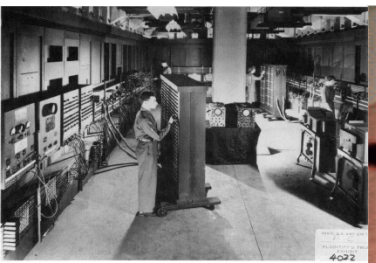
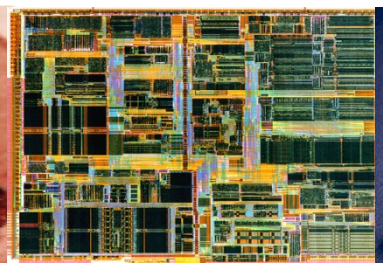
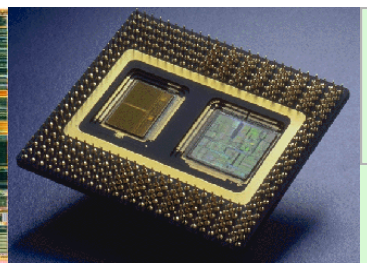
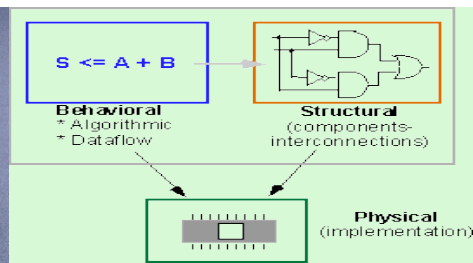
CpE 690: Introduction to VLSI Design

Lecture 13
Design of Arithmetic Circuits

Bryan Ackland
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07030

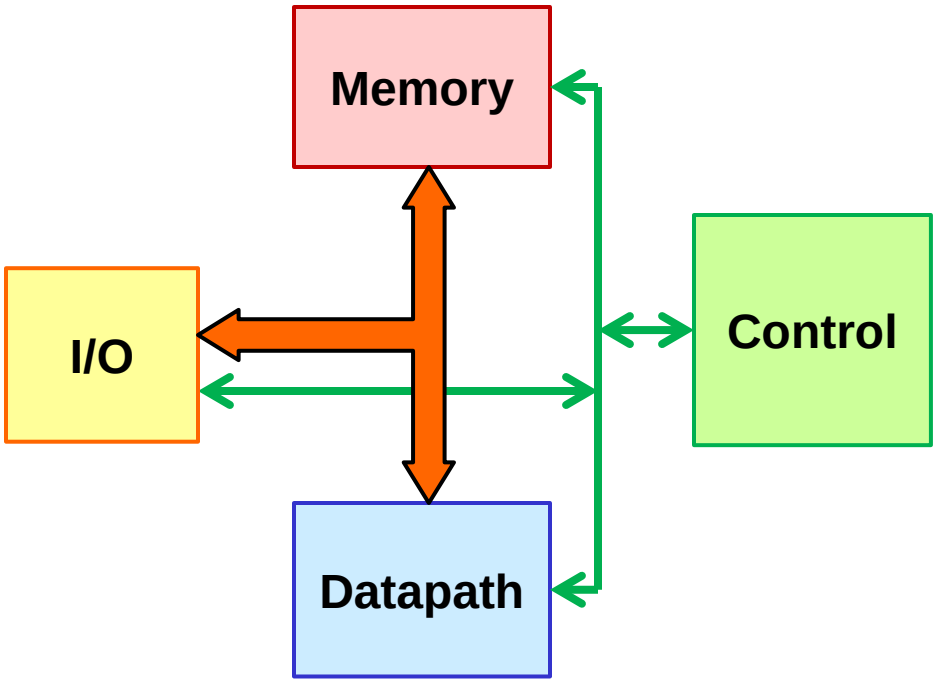
Adapted from Lecture Notes, David Mahoney Harris CMOS VLSI Design



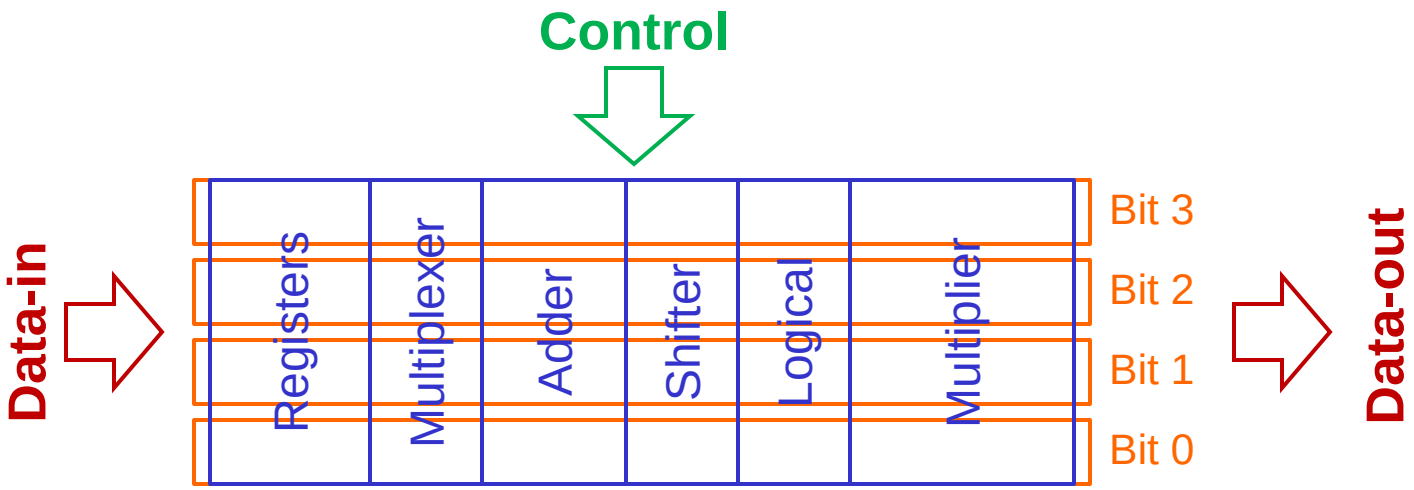
Generic Digital Processor

- Arithmetic Unit
  - bit-sliced datapath
  - adder, multiplier, shifter, comparator etc.
- Memory
  - RAM, ROM
  - registers, FIFO etc.
- Control
  - Finite state machine
  - PLA, random logic
- Interconnect
  - switches
  - arbiters
  - bus



Bit-Sliced Datapath

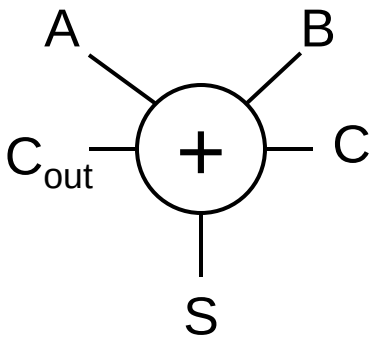
- Datapath (or ALU) may consist of number of arithmetic units components that operate on uniform width data words (e.g. 32-bit)
  - Arithmetic components often apply the same operation to each bit in the data word (e.g. add, and, left-shift)
- Bit-Sliced is an efficient physical layout style in which an *n*-bit datapath built by stacking together *n* 1-bit data paths
  - Data buses run (mostly) in the horizontal direction
  - Control runs (mostly) in the vertical direction



Single Bit Addition: Full Adder

$S = A \oplus B \oplus C$

$C_{out} = MAJ(A,B,C)$



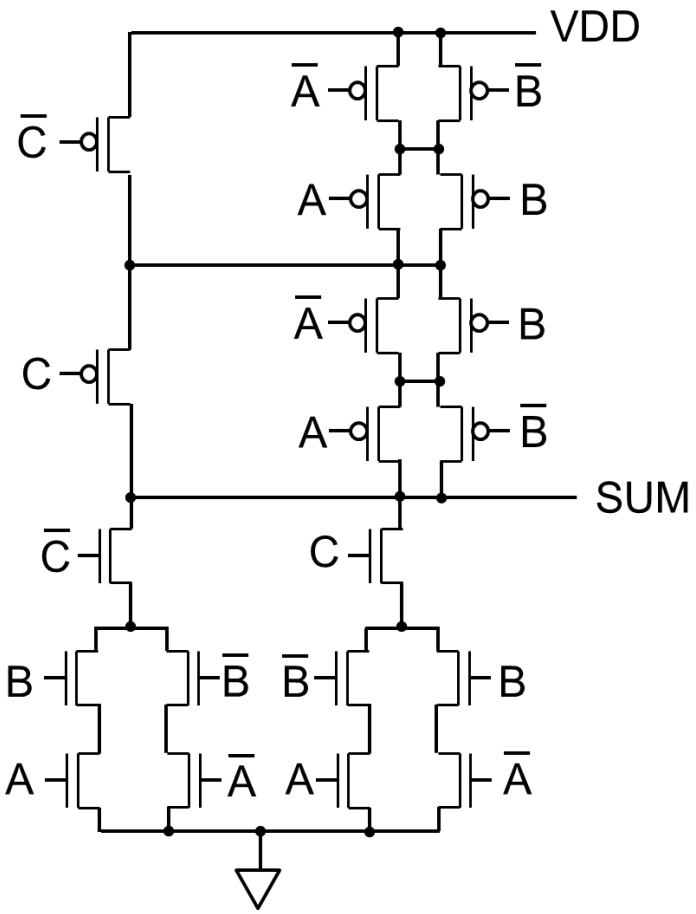
A	B	C	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

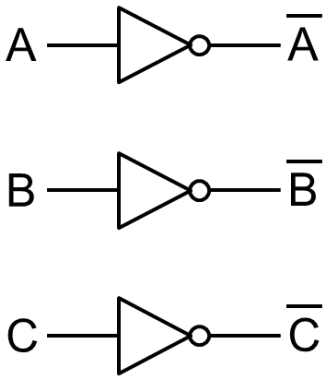
Full Adder Design I

- Direct Implementation of Boolean Equations:

