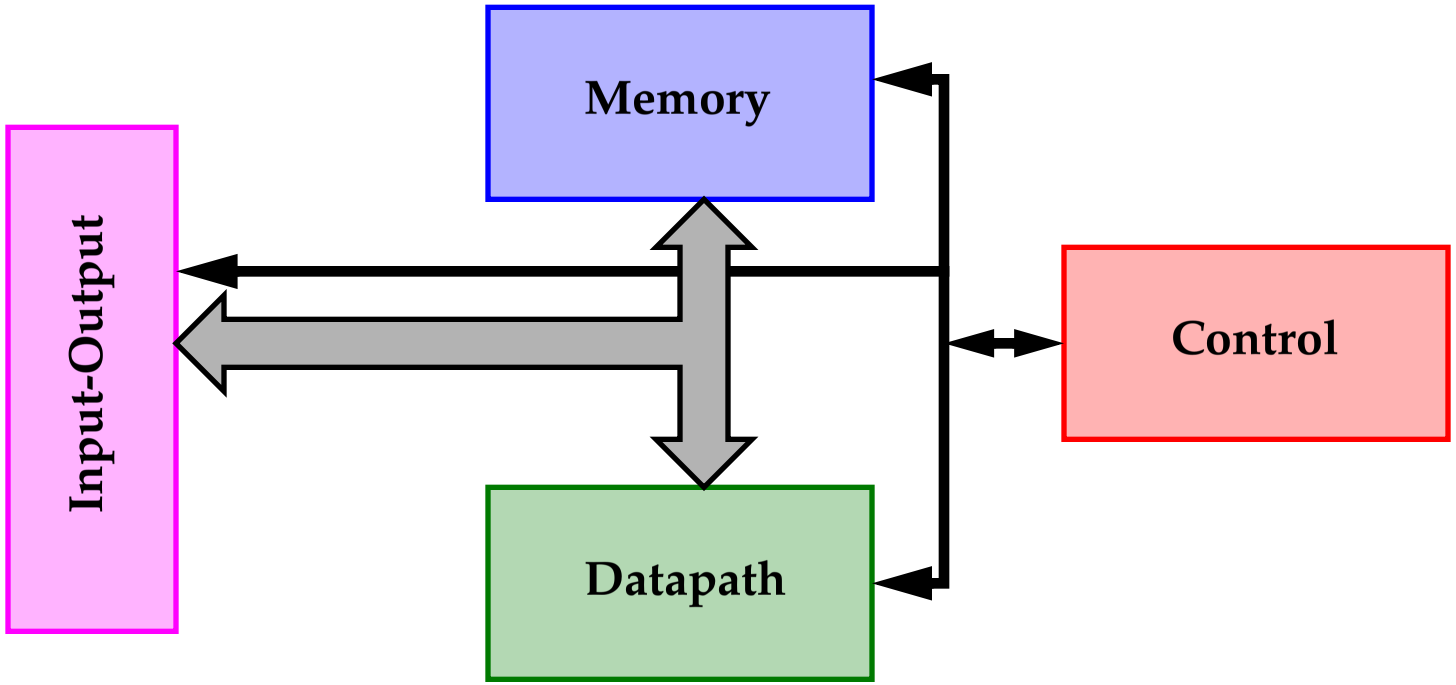
Digital Device Components

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.

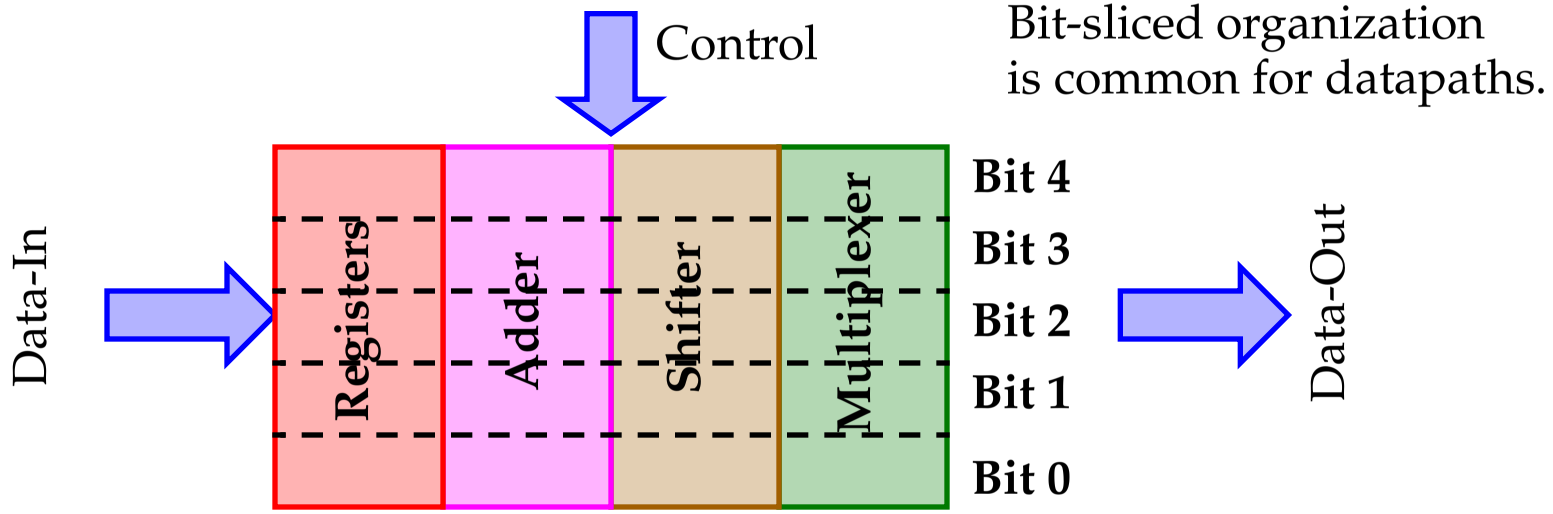
Digital Device Components

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′:

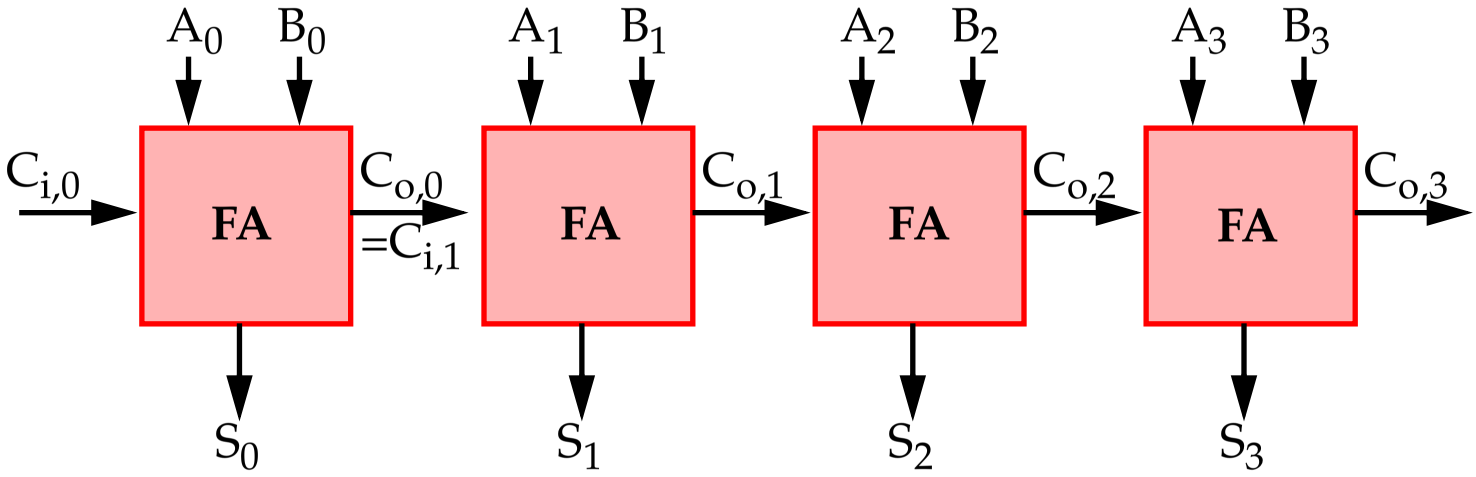
$C_o(G, P') = G + P'C_i$ (or P in this case).

$S(G, P') = P'$ 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.

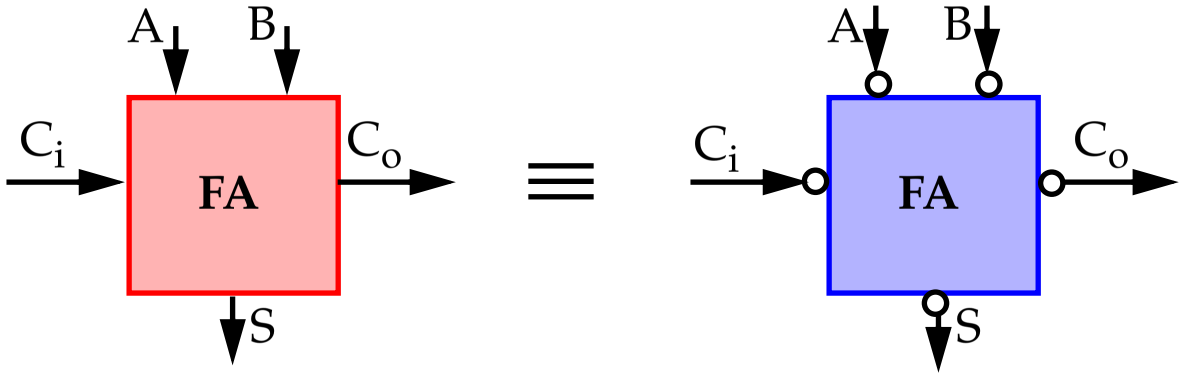
One possible worst case bit pattern (from *lsb* to *msb*) is:

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:



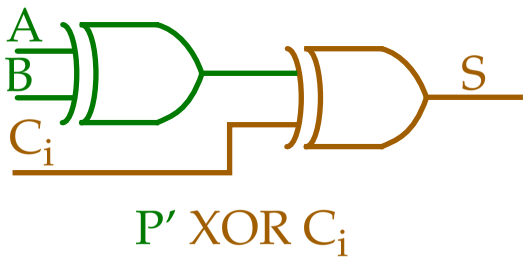
Datapath Operators: Addition/Subtraction

Thus,

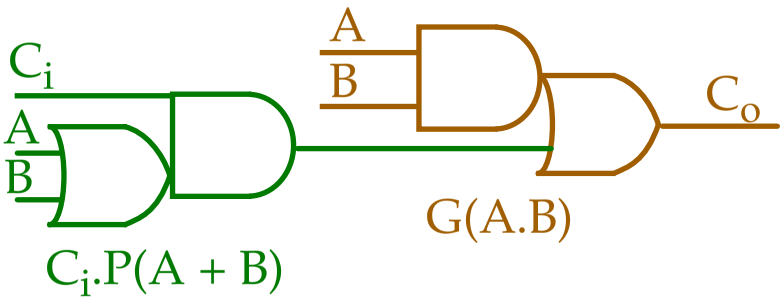
$\overline{S}(A, B, C_i) = S(\overline{A}, \overline{B}, \overline{C}_i)$

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

One possible (un-optimized) implementation:



