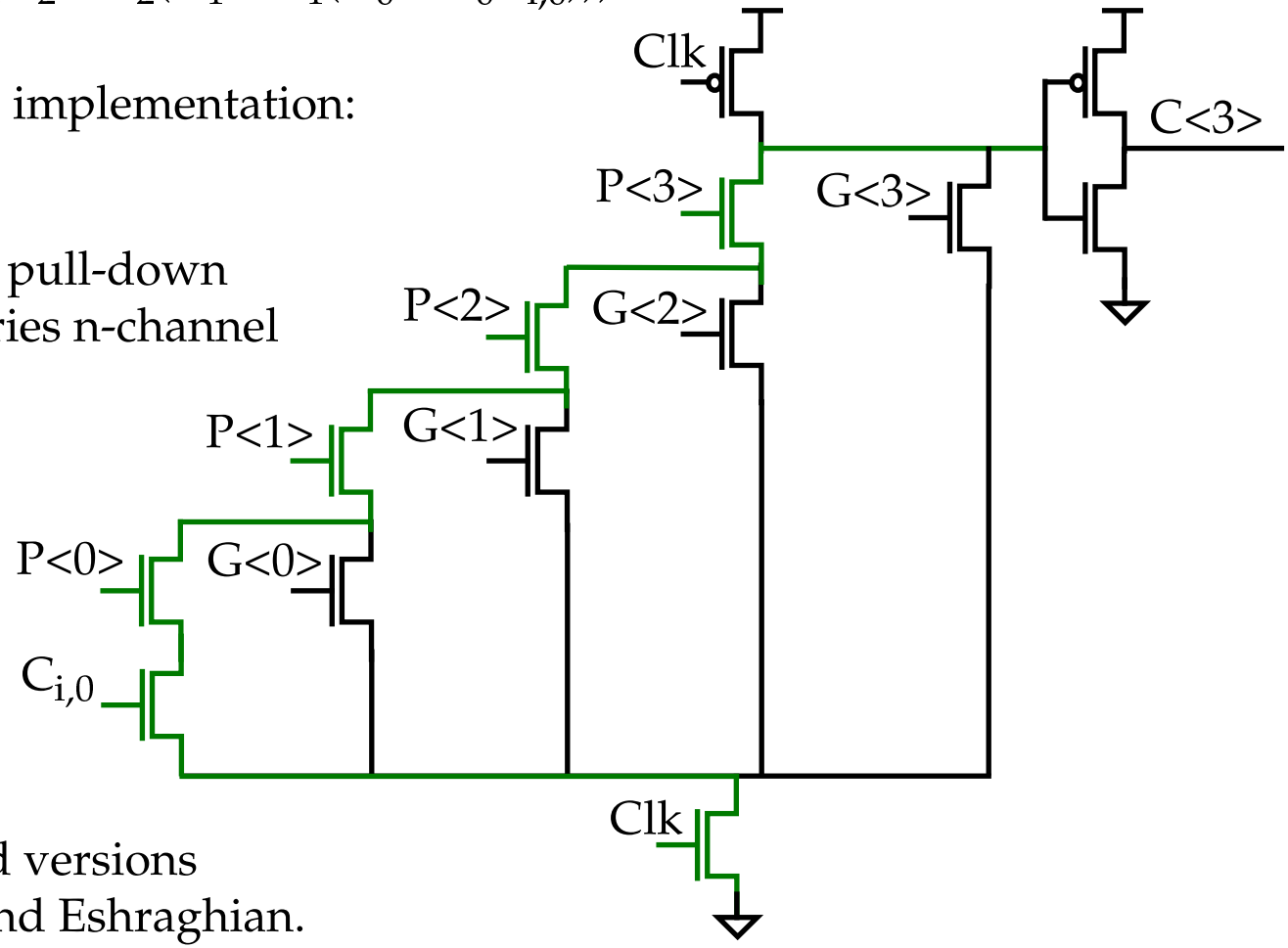
Carry look-ahead adder:

Factoring term C_3 yields:

$$C_3 = G_3 + P_3(G_2 + P_2(G_1 + P_1(G_0 + P_0C_{i,0})))$$

Domino CMOS implementation:

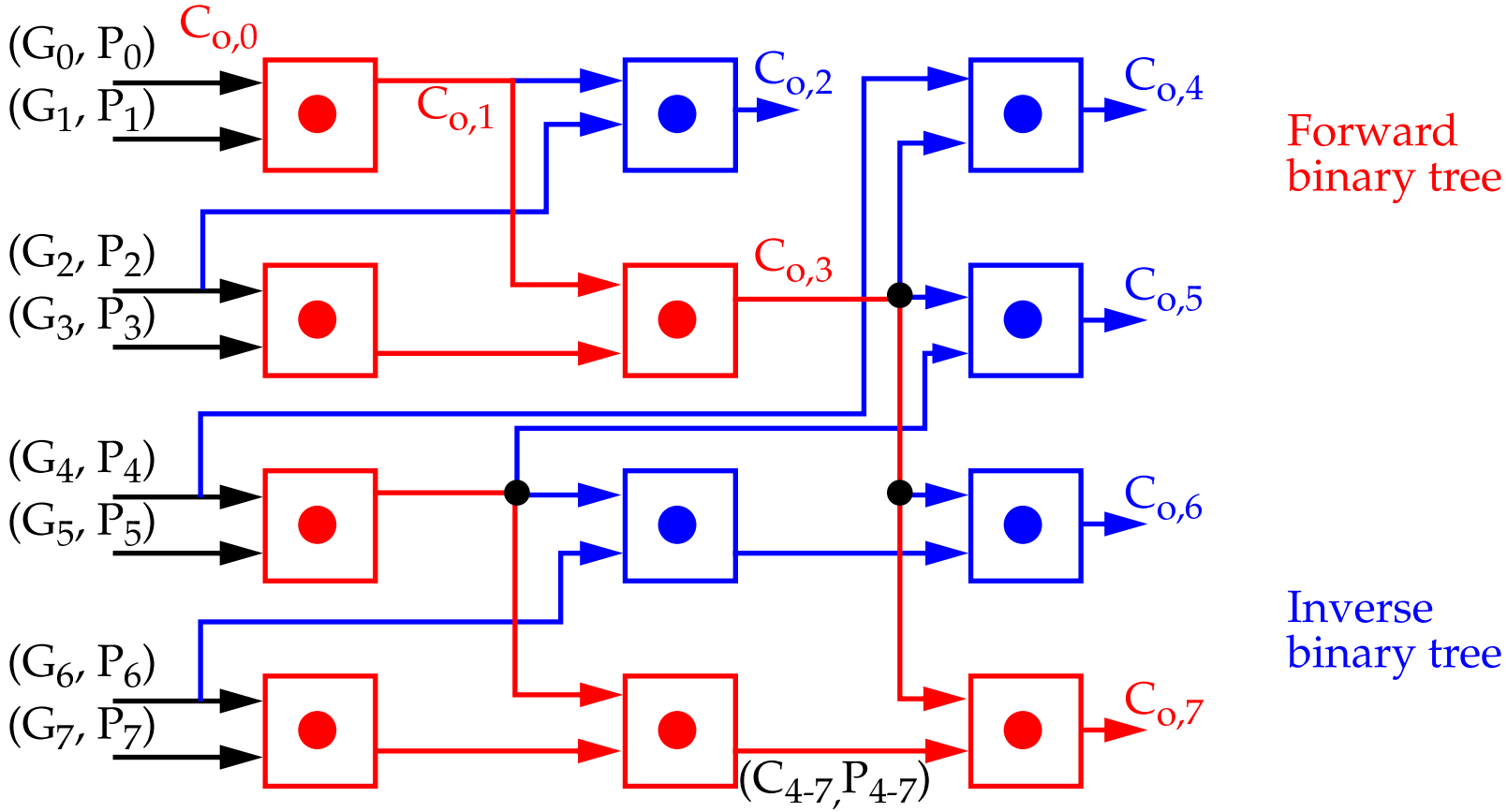
Worst case is pull-down through 6 series n-channel transistors.



Other high speed versions given in Weste and Eshraghian.

Datapath Operators: Addition/Subtraction

The *Logarithmic look-ahead* adder: O($\log_2$N) delay:



The dot operator (●)is defined as: (g, p) . (g′, p′) = (g + pg′, pp′)

The number of logic levels is proportional to $\log_2$N, fan-in is limited and the layout is compact (jigsaw puzzle) (see Rabaey for details).