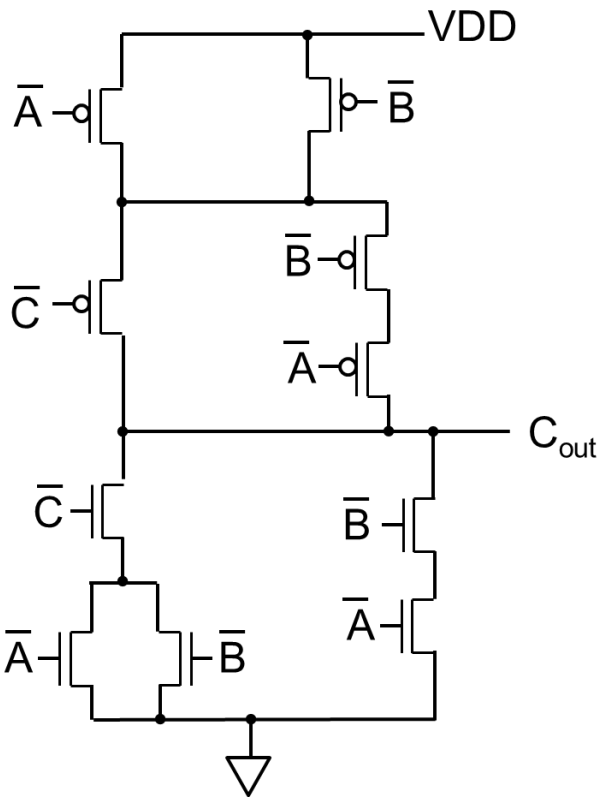
$SUM = A \oplus B \oplus C$

$C_{out} = MAJ(A,B,C)$



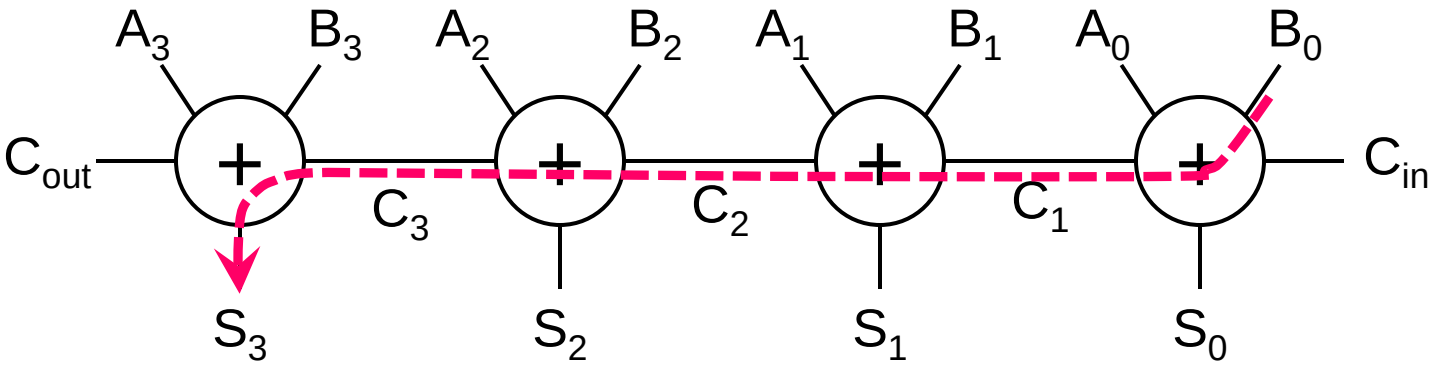


36 transistors



Mult-bit Ripple Carry Adder

- Simplest design: cascade full adders



- Critical path goes from C_{in} to C_{out}
- Worst case delay is linear in number of bits

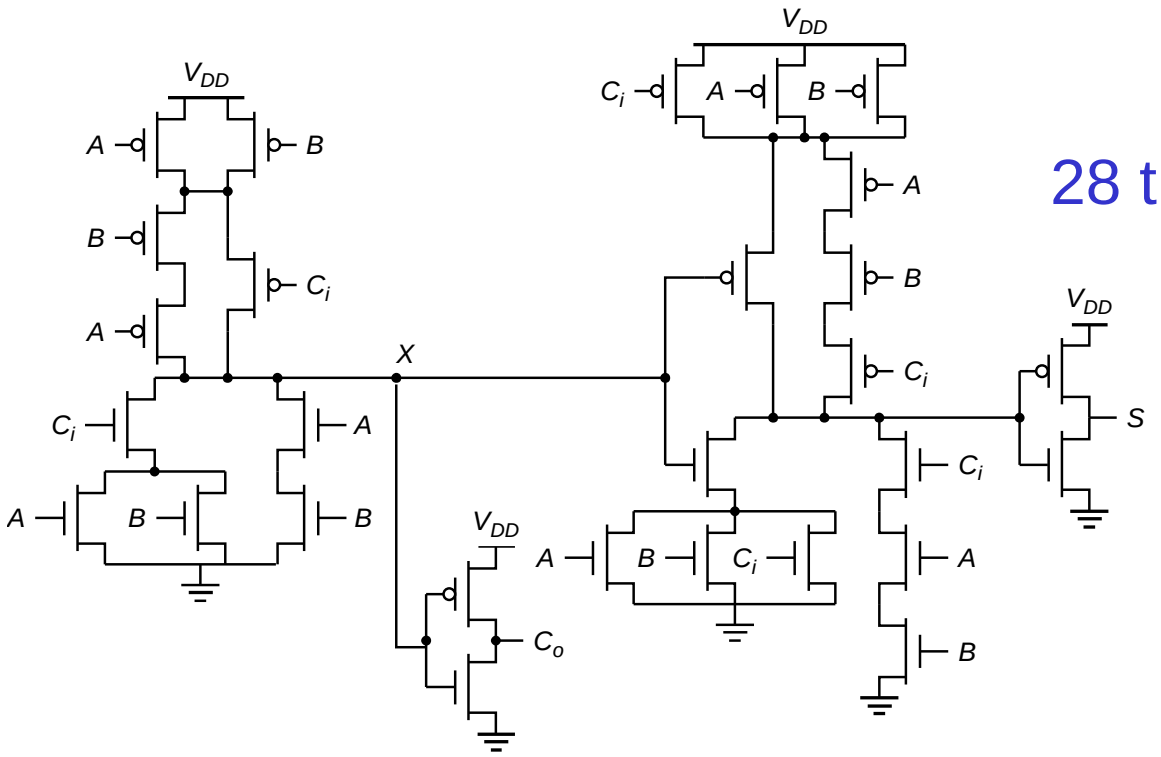
$$t_{d\ (N\text{-}bit\ adder)} = (N\text{-}1).t_{carry} + t_{sum}$$

- To make a fast adder, we need to minimize delay
  t_{carry} = (delay from C to C_{out}) in each full adder
  - t_{sum} (delay from A,B,C to S) is negligible for large N

Full Adder Design II

- A more compact design can be realized by generating S as a function of C_{out} :

  $S = ABC + (A + B + C).\overline{C}_{out}$

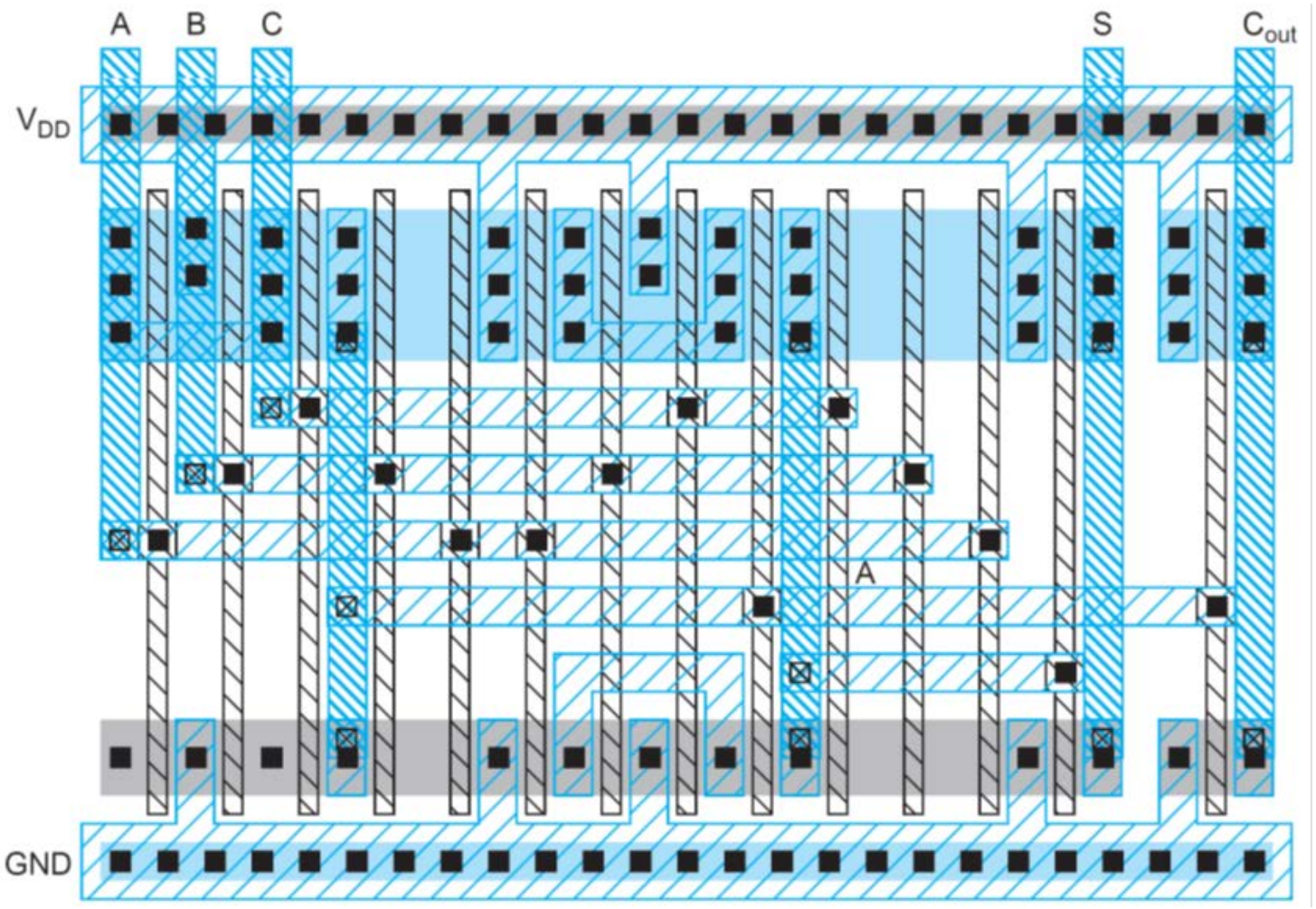


28 transistors

- If we could eliminate output inverters
  - simplify design and reduce C to C_{out} delay

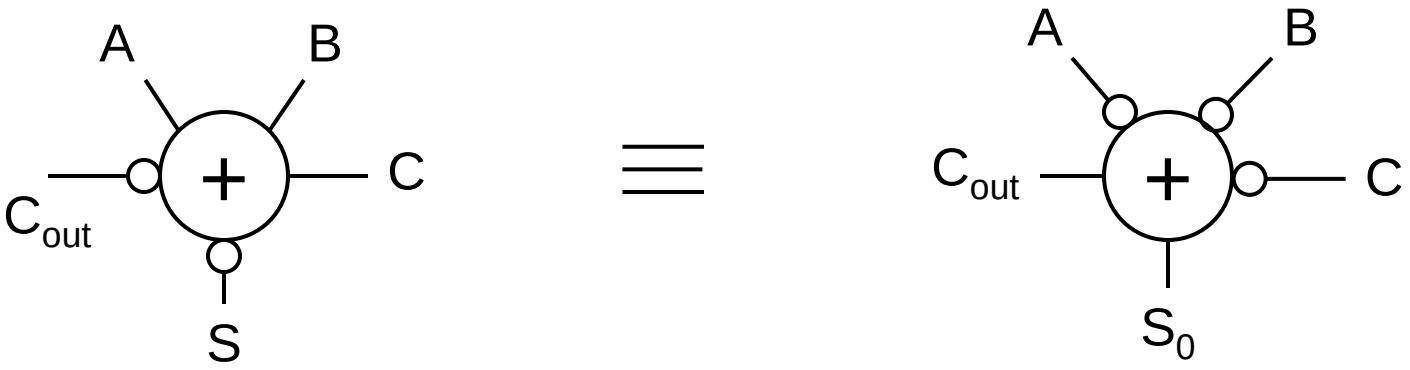
Full Adder Design II - Layout

- Standard cell style (not bit-slice) layout:

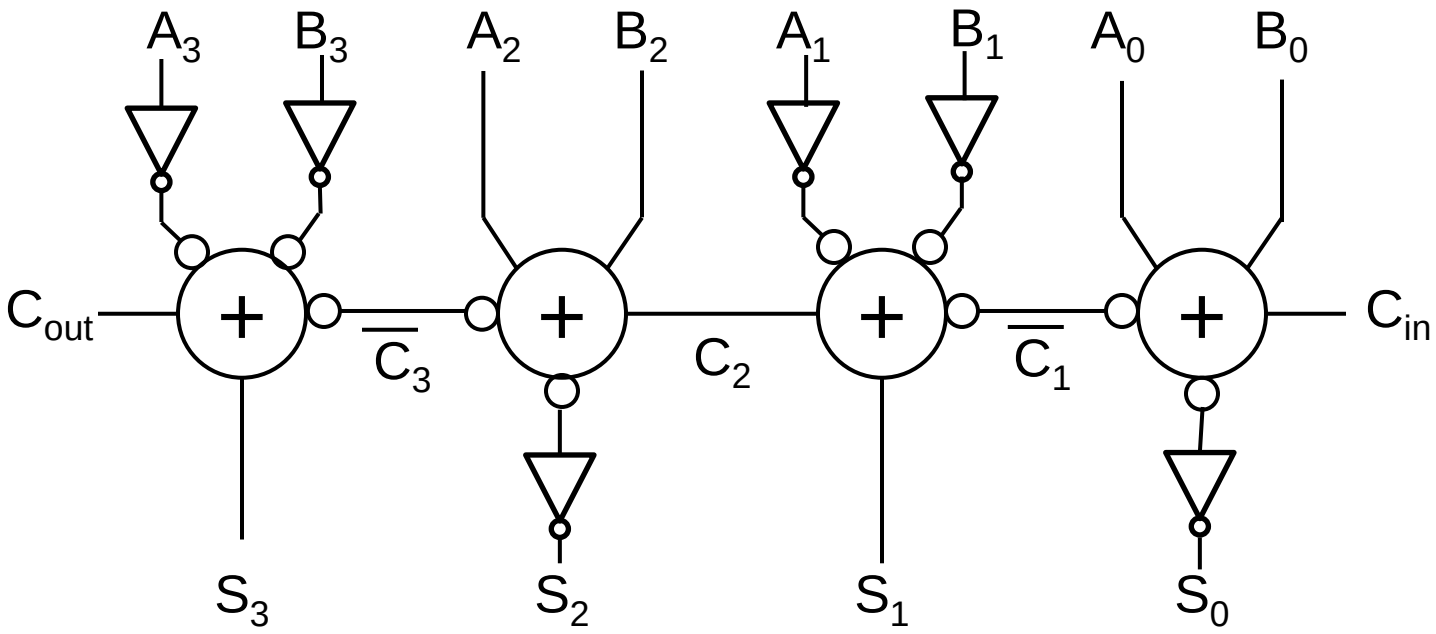


Full Adder Inversion

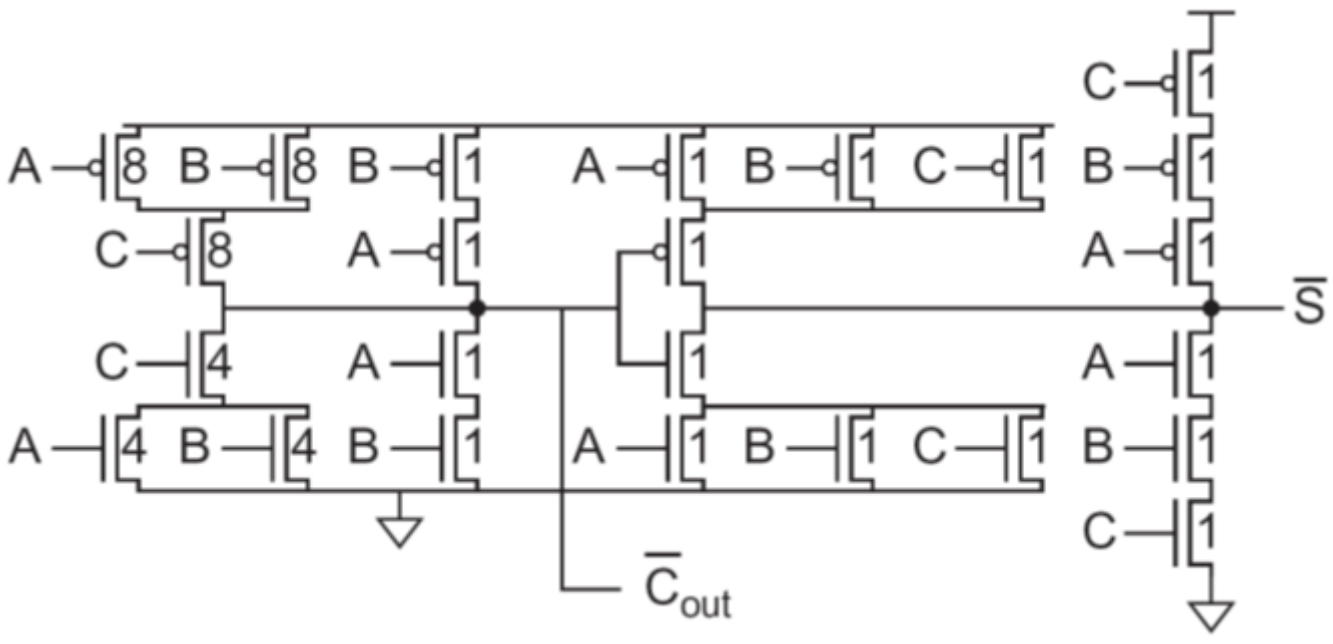
- Full adder is symmetric with respect to signal inversion:



- Eliminate inverters and use inverting full adder



Full Adder: Design III (Mirror Adder)

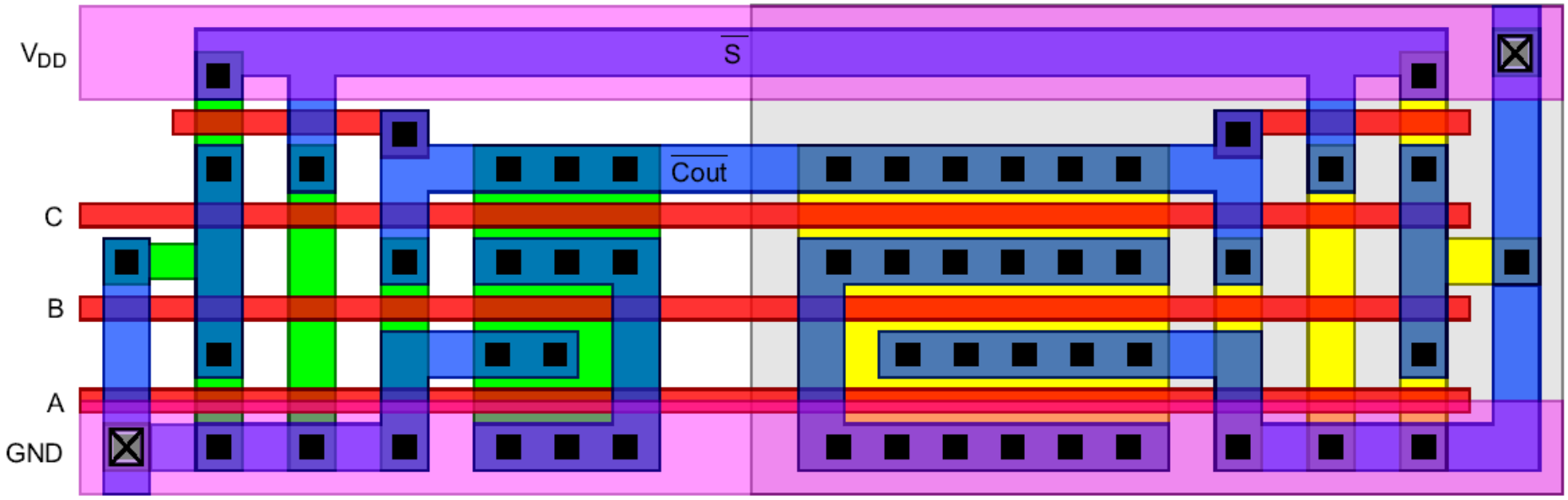


24 transistors

- Output inverters removed
- pMOS and nMOS networks are “mirror” of each other
  - rather than complimentary – simplifies layout
  - enabled by symmetry of the add operation
- Transistors placed & sized to minimize carry propagation
  - at the expense of sum generation

Mirror Adder Layout

- Bit-slice cell style (not standard cell) layout:



- Transistors now run vertically with horizontal poly
- Data travels from left to right
  - carry propagates vertically from one bit to the next
- Can build wide transistors without affecting bit pitch

GPK Representation

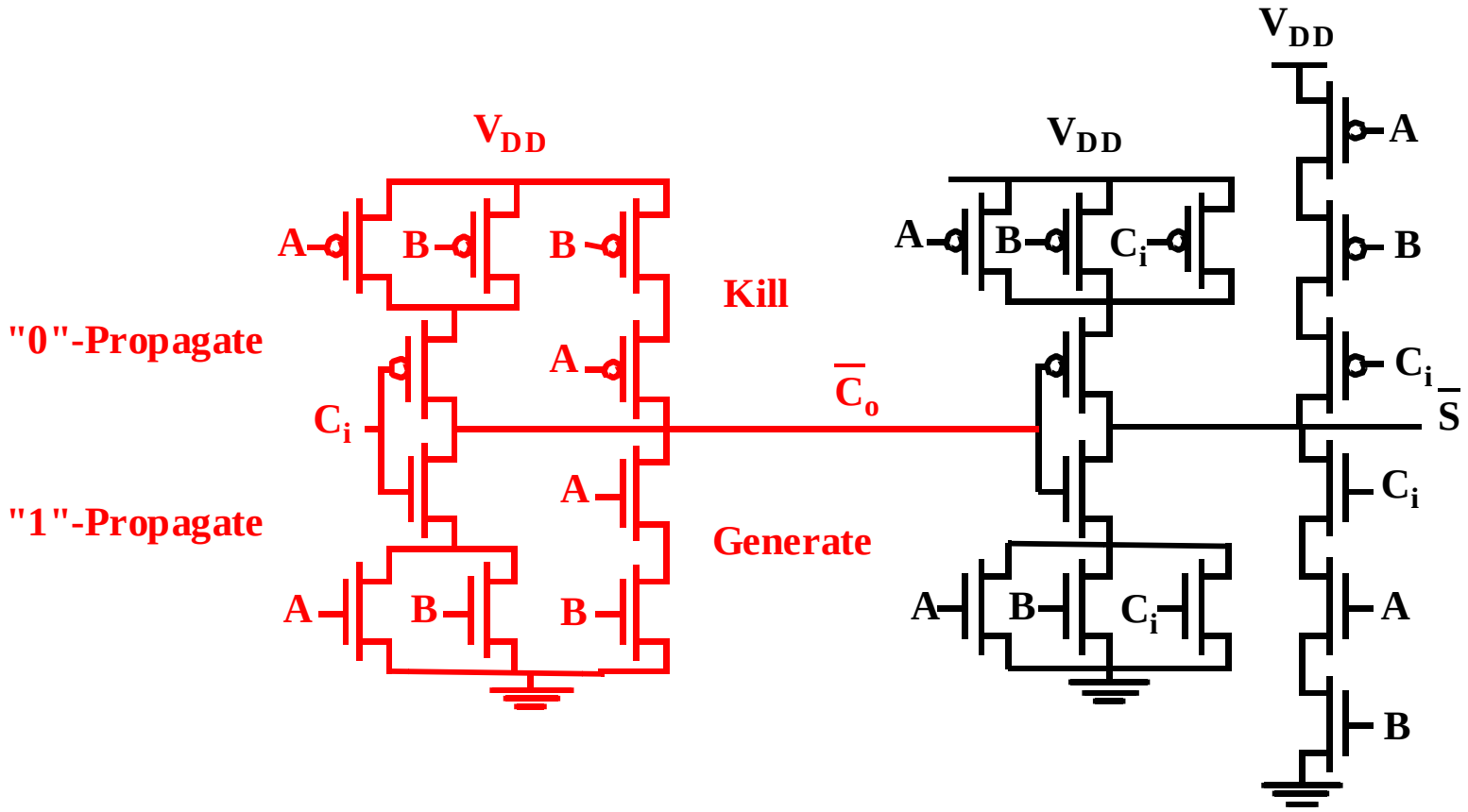
- Introduce new intermediate signals that describe full adder operation in terms of carry propagation