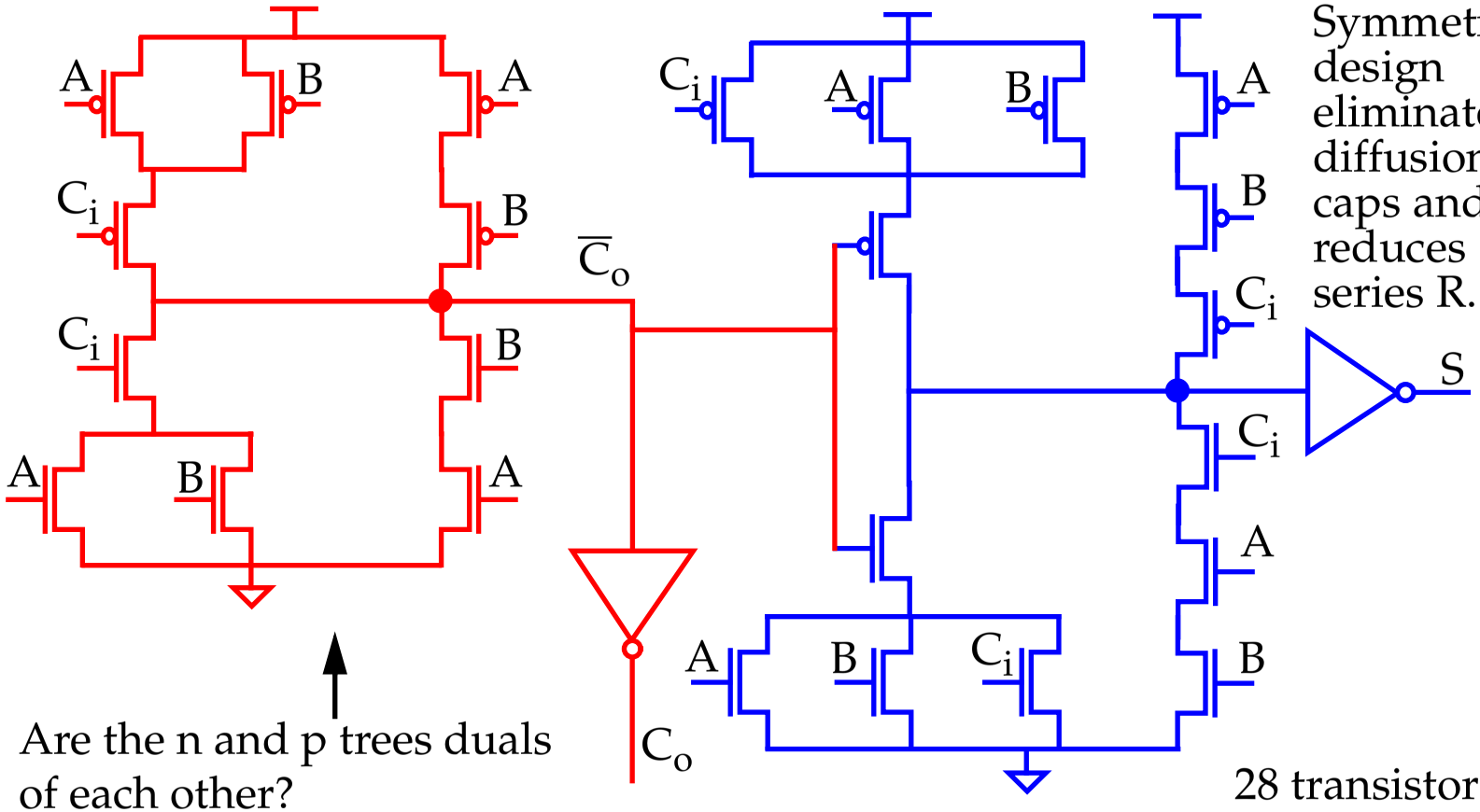
Transistor level diagram uses 32 transistors.
(see Weste and Eshraghian).



Datapath Operators: Addition/Subtraction

C_o is reused in the S term as:

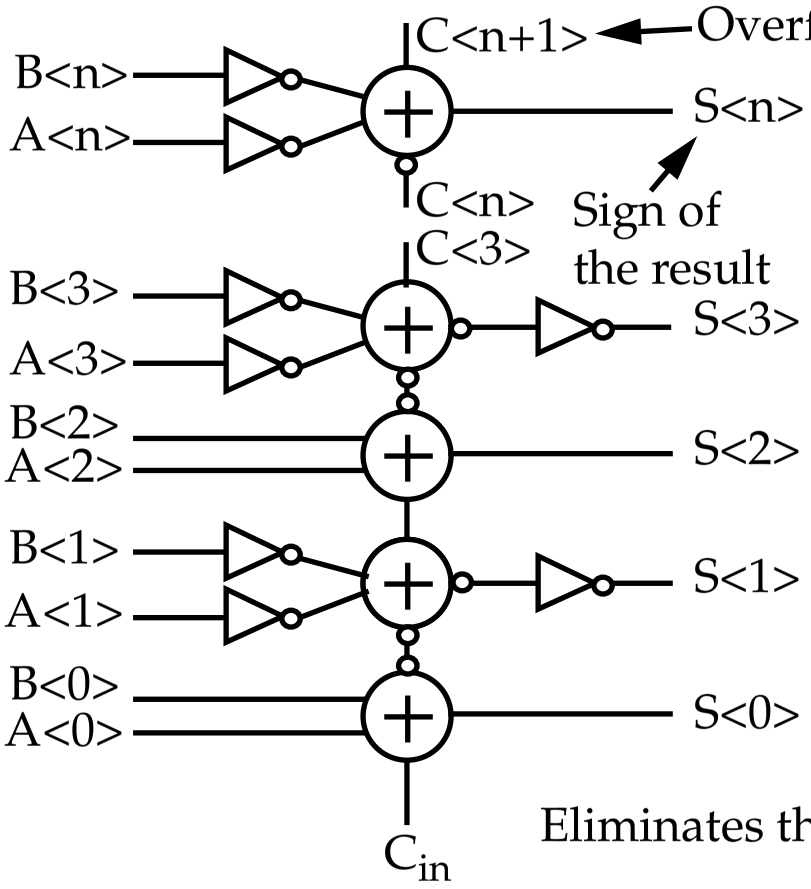
$$Sum = A.B.C_i + (A + B + C_i)\overline{C}_o$$

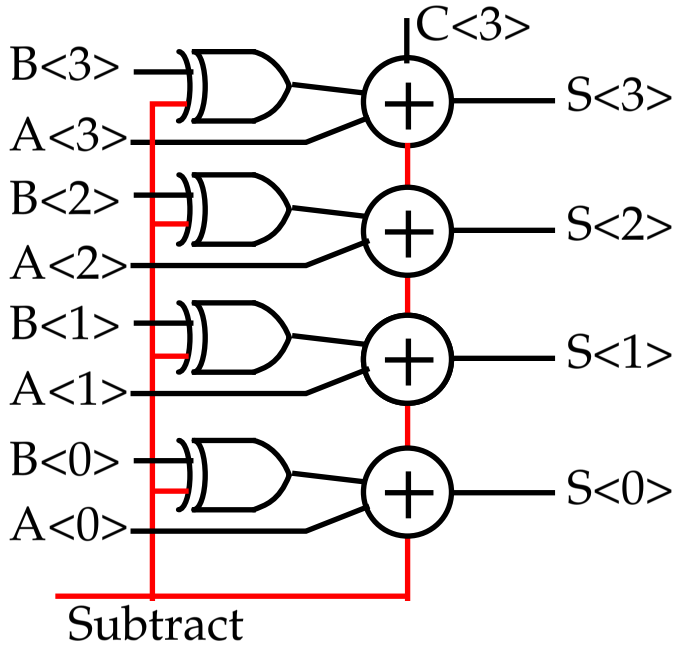


Even with some design tricks, e.g., transistors on the critical path, C_i placed closest to the output and symmetrical design, this implementation is slow.

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:



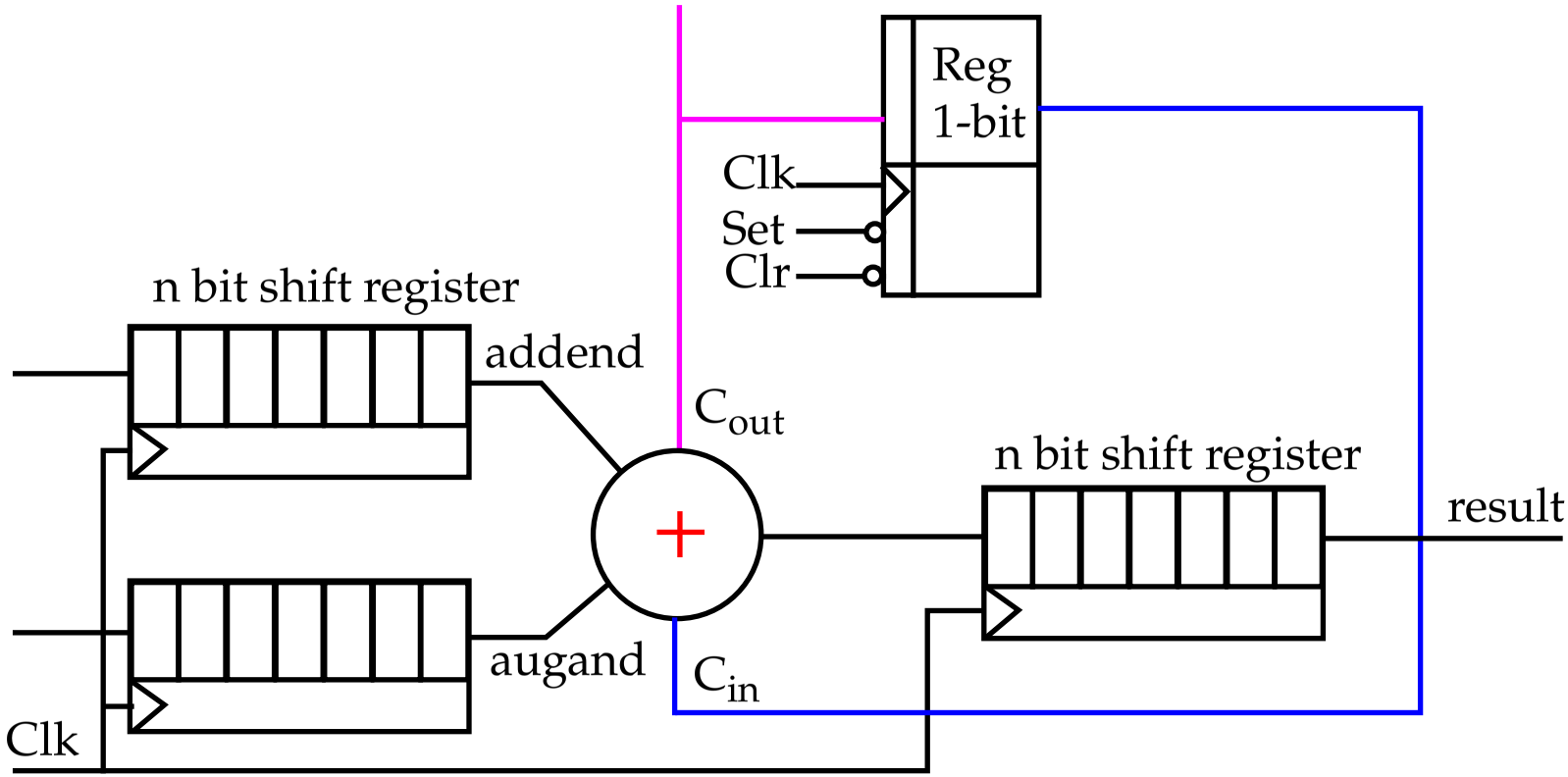


Eliminates the inverter delay per bit for carry!