A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

- $G = A \bullet B$ (i.e. generate carry: $C_{out} = 1$ independent of C)
- $K = \overline{A} \bullet \overline{B}$ (i.e. kill carry: $C_{out} = 0$ independent of C)
- $P = A \oplus B$ (i.e. propagate carry: $C_{out} = C$)
- Note that G, P and K are:
  - mutually exclusive
  - only functions of A and B (don't need to wait for C)

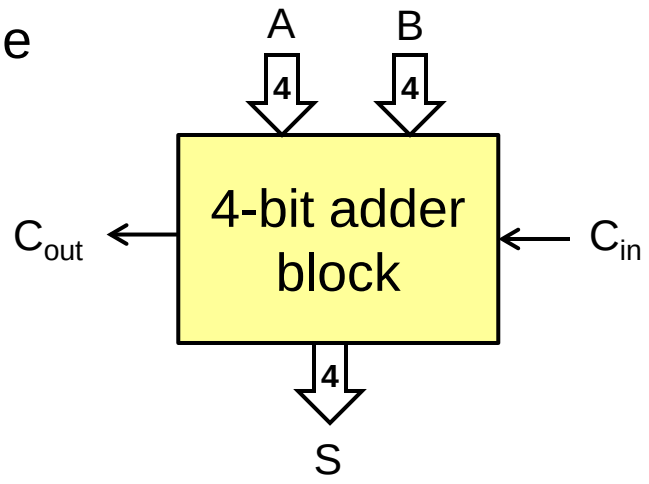
GPK Representation

- Can see the action of generate, propagate and kill operators in mirror adder:



Using GPK to Speed up Carry Propagation

- Divide the words to be added into bit groups or blocks
  - e.g. think about adding 4-bits at a time



- Addition of each block is a three-step process:

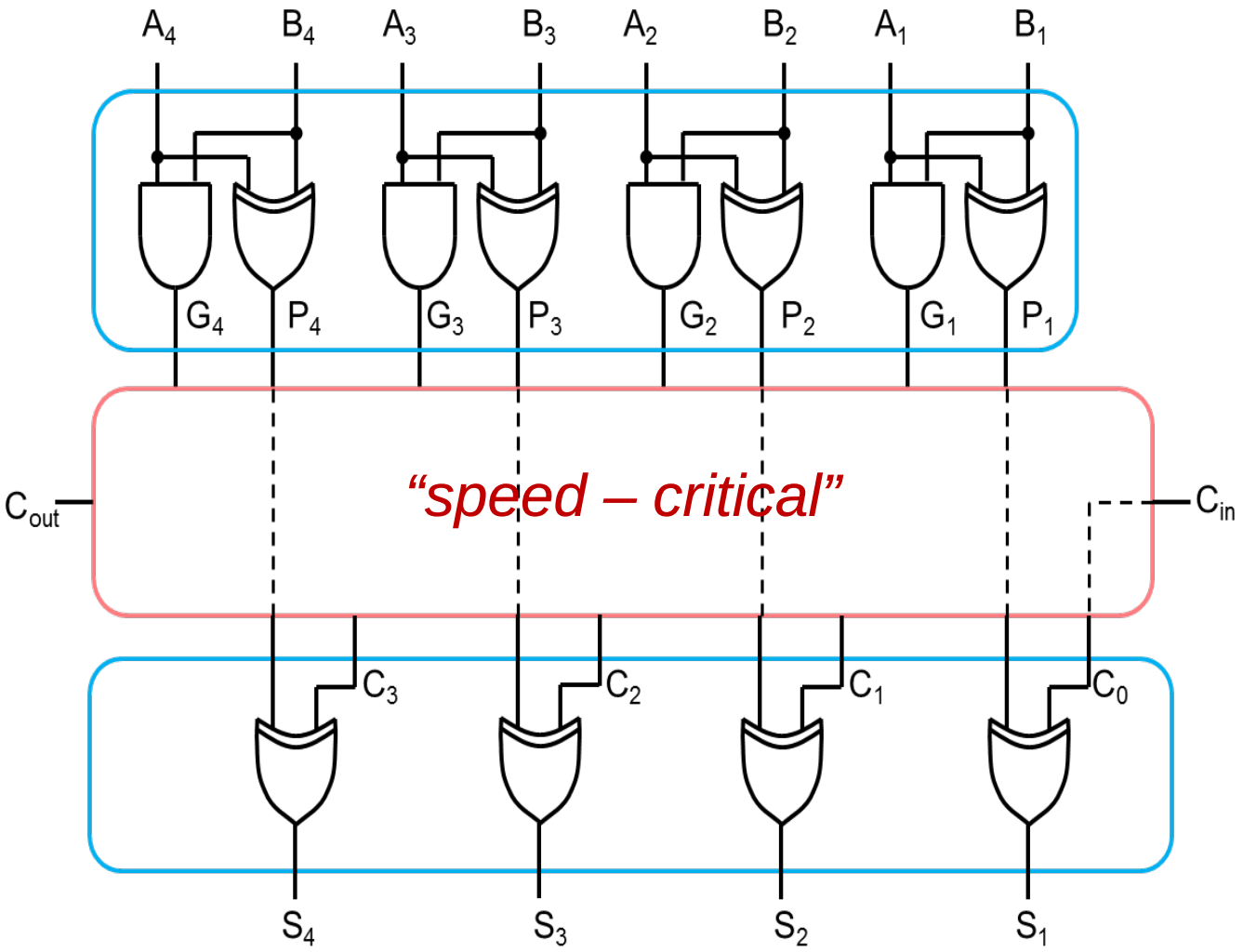
1. Compute bit-wise generate, propagate (& kill) signals

   $G_i = A_i \bullet B_i$ $\quad P_i = A_i \oplus B_i$ $\quad K_i = \overline{A_i} \bullet \overline{B_i}$

2. Use PG(K) signals and C_{in} to determine C_i for each bit (and C_{out}) *(this is the only speed-critical operation)*

3. Calculate sums using $S_i = P_i \oplus C_i$

Group Addition with PG Logic



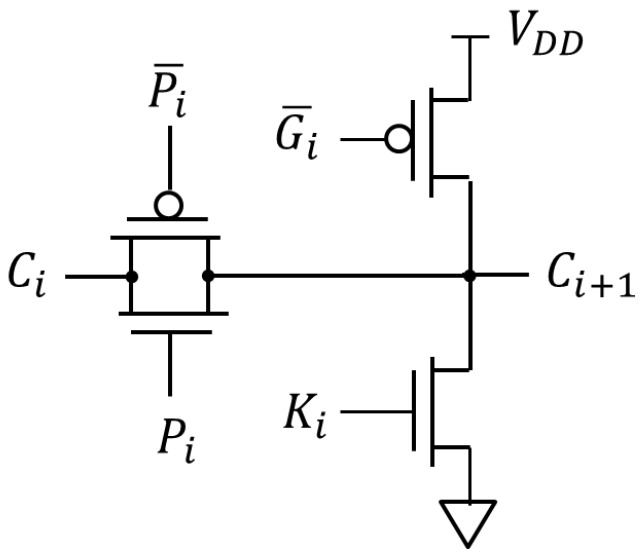
1: Bitwise PG Logic

2: Group Carry Logic

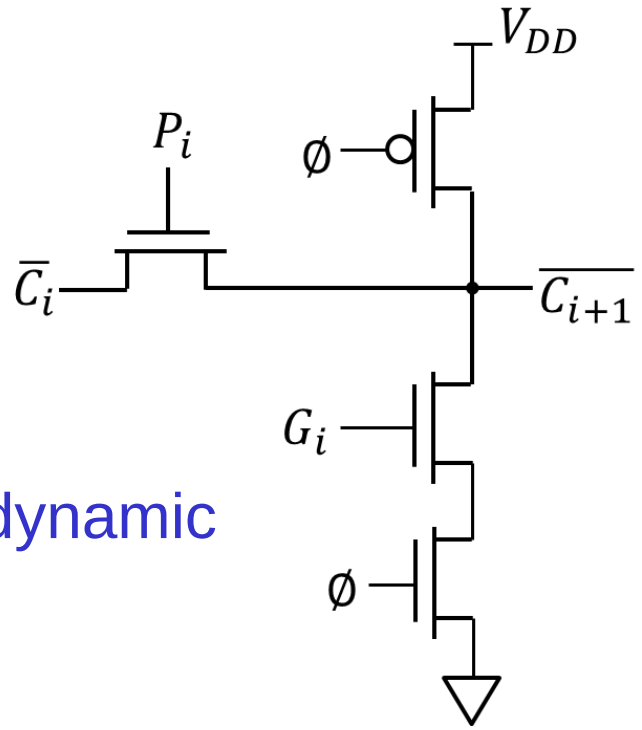
3: Bitwise Sum Logic

Group Carry Logic: Manchester Carry

- Use transmission gates to provide carry propagation

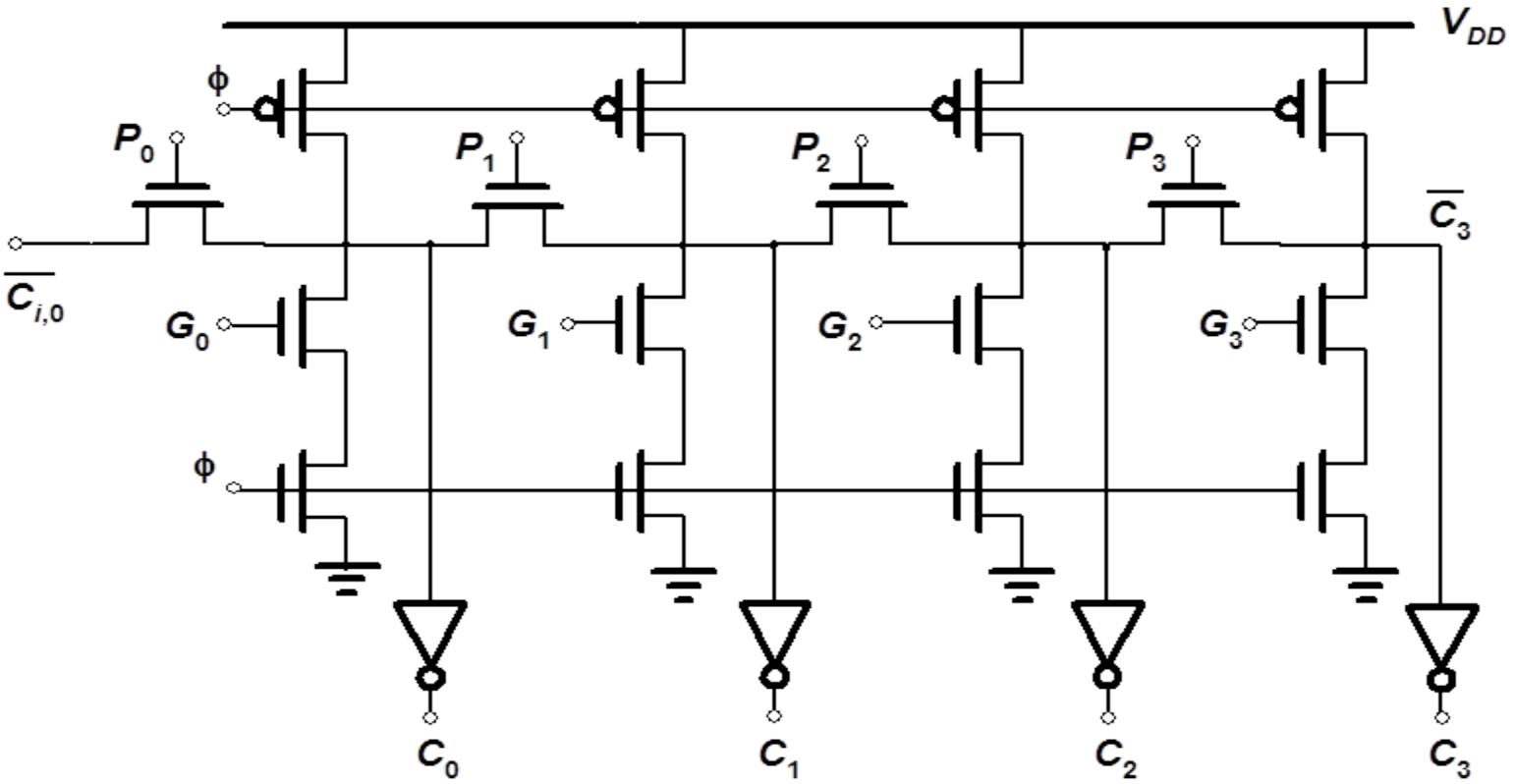


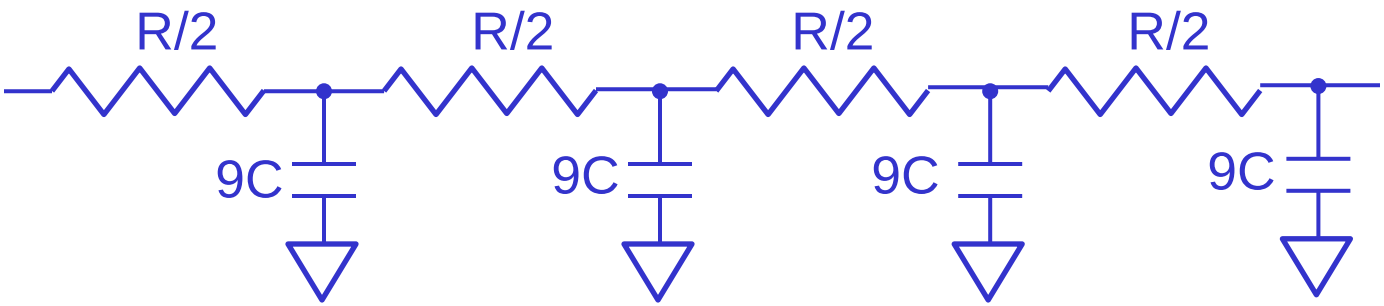
static



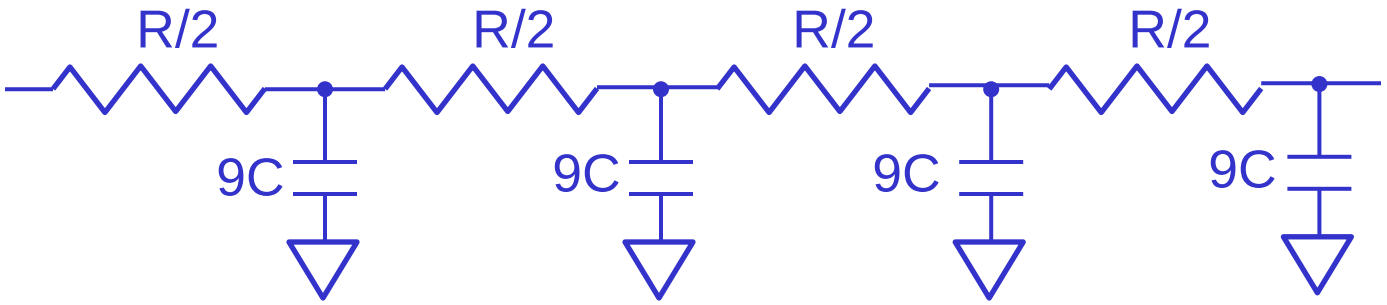
dynamic

4-bit Manchester Carry Logic





Delays in Manchester Chain

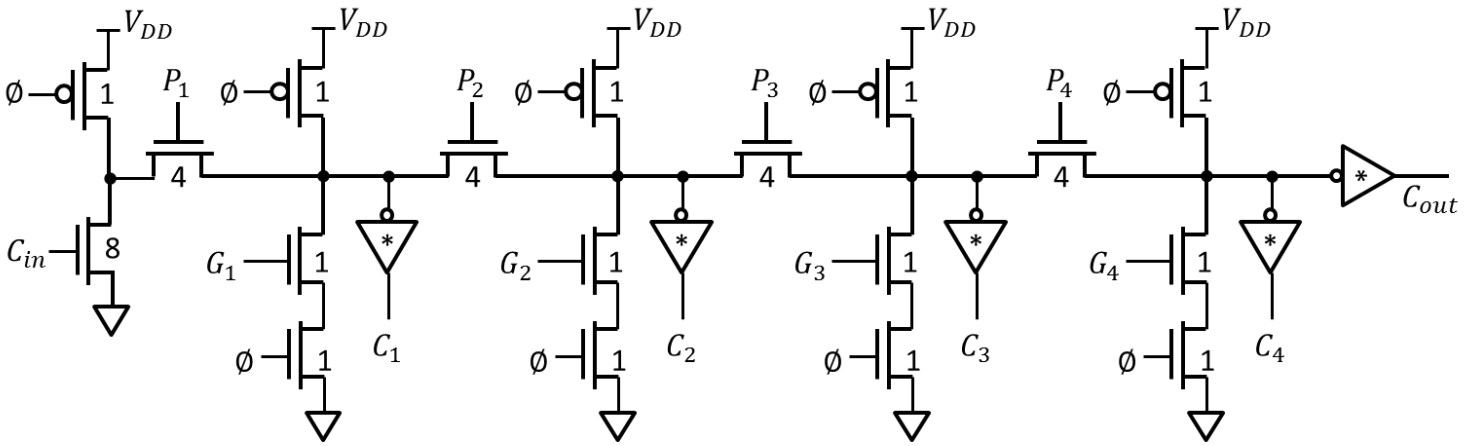


- Using Elmore, delay (after *n* stages) = *(9/4).n(n+1)RC*
- Delay increases quadratically with *n*

n	1	2	3	4
total delay	4.5 RC	13.5 RC	27 RC	45 RC
delay of extra stage		9 RC	13.5 RC	18 RC

- Better to add a couple of inverters after 3-4 bits
  - makes overall delay linear in *n*