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:

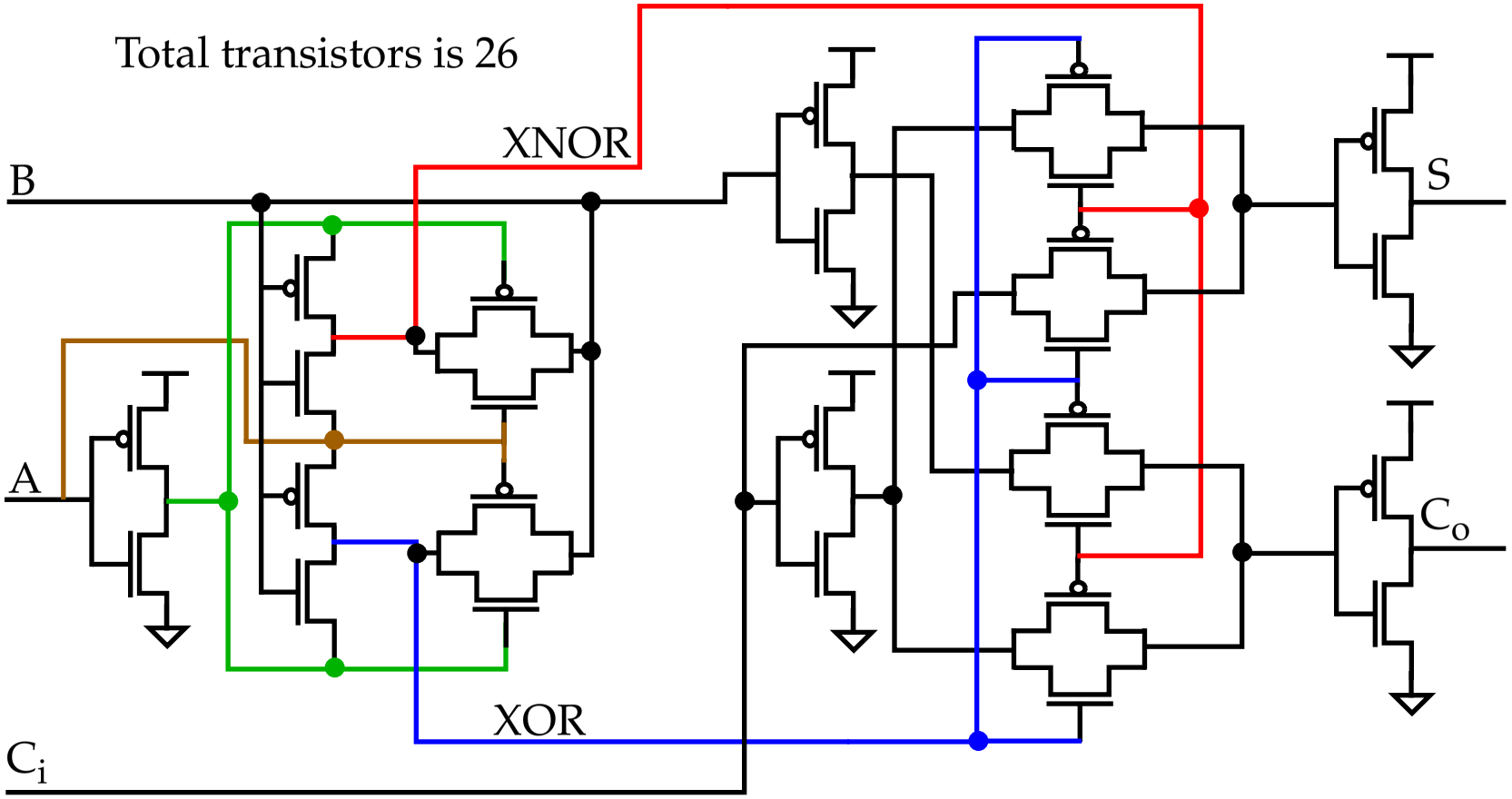


In this case, you want equal Sum and Carry delays in order to minimize clock cycle time.

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:

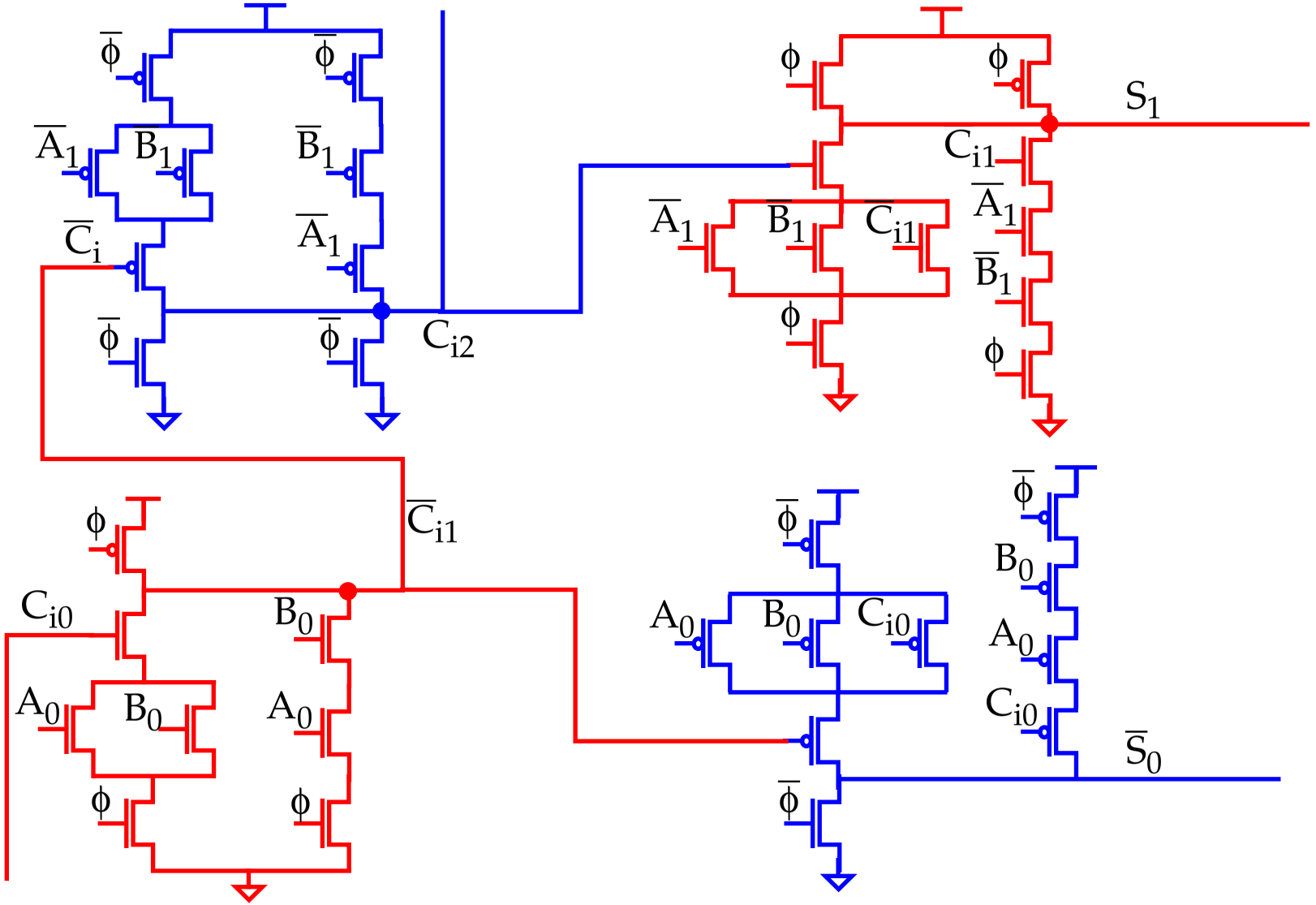


Note: S and C_o delay times are approximately equal -- good for multipliers.

See Weste and Eshraghian for an 18 transistor implementation.

Datapath Operators: Addition/Subtraction

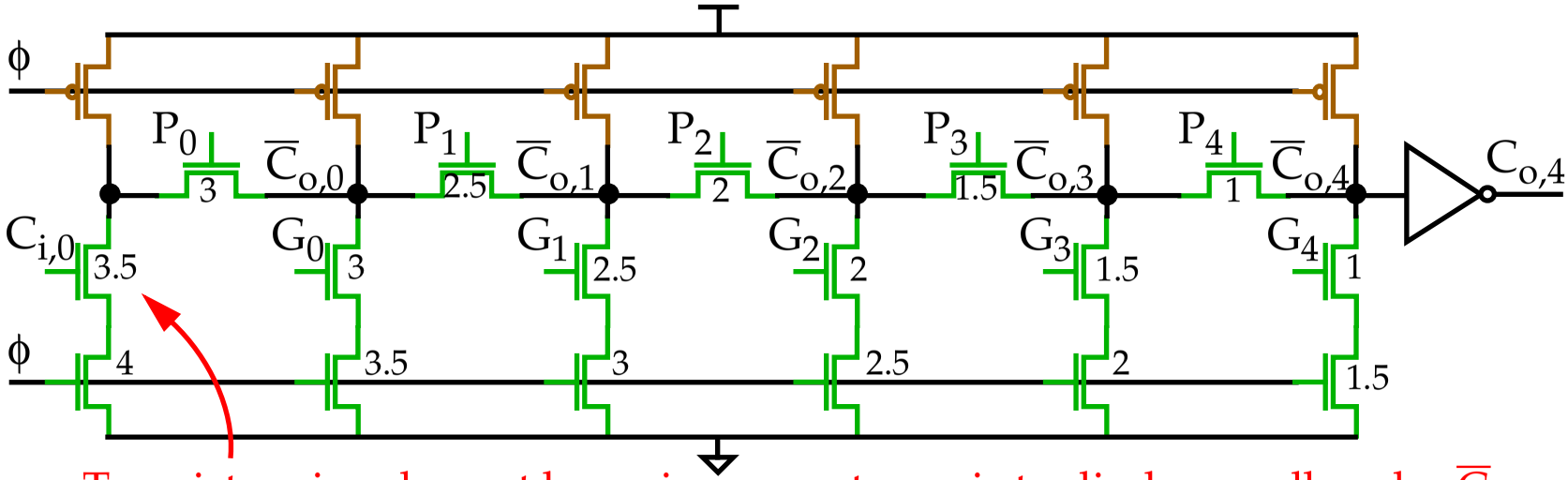
Dynamic Adder Design: *np-CMOS* adder



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}.

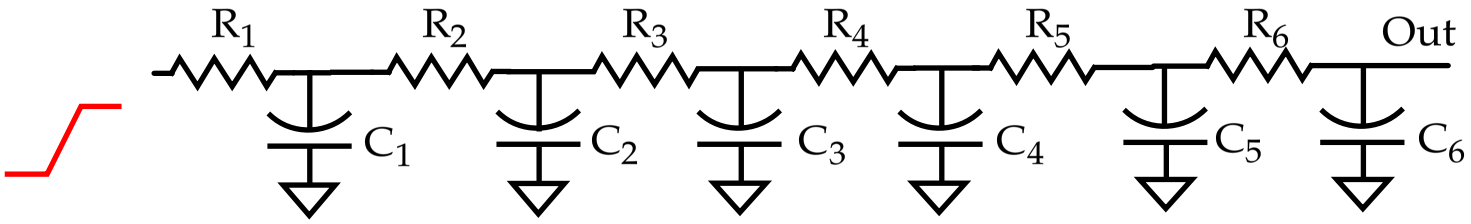
Evaluate: Node $\overline{C}_{o,k}$ is discharged, for example, if there is an incoming carry, $C_{i,0}$ and the previous propagate signals are high, P_0 to P_{k-1}.