Manchester Chain with pre and post-inverter

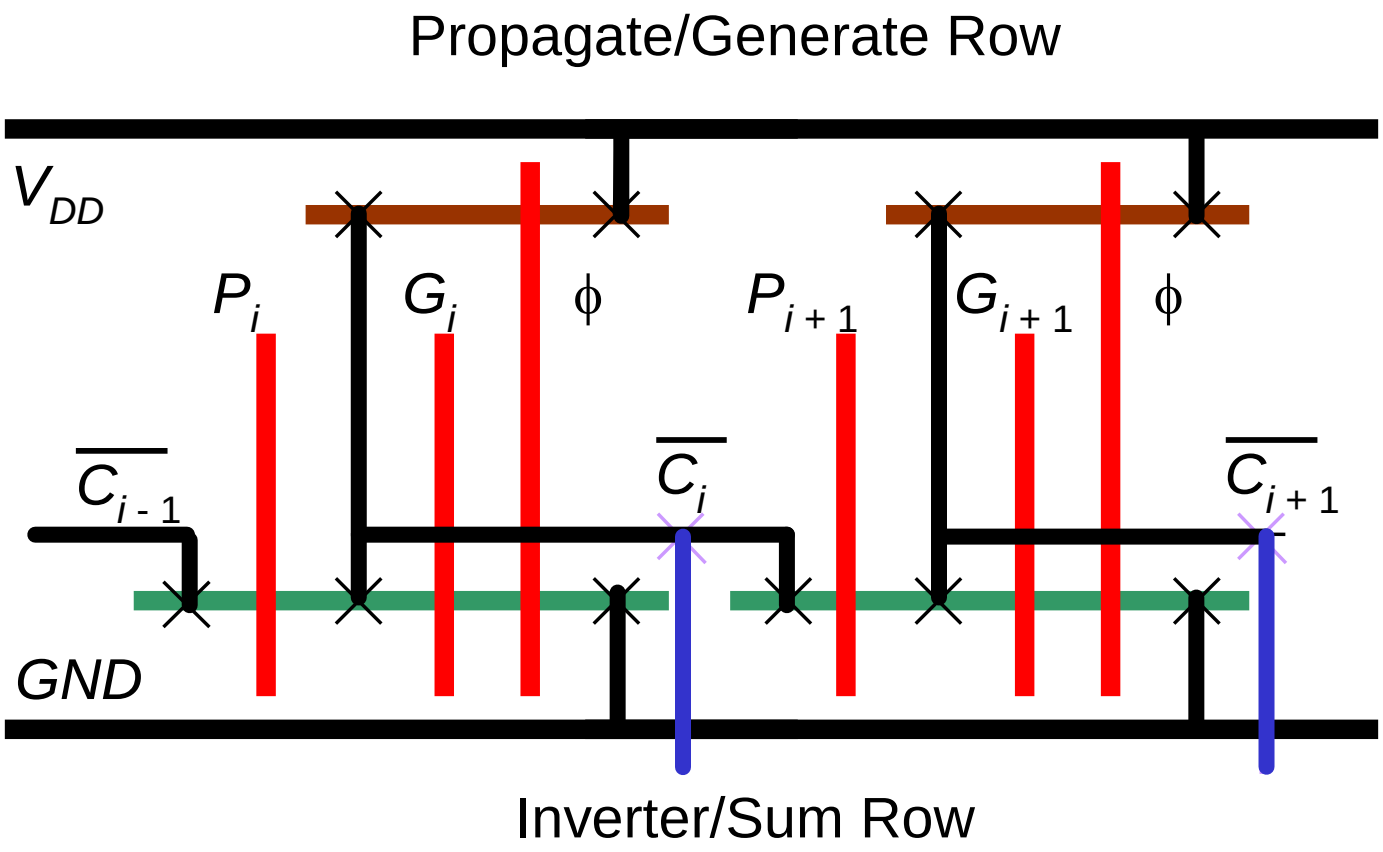


* each inverter has P=1, N=1

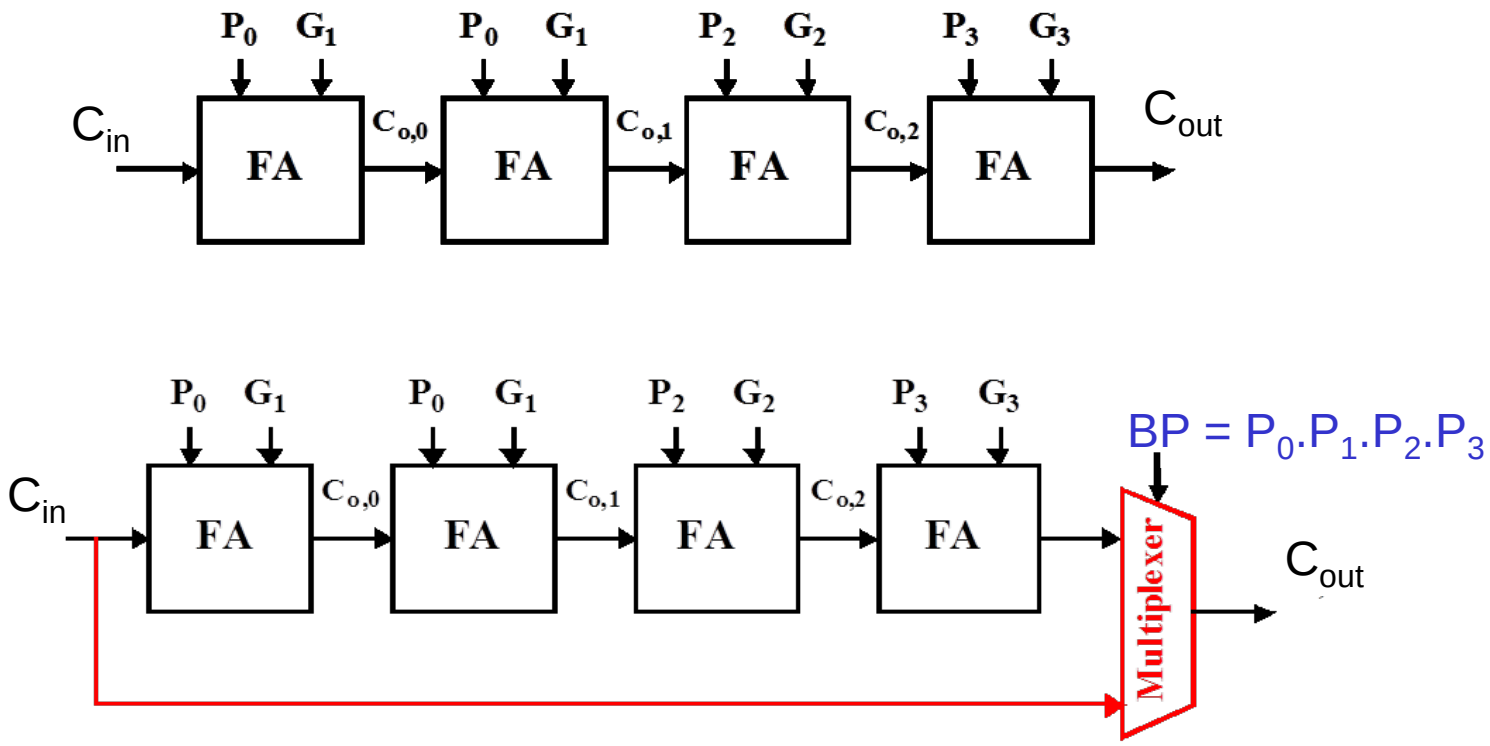
number of bits	1	2	3	4	5	6	7	8
total delay (RC)	16.8	22.5	30.3	40	51.8	65.5	81.3	99.0
delay per bit (τ)	5.6	3.8	3.4	3.3	3.5	3.6	3.9	4.1

- Compare to Mirror Adder Ripple Carry = 5.5τ per bit

Manchester Carry Stick Layout

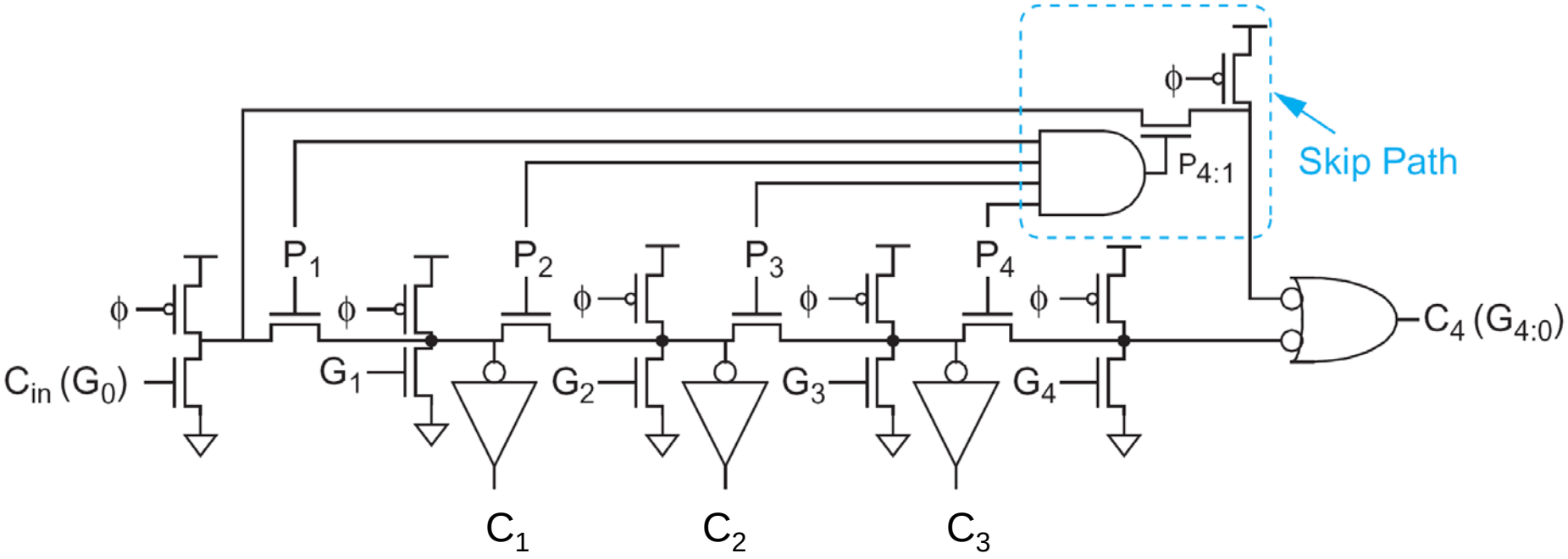


Carry-Bypass Adder

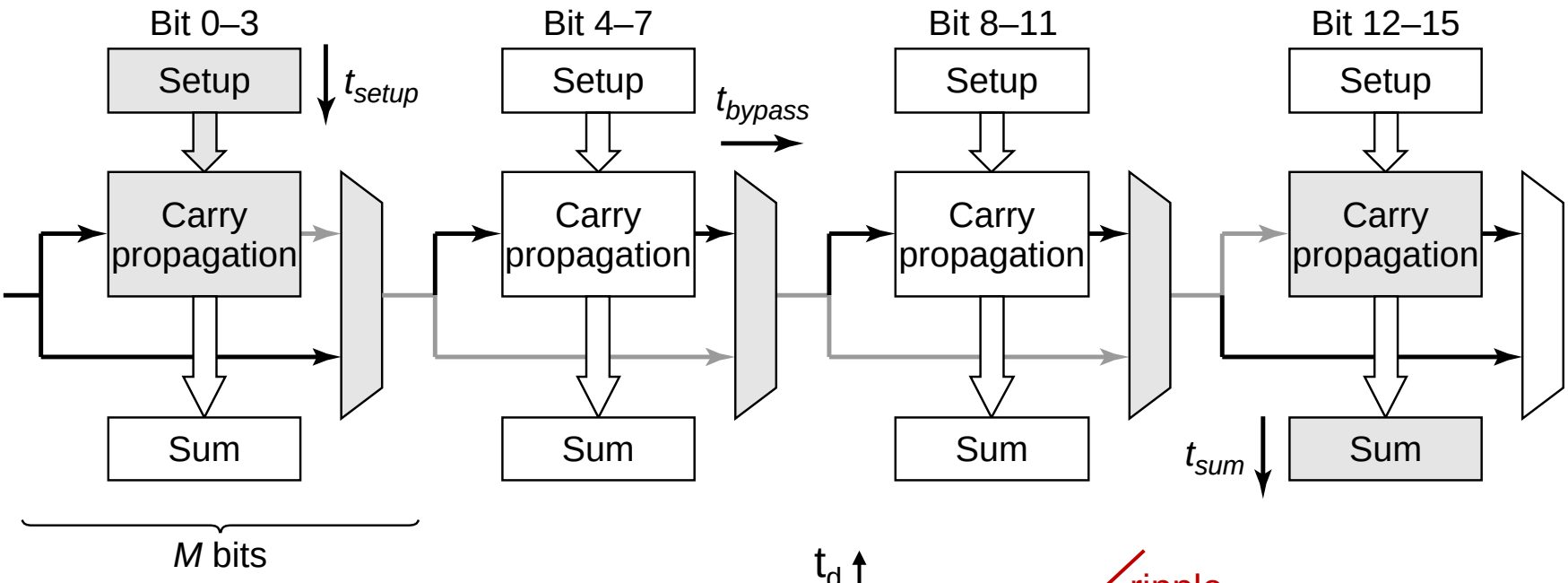


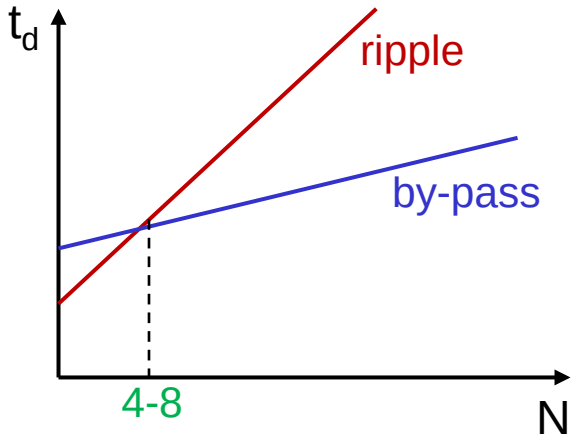
- If (P0 and P1 and P2 and P3) then $C_{out} = C_{in}$
- Otherwise use PG within the block
- In an large adder with many blocks, BP is set up well before C_{in} arrives
- Also known as Carry-Skip Adder