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:

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

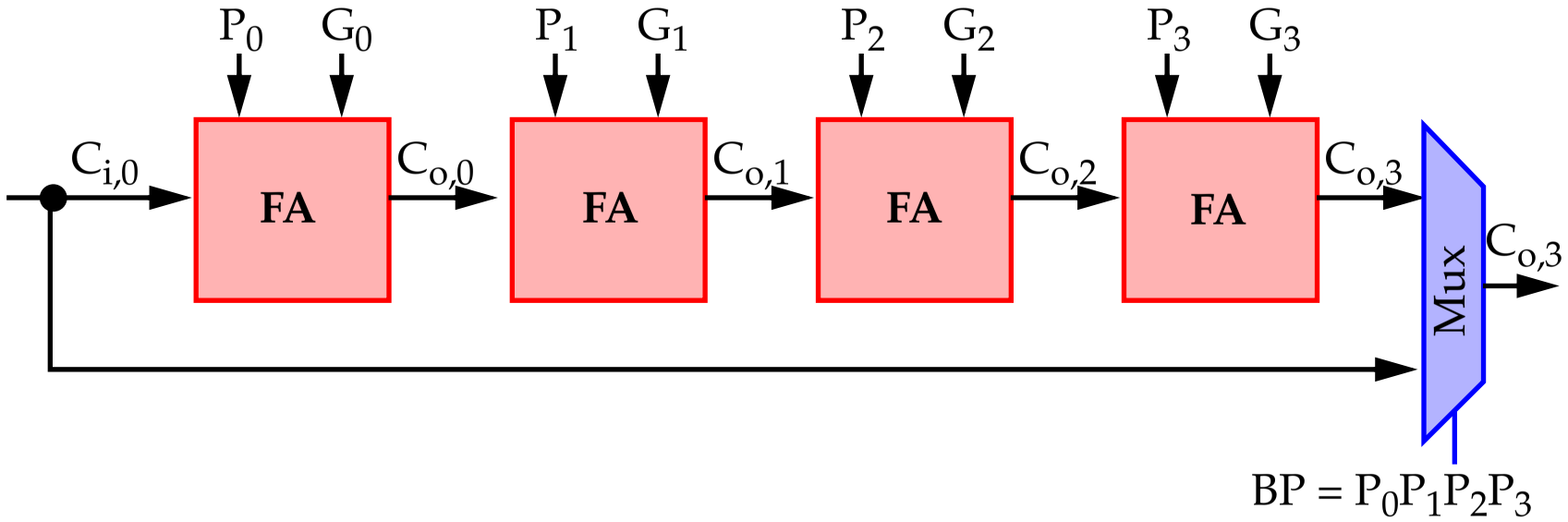
Since R_1 appears **6 times** in the expression, it makes sense to minimize its contribution.

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 $C_{o,3} = 1$.

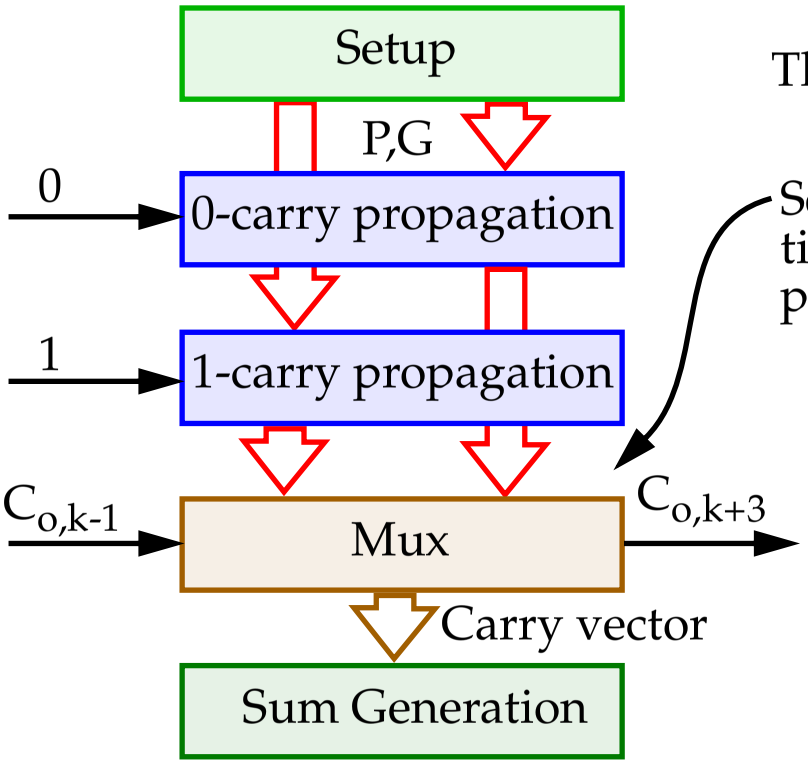
In other words:

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:

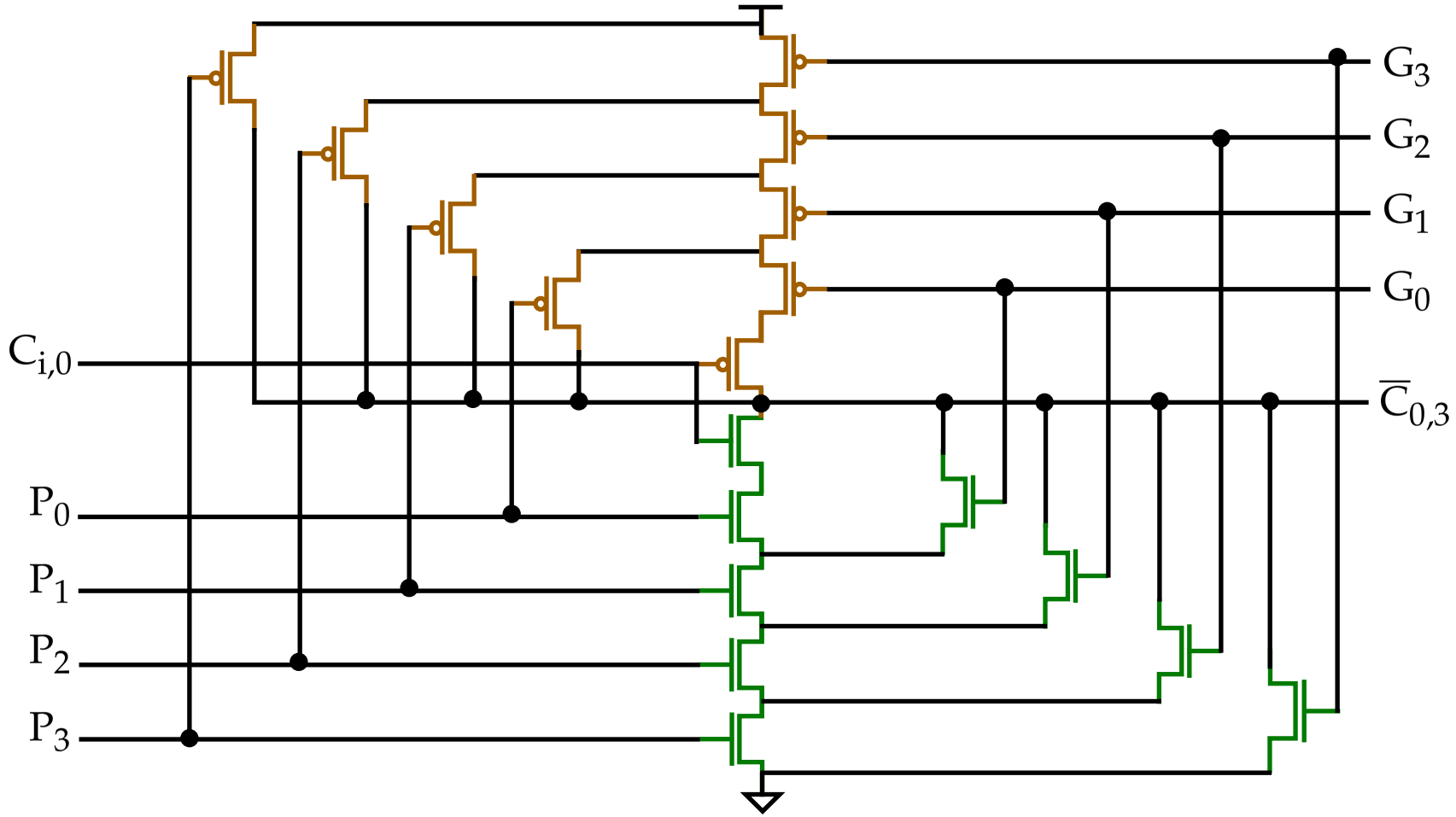
$$C_0 = G_0 + P_0C_i$$
$$C_1 = G_1 + P_1G_0 + P_1P_0C_i$$
$$C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_i$$
$$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_i$$

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.

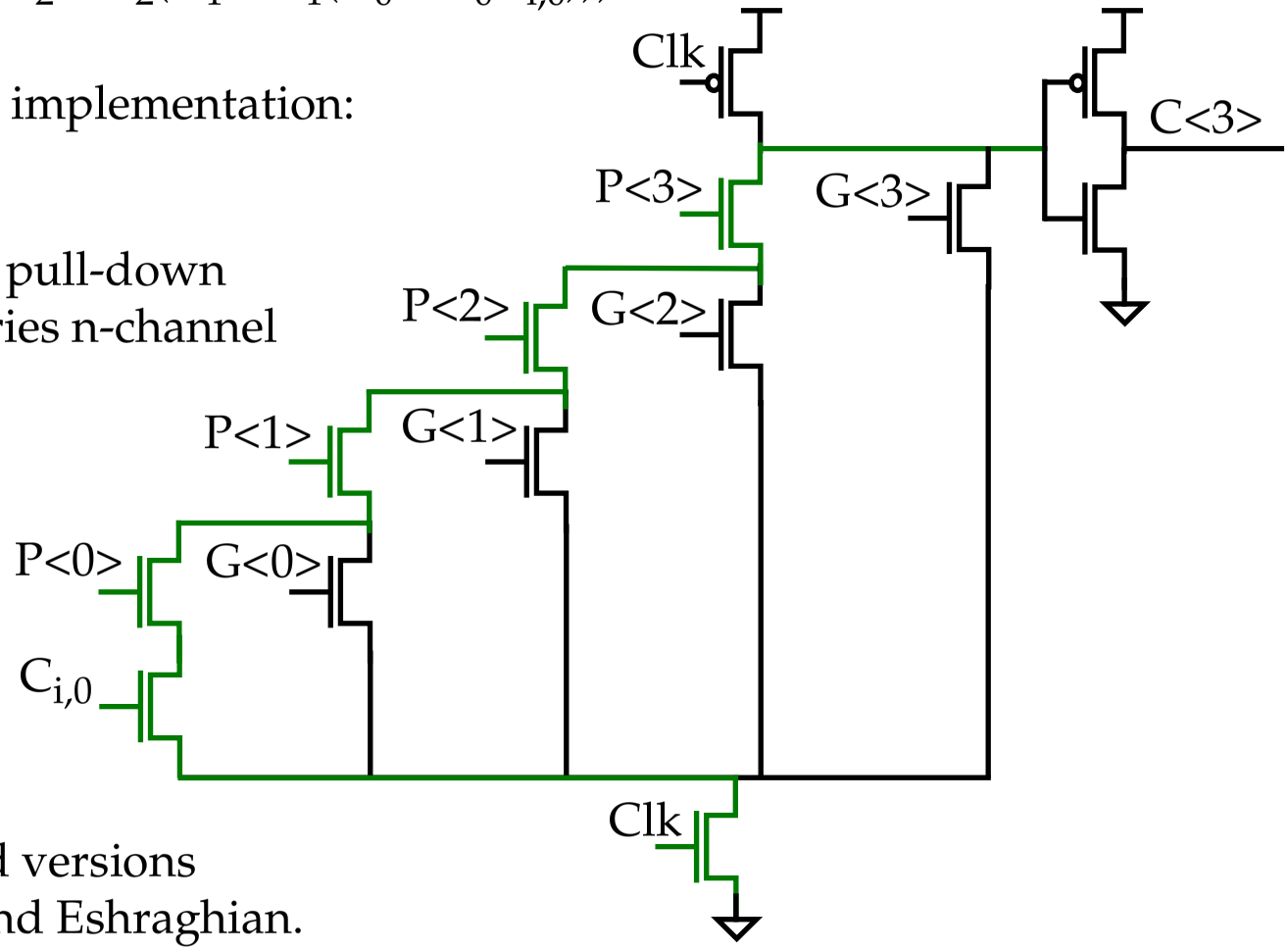
Datapath Operators: Addition/Subtraction

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:

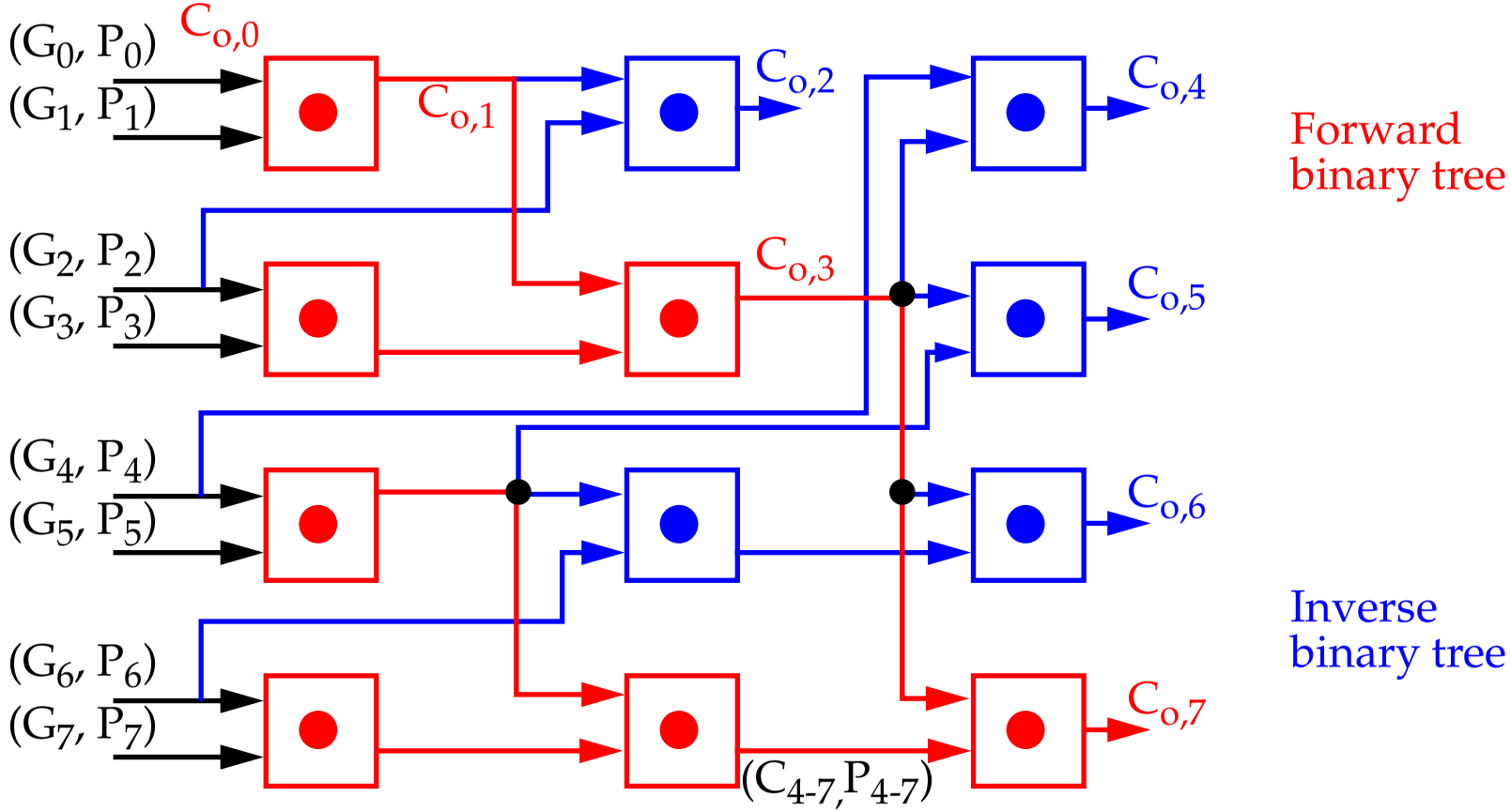


Worst case is pull-down through 6 series n-channel transistors.

Other high speed versions given in Weste and Eshraghian.

Datapath Operators: Addition/Subtraction

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



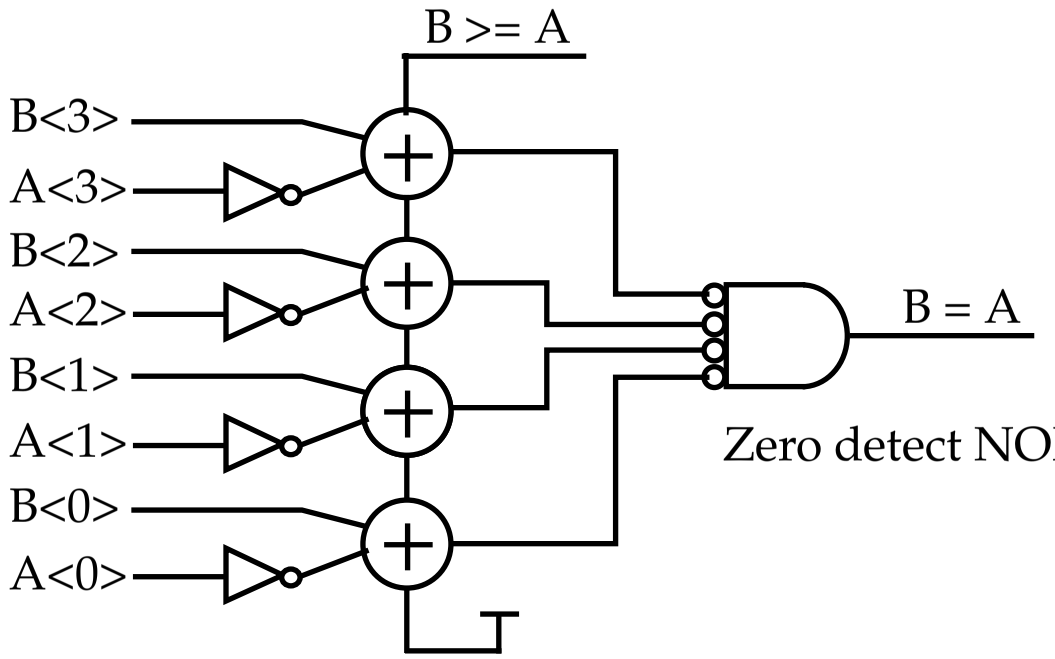
The dot operator (●)is defined as: (g, p) . (g′, p′) = (g + pg′, pp′)

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).

Datapath Operators: Comparison

Magnitude Comparators:

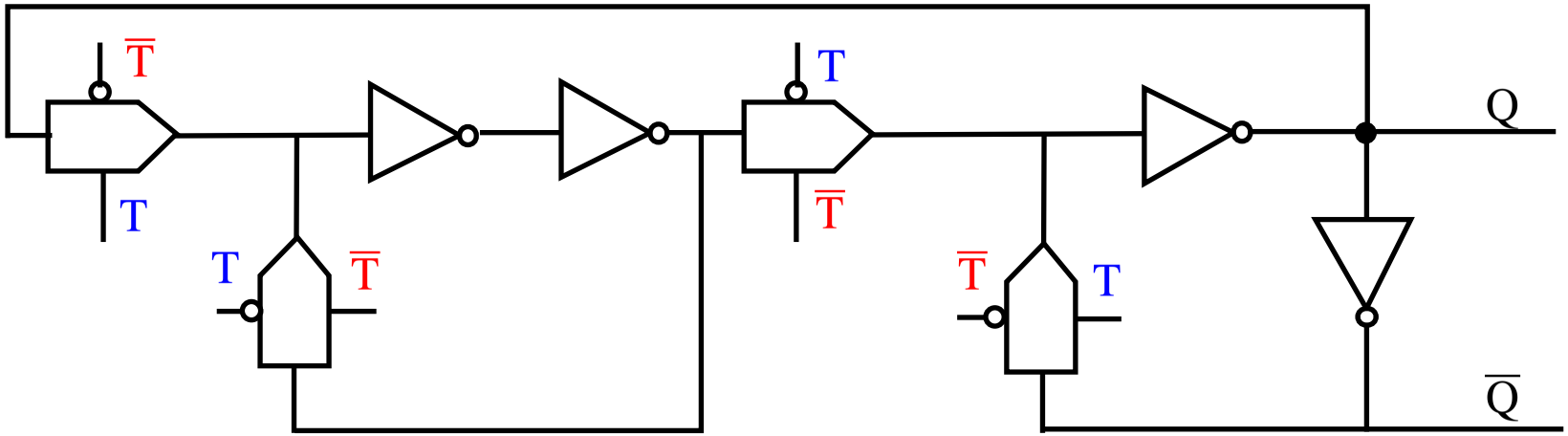
May be built from an adder, complementer (XOR gates) and a zero detect unit.

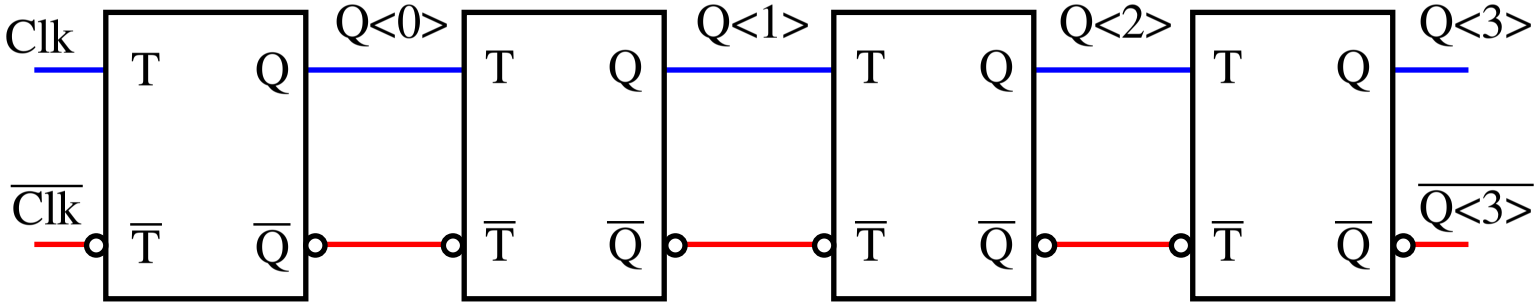


Think about the modifications necessary to make it a signed comparator (Hint: A couple of XOR gates).

Datapath Operators: Binary Counters

Asynchronous: Based on the Toggle register.



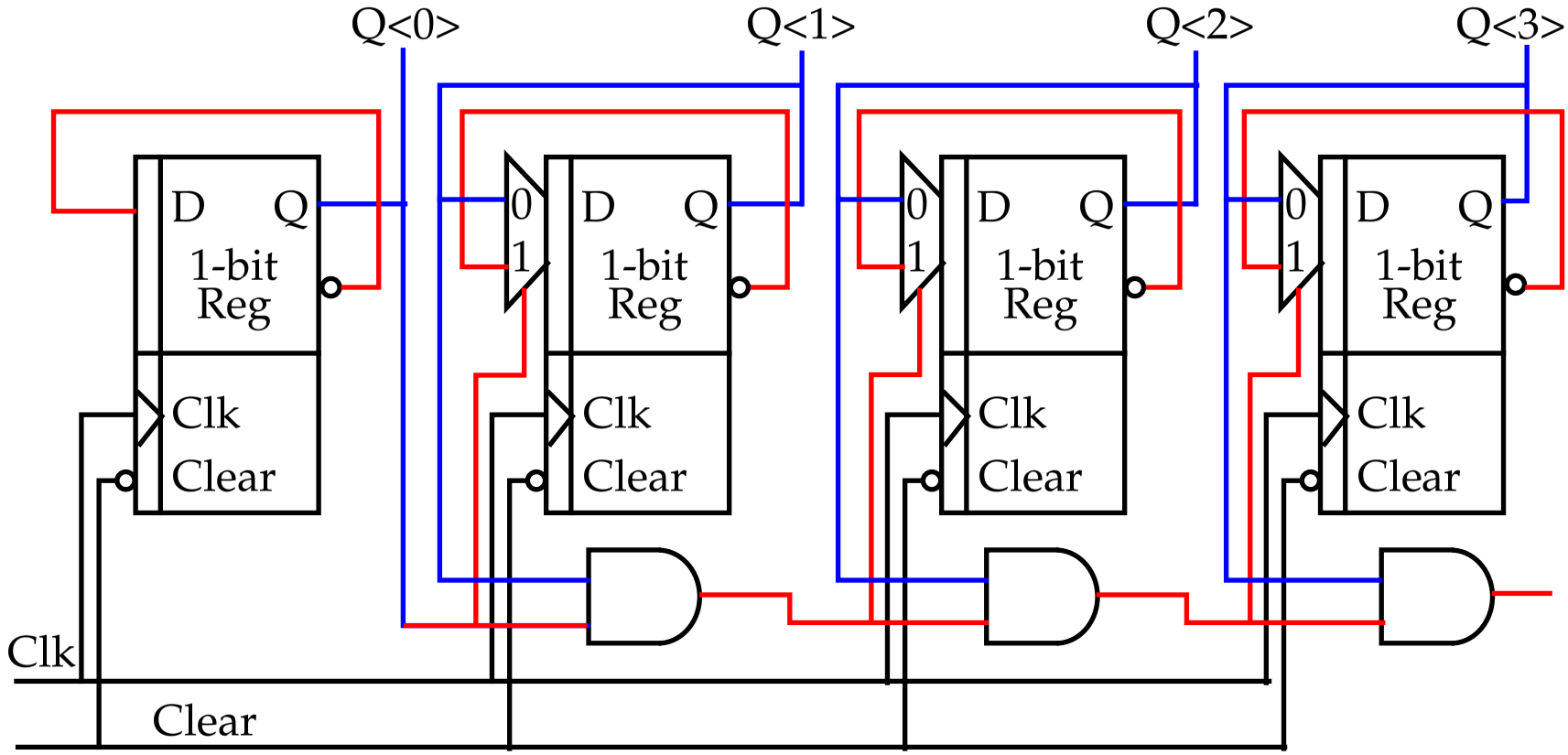


"Ripple Carry" Binary counter

Not a good choice for performance and testability (with no reset).

Datapath Operators: Binary Counters

Synchronous counter.



Replace AND gate with an adder for up/down counting capability.

Weste and Eshraghian also show a version that can be initialized.

Datapath Operators: Multiplication

Multiplication can be broken down into two steps:

- Computation of partial products.
- Accumulation of the shifted partial products.

```
    1100
X   0101
------
    1100
   0000
  1100
0000
------
0111100
```

Binary multiplication equivalent to AND operation

Multipliers may be classified by the format in which data words are accessed:

- Serial
- Serial/parallel
- Parallel

The parallel form computes the partial products in parallel.

Datapath Operators: Multiplication

Parallel Unsigned Multiplication:

Multiplying 2 unsigned binary integers results in:

$$X = \sum_{i=0}^{m-1} X_i 2^i$$

$$Y = \sum_{j=0}^{n-1} Y_j 2^j$$

$$P = X \times Y = \sum_{i=0}^{m-1} X_i 2^i \sum_{j=0}^{n-1} Y_j 2^j = \sum_{k=0}^{m+n-1} P_k 2^k$$

			X_3	X_2	X_1	X_0	Multiplicand
			Y_3	Y_2	Y_1	Y_0	Multiplier
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0	
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1		
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2			
X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3				
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

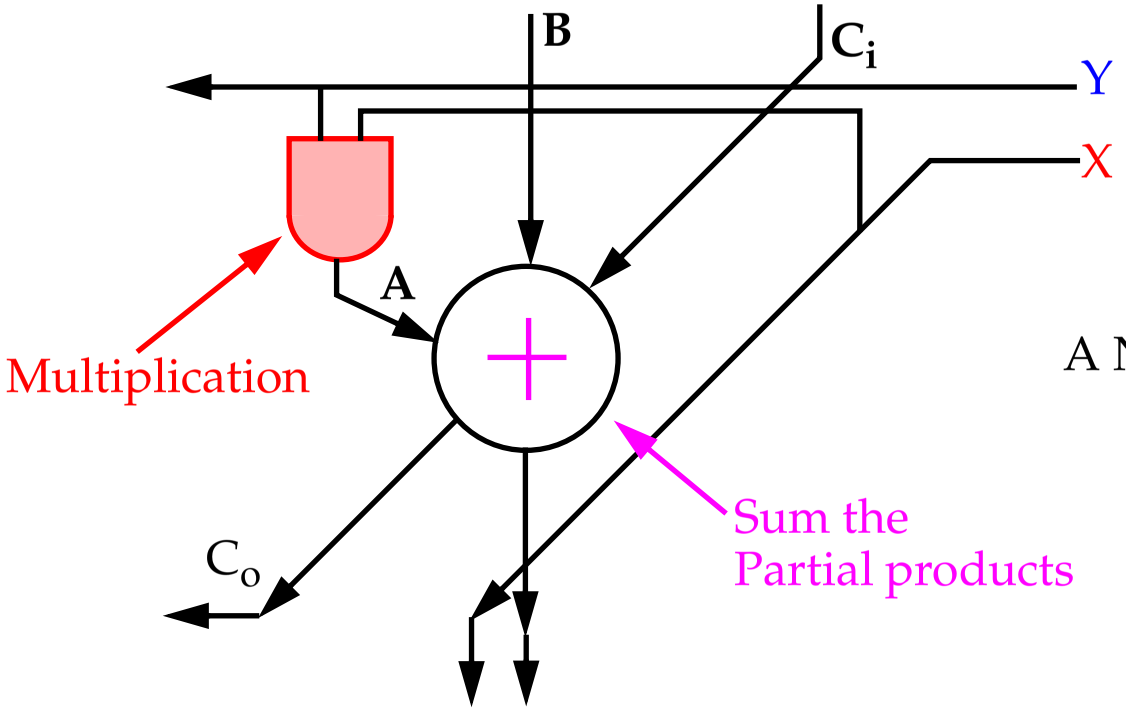
There are m*n summands produced by a set of m*n AND gates in parallel.

Datapath Operators: Multiplication

Parallel Multiplication:

Multiplication is carried out using a bitwise AND of the operands, X_i and Y_i.

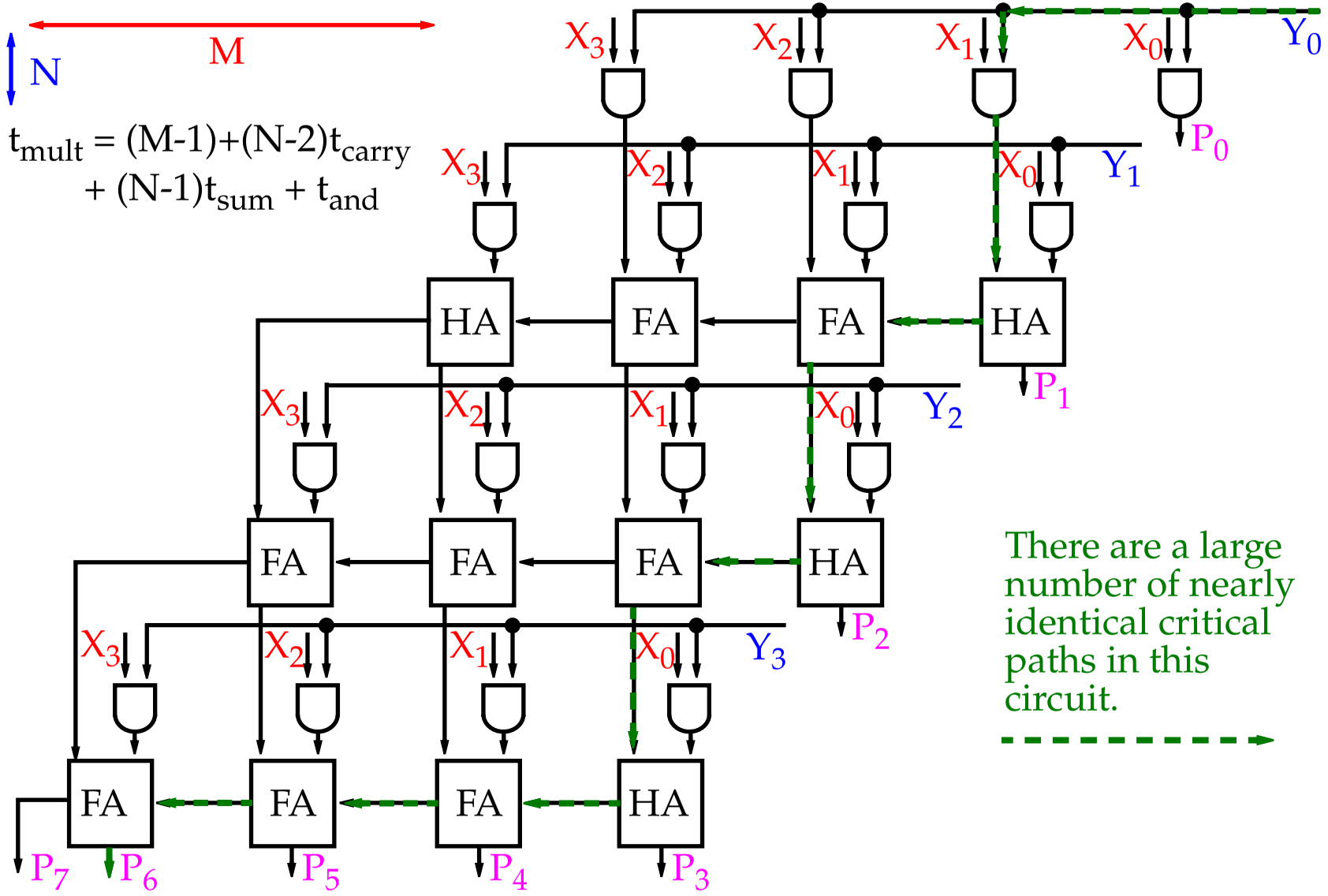
Most of the work (and delay) is in summing the partial products.



A NxN multiplier requires:
- N(N-2) full adders
- N half adders
- N^2 AND gates

Datapath Operators: Multiplication

Array multiplier:



Datapath Operators: Multiplication

From the delay expression and the fact that all critical paths have the same length, minimizing t_{mult} requires minimizing both t_{carry} and t_{sum}.

This is in contrast with the adder where minimizing t_{carry} was key.

The transmission gate adder is a good choice here.

Parallel Signed Multiplication:

Baugh-Wooley algorithm: Only 3 additional adders required over the unsigned version.

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

$$B = -b_{m-1}2^{m-1} + \sum_{i=0}^{m-2} b_i 2^i$$

Let A and B represent signed integers.

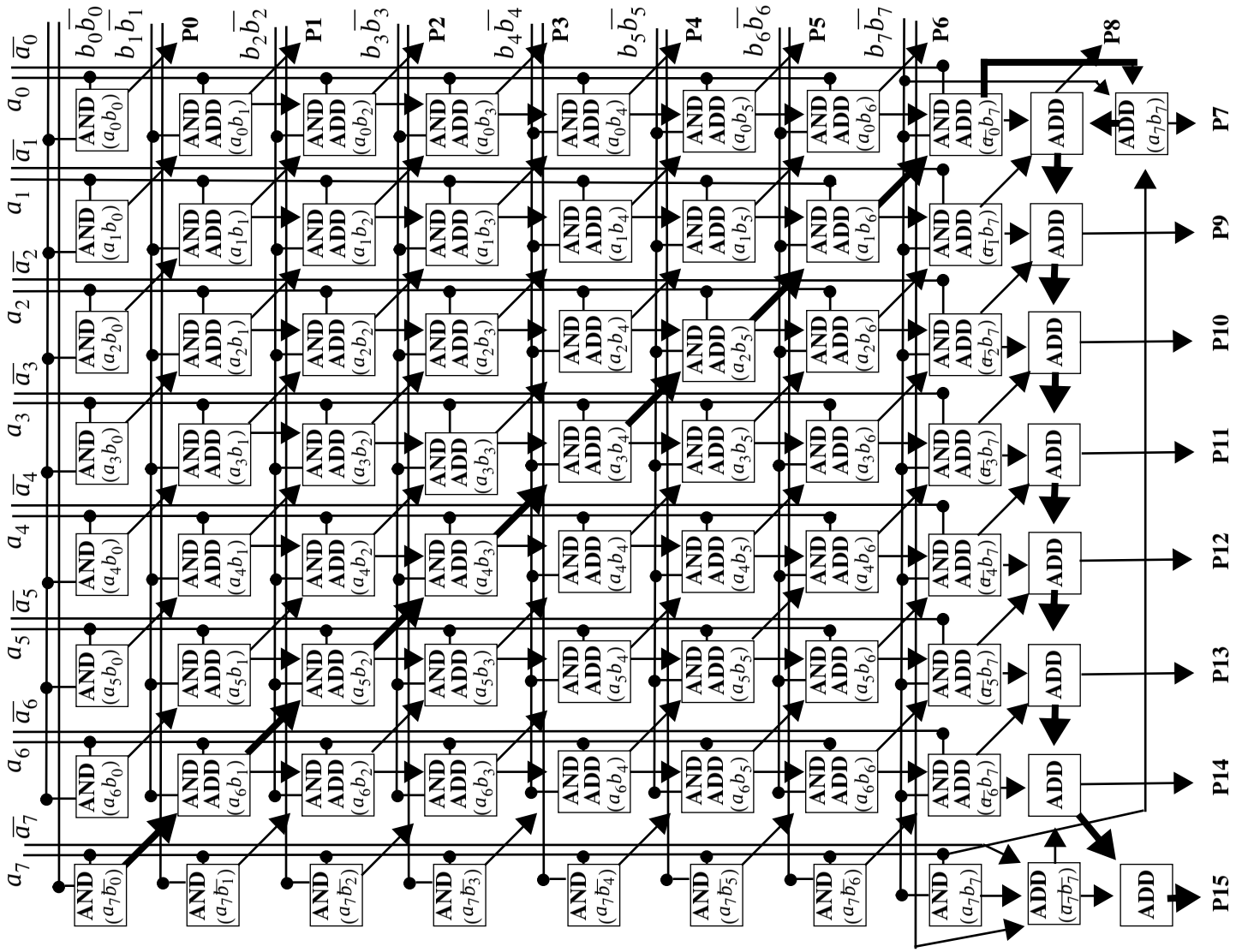
Expanding shows that the last two rows of summands are all negative so the algorithm simply adds in their negations.

$$P = \left(-a_{m-1}2^{m-1} + \sum_{i=0}^{m-2} a_i 2^i\right)\left(-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i\right)$$

$$= a_{m-1}b_{n-1}2^{m+n-2} + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} - \sum_{i=0}^{m-2} a_i b_{n-1} 2^{n-1+i} - \sum_{i=0}^{n-2} a_{m-1} b_i 2^{m-1+i}$$

Datapath Operators: Multiplication

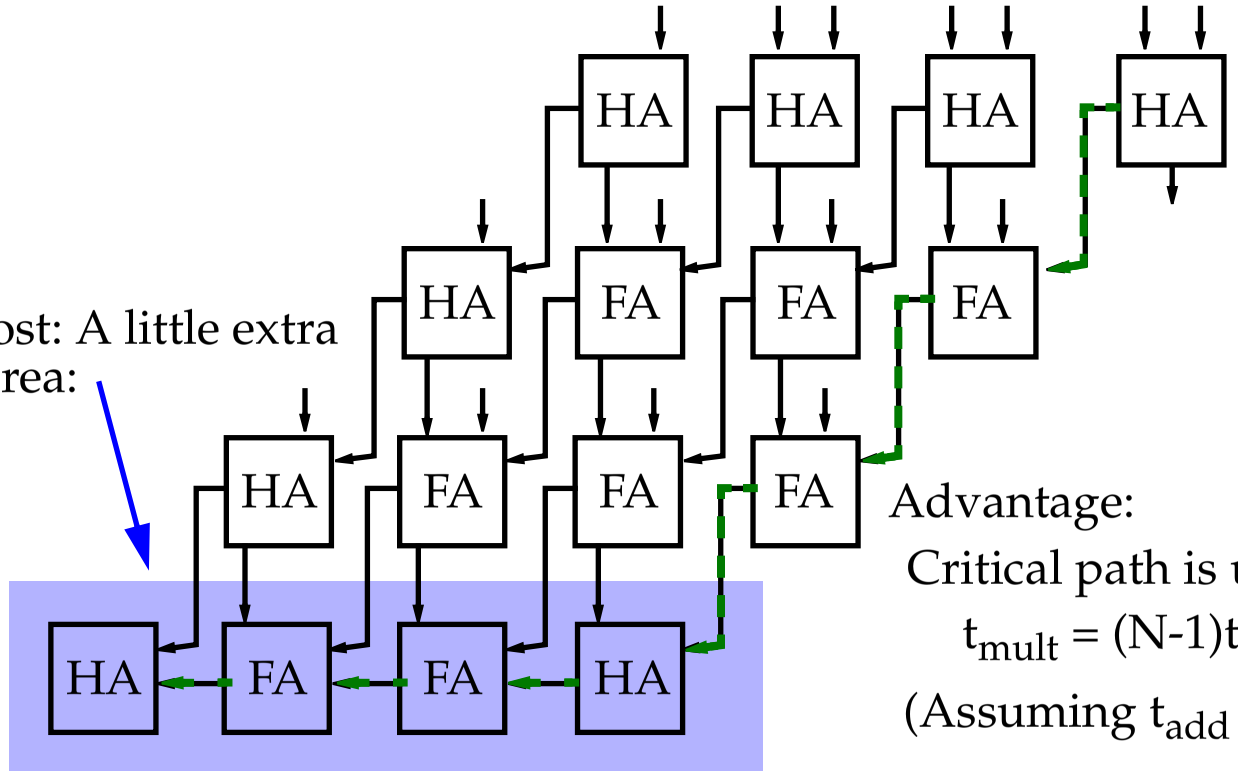
Parallel Signed Multiplication:



Datapath Operators: Multiplication

Carry-Save Multiplier:

Carry bits can be passed diagonally downwards instead of to the left.



4x4 version

Advantage:

Critical path is uniquely defined:

$t_{mult} = (N-1)t_{carry} + t_{and} + t_{merge}$

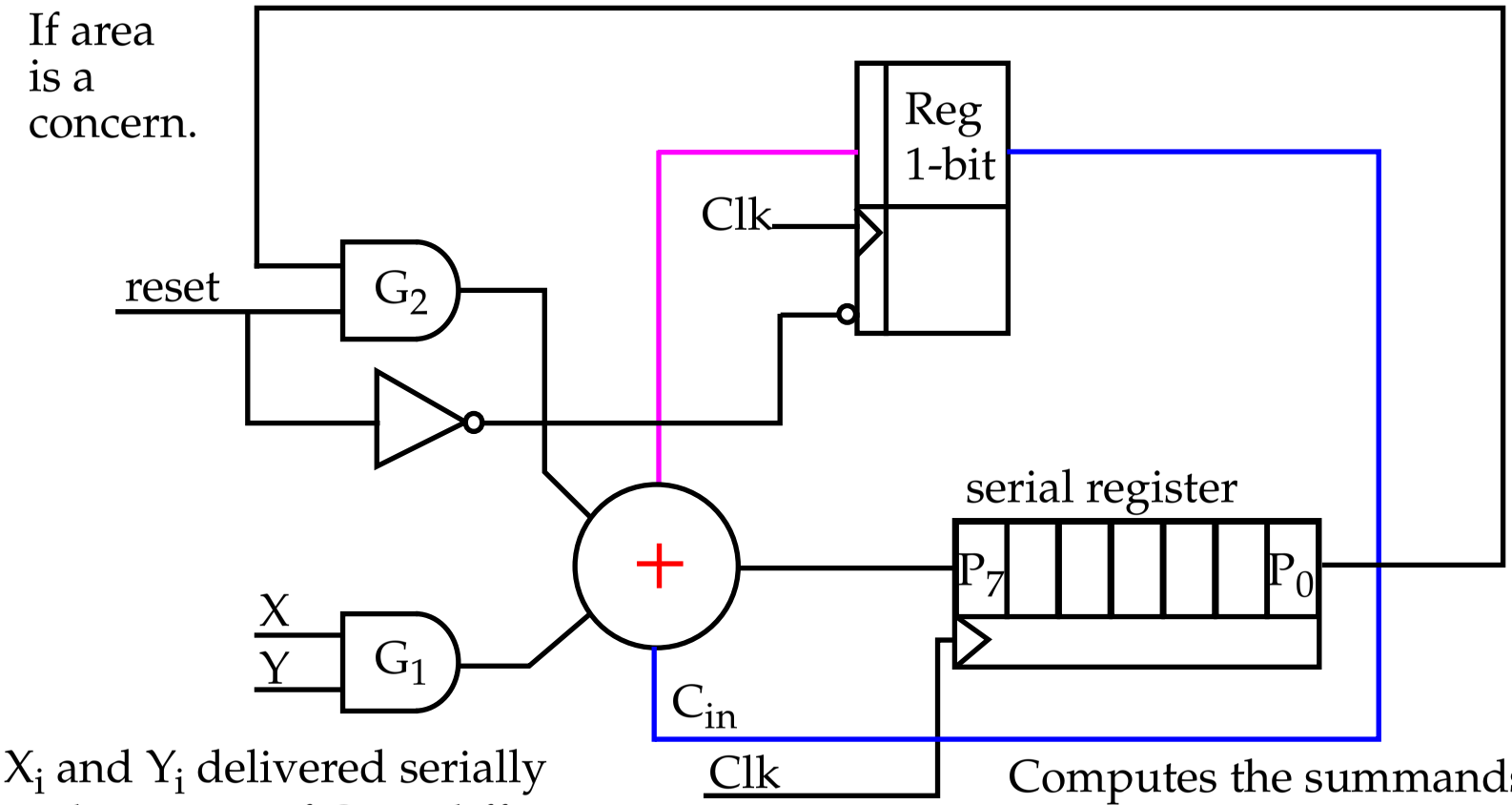
(Assuming $t_{add} = t_{carry}$).

Minimizing t_{merge} is useful,
e.g. use carry-select or lookahead.

Here the carry bits are not immediately added but rather "saved" for the next adder stage.

Datapath Operators: Multiplication

Serial Unsigned Multiplication:



Disadv: Quadratic delay: $t_{mult} = M \times N \times t_{carry}$

Serial/Parallel Unsigned Multiplier shown in Weste and Eshraghian.

Datapath Operators: Multiplication

Booth Encoding:

A special encoding of the multiplier word reduces the number of required addition stages and speeds up multiplication substantially.

Radix-4 scheme:

$$Y = \sum_{j=0}^{(N-1)/2} Y_j 4^j \text{ with } (Y_j \in \{-2, -1, 0, 1, 2\})$$

The number of partial products (and additions) is halved, resulting in area and speed advantage.

The disadvantage is a somewhat more involved multiplier cell.

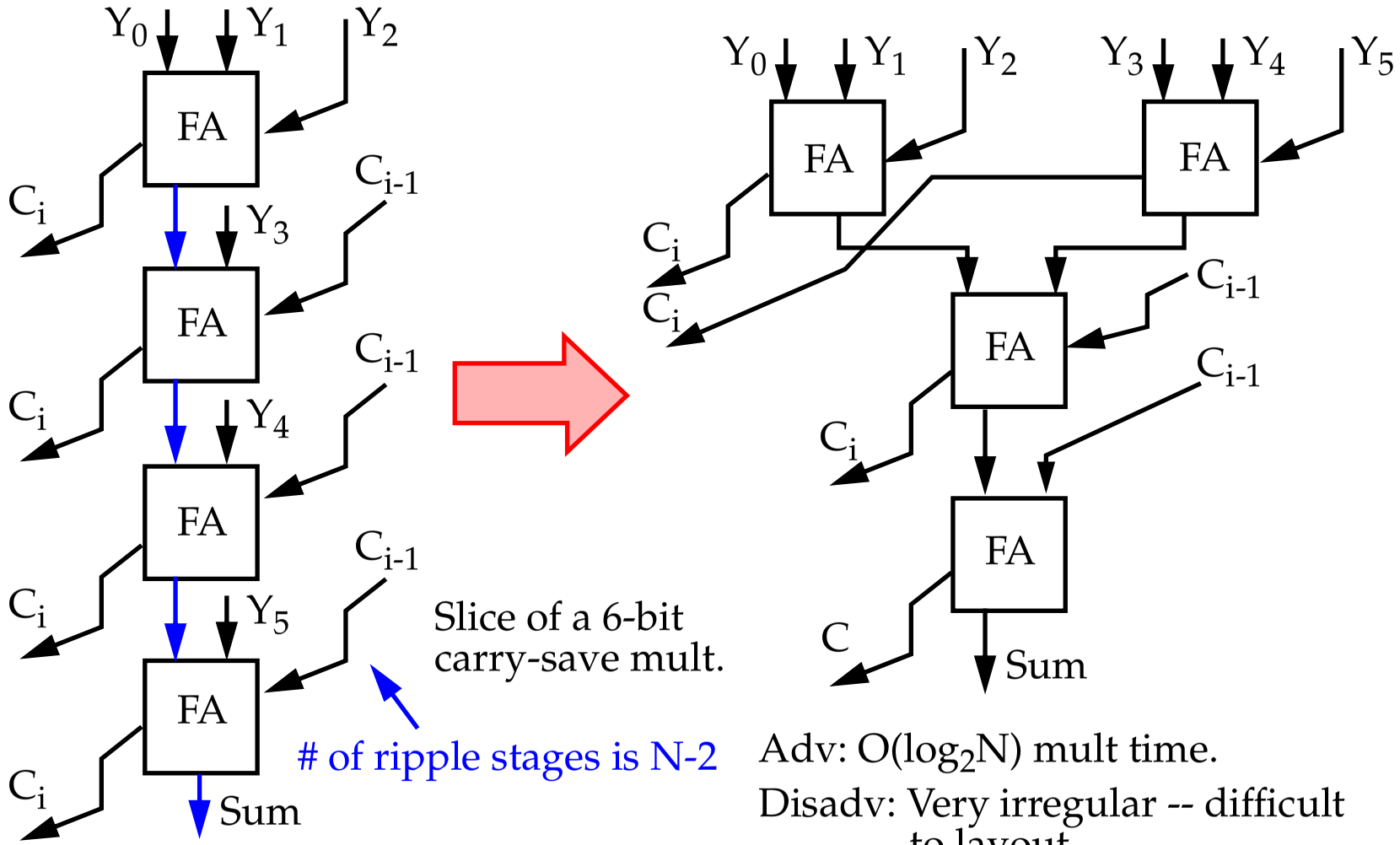
AND operation replaced with inversion and shift logic.

Virtually every multiplier in use employs the Booth scheme.

Datapath Operators: Multiplication

Wallace Multiplier:

Trees can be used to replace the linear partial-sum adders:

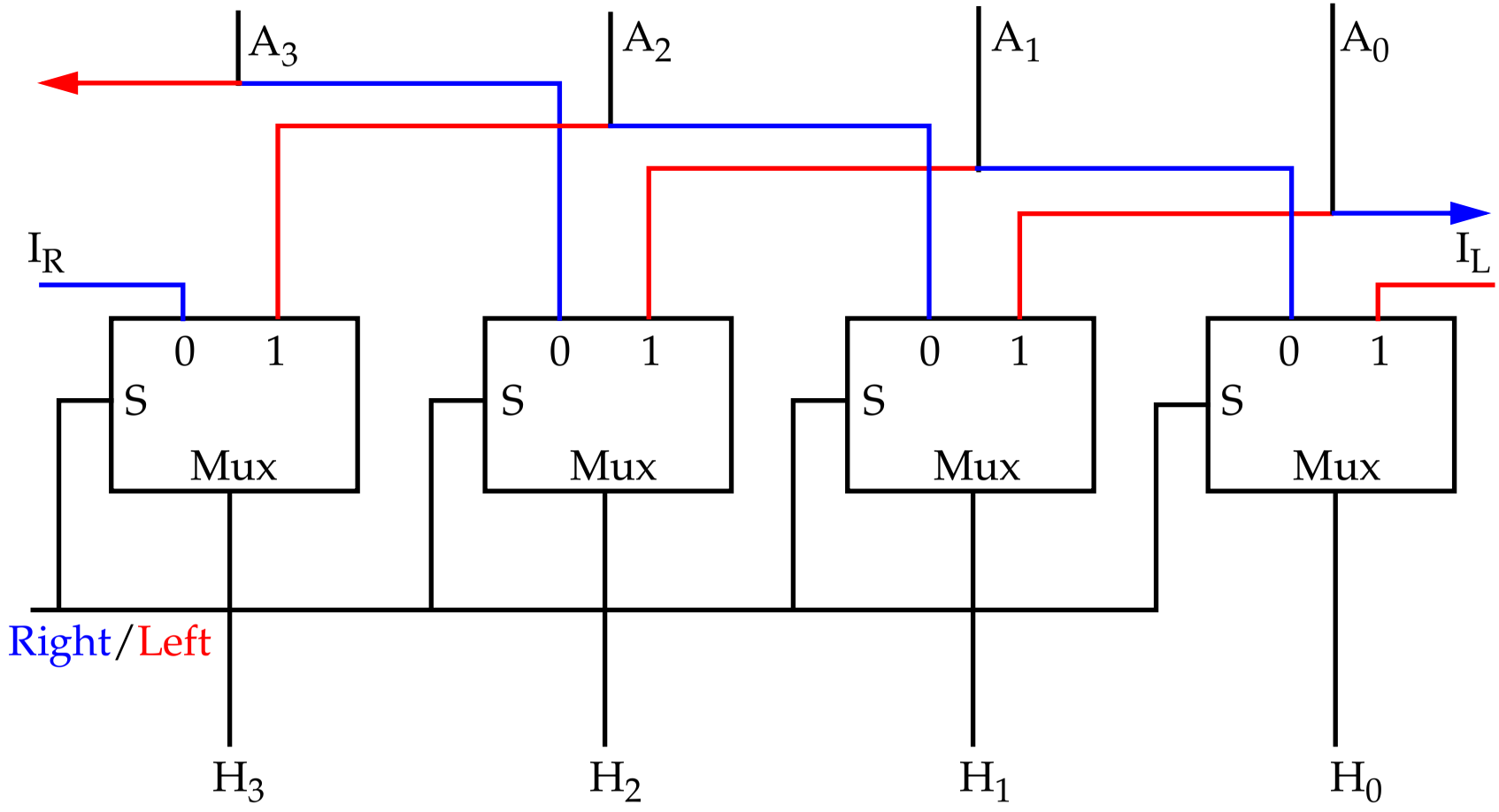


Adv: $O(\log_2 N)$ mult time.

Disadv: Very irregular -- difficult to layout.

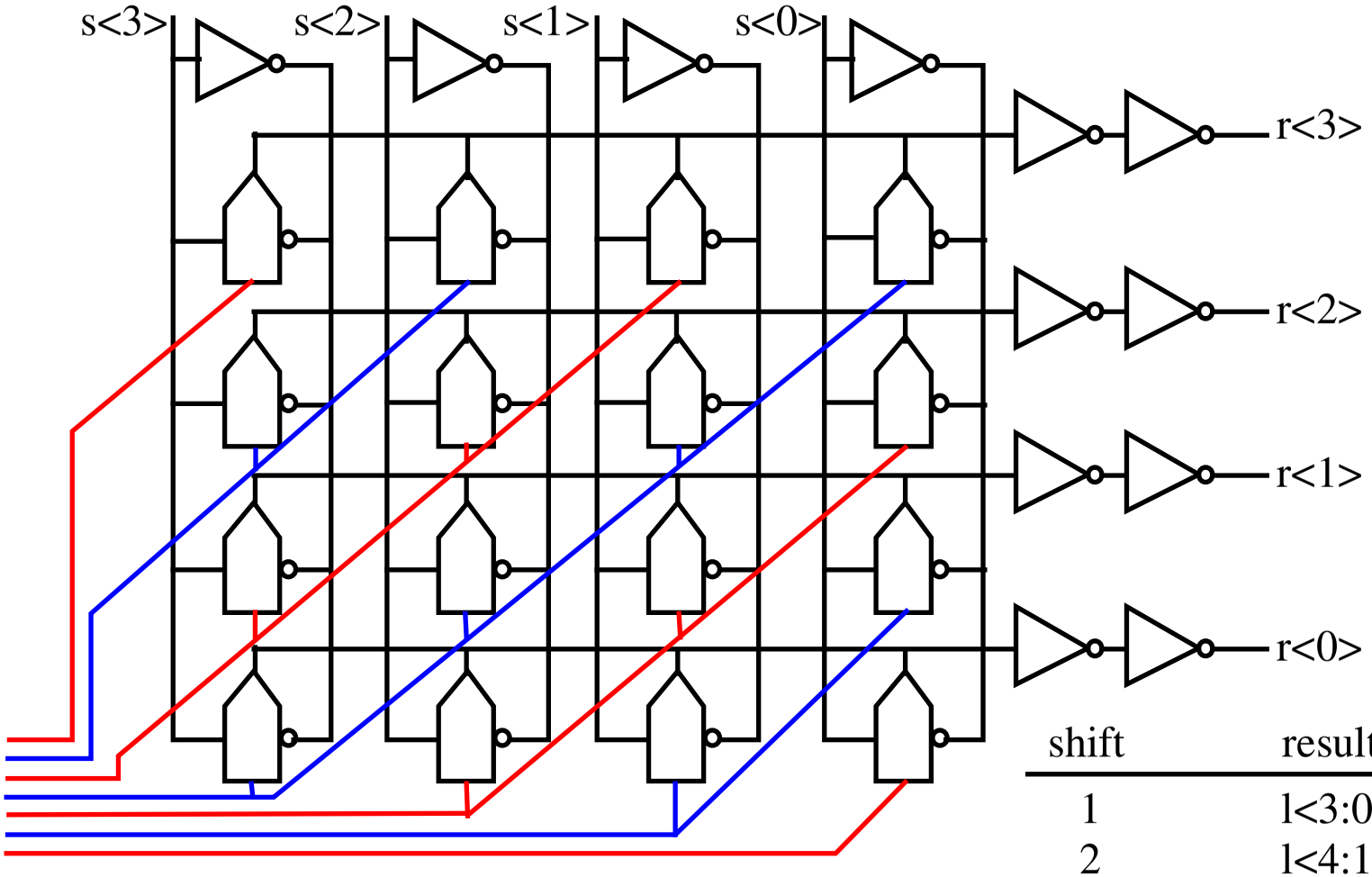
Datapath Operators: Shifters

Right/Left 1-bit shifter:



Datapath Operators: Shifters

Barrel shifter:



shift	result
1	l<3:0>
2	l<4:1>
4	l<5:2>
8	l<6:3>

Arithmetic and logical shifts and rotates possible by muxing l<6:0> to the appropriate values.