Carry-Bypass Manchester Block



Carry-Bypass Critical Path





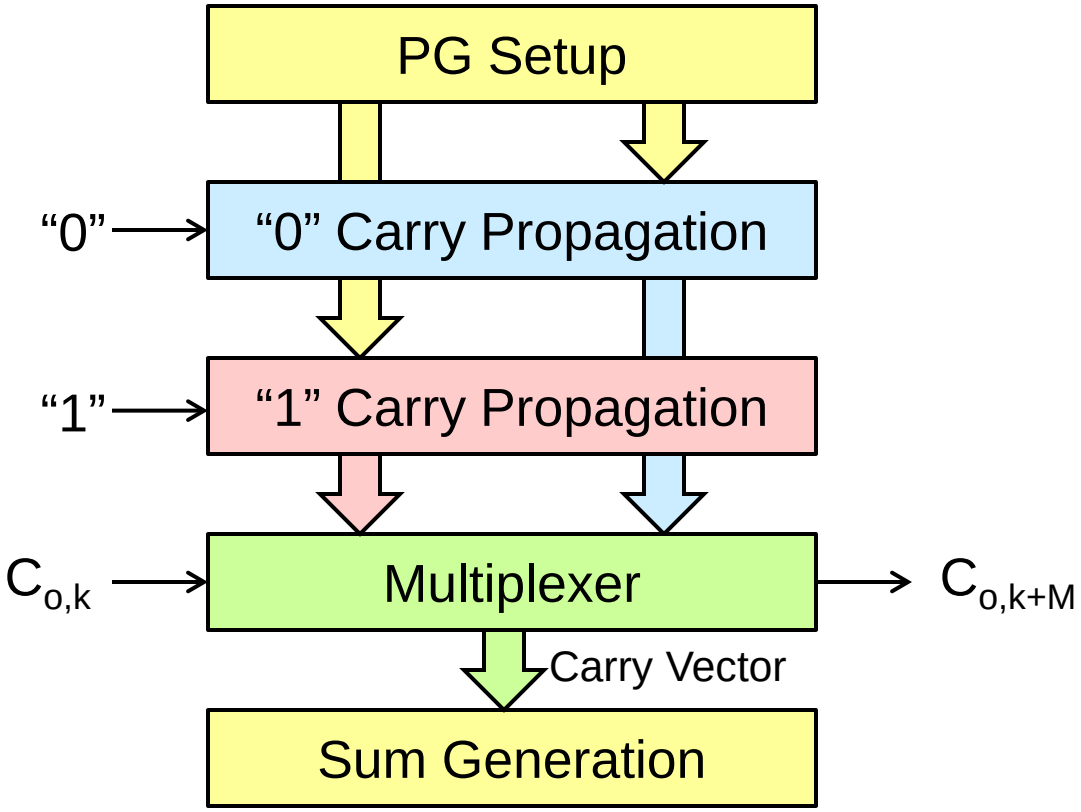
If we have *N* bits,
M bits/block,
N/M blocks,

worst case delay is

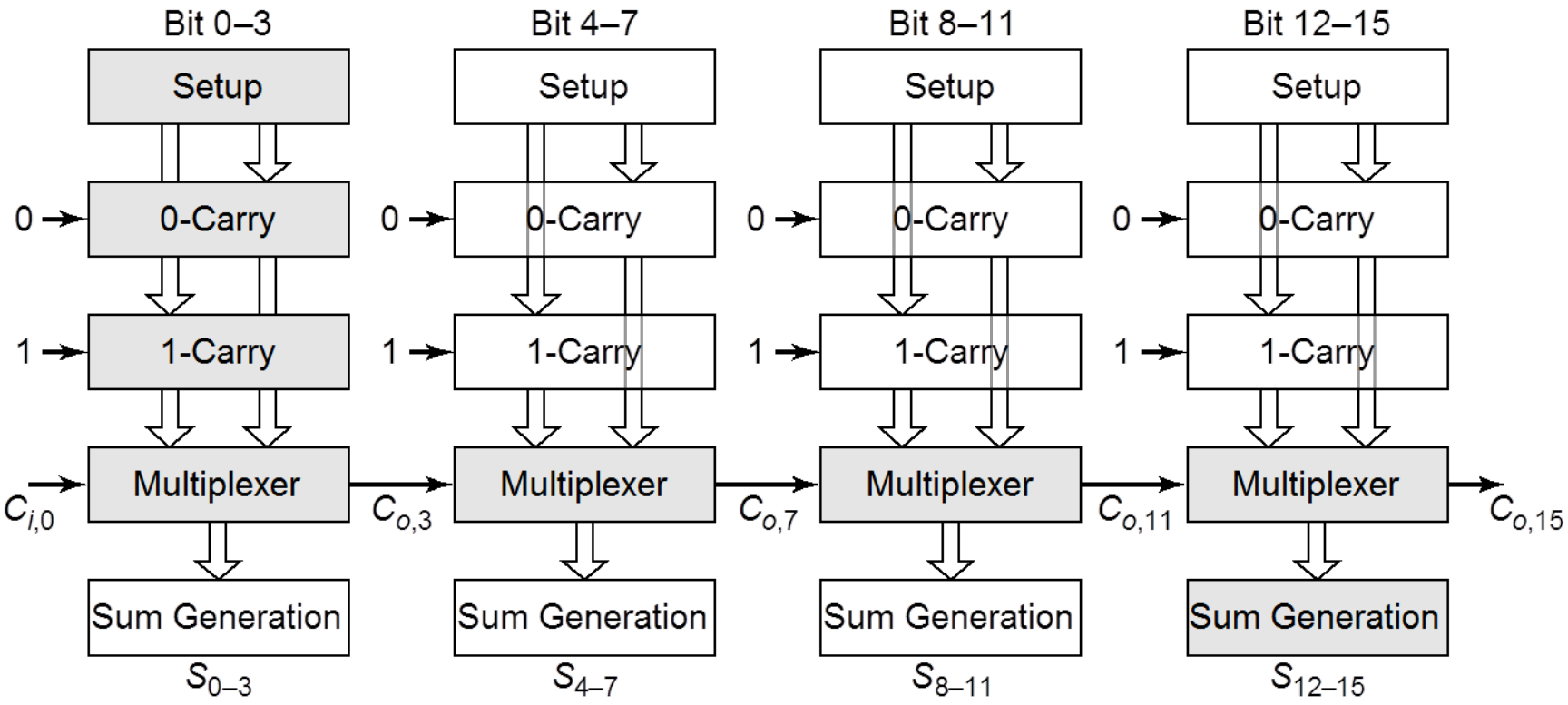
$$t_{adder} = t_{setup} + M.t_{carry} + (N/M\text{-}1).t_{bypass} + (M\text{-}1).t_{carry} + t_{sum}$$

Carry-Select Adder

- For each M-bit block:
- Calculate block carries for both C_{in}=0 and C_{in}=1
- Then when C_{in} finally arrives, use multiplexer to select correct result



Carry-Select Adder – Critical Path

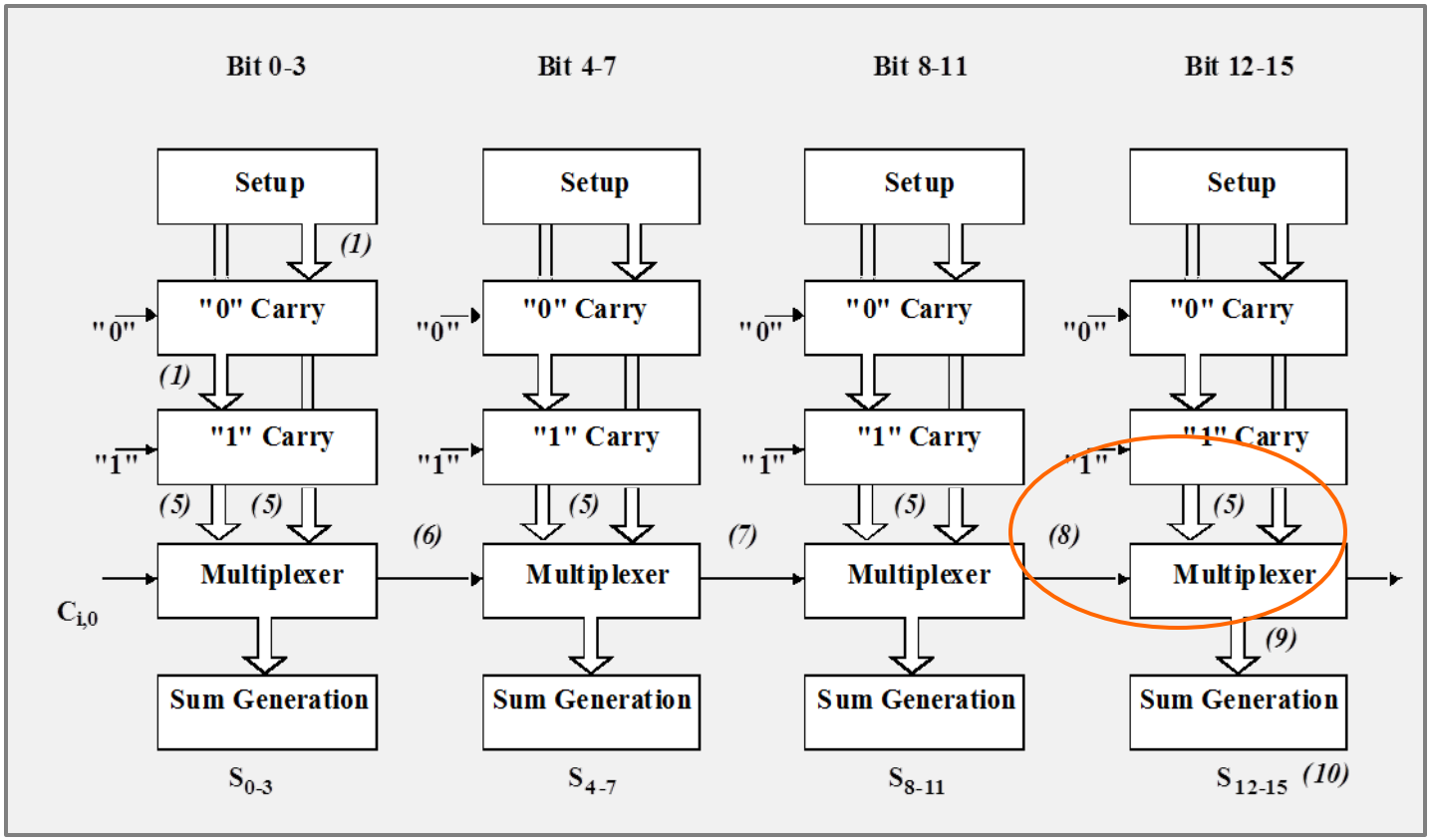


worst case delay is:

$$t_{adder} = t_{setup} + M.t_{carry} + (N/M).t_{mux} + t_{sum}$$

Linear Carry-Select Adder

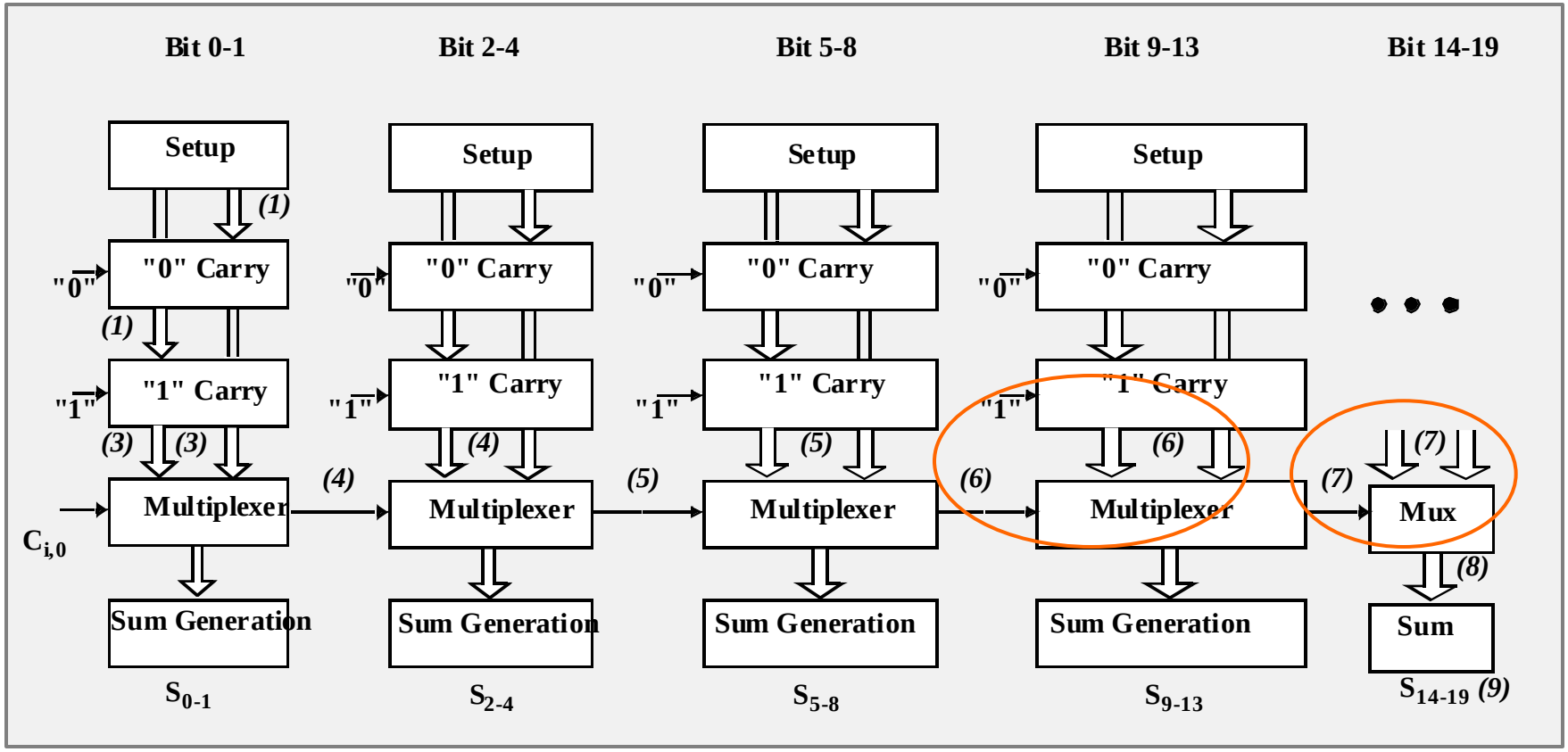
- Lets look at worst case delays in adder with N=16, M=4
- Assume $t_{full\text{-}adder} = t_{multiplexer} = 1$



- For last block, output of “0” and “1” carry sections arrive well before multiplexer select signal from previous block

Square Root Carry-Select Adder

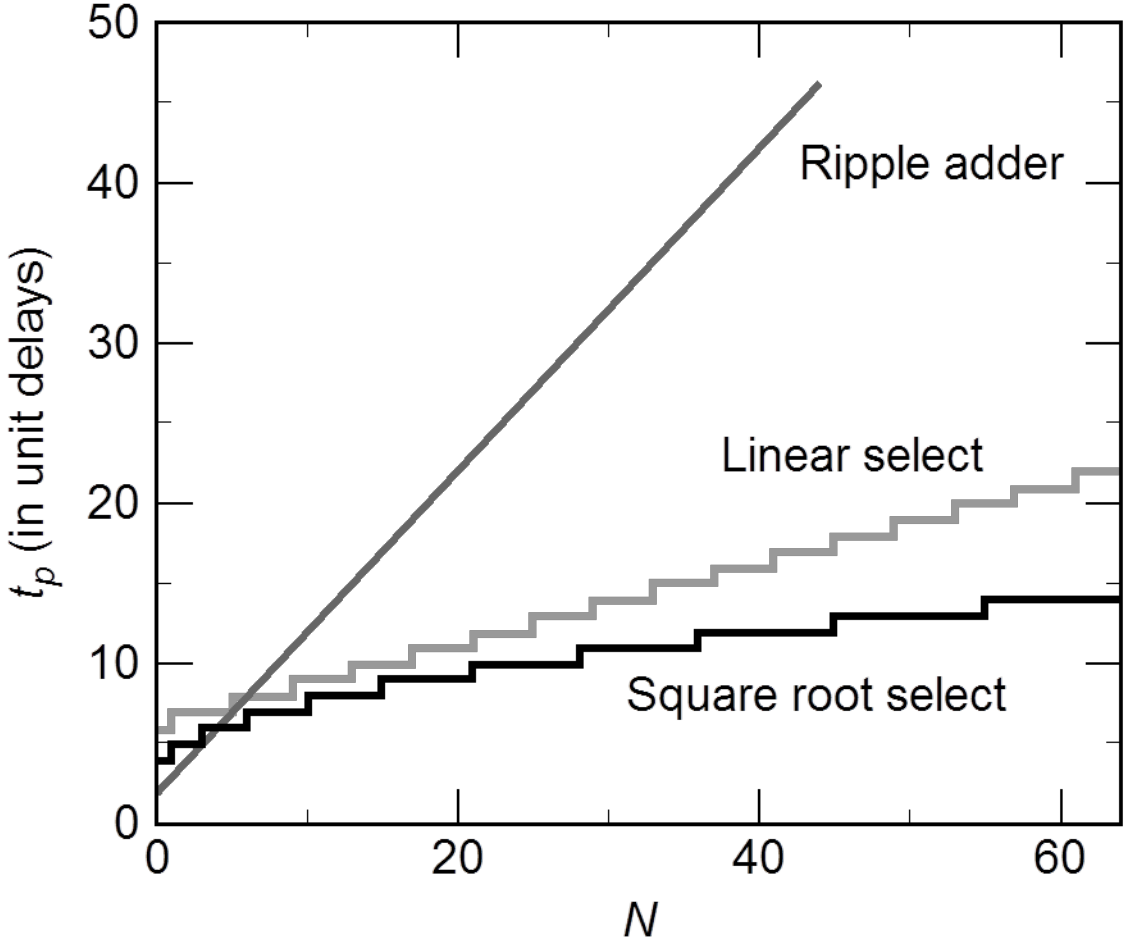
- By making blocks of increasing length, we can perform more carry calculations while waiting for the multiplexer select signal



$$t_{adder} = t_{setup} + M.t_{carry} + (\sqrt{2N}).t_{mux} + t_{sum}$$

Carry-Select Adder: Delay Comparisons

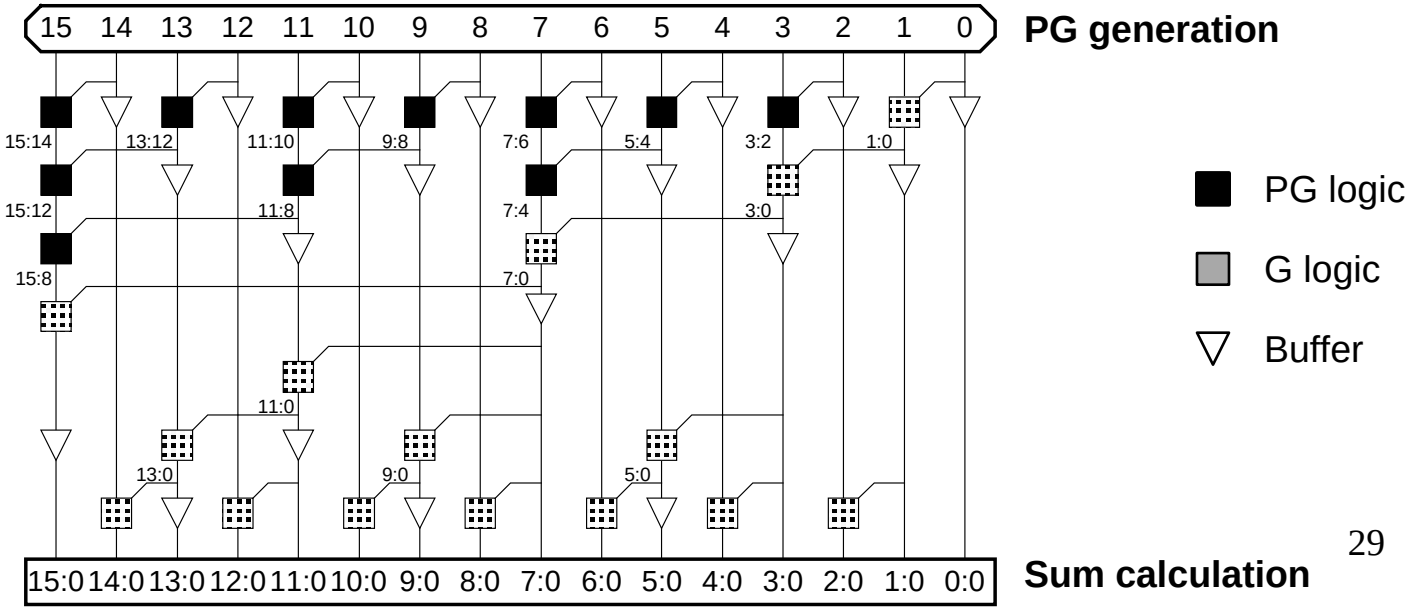
- Square root select particularly effective for large N (e.g. 64-bit)



Tree Adders

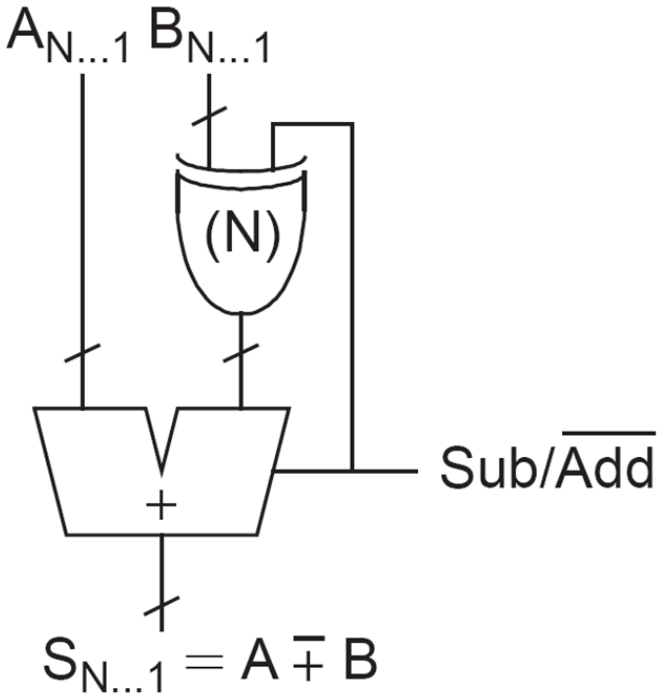
- For wide adders (N>32 bits) delay of carry lookahead (bypass or select) adders is dominated by delay of passing carry through the lookahead stages (multiplexers).
- This delay can be reduced by recursively looking ahead across lookahead blocks, e.g.
  - lookahead across 2-bit blocks to generate C_{in} to 4-bit blocks
  - lookahead across 4-bit blocks to generate C_{in} to 8-bit blocks, etc.
- Delay can *O(log N)* *(at expense of area and power!)*

e.g. Brent-Kung Adder



Subtraction

- A – B = A + (-B) (where –B is two's complement of B)
- -B = NOT(B) + 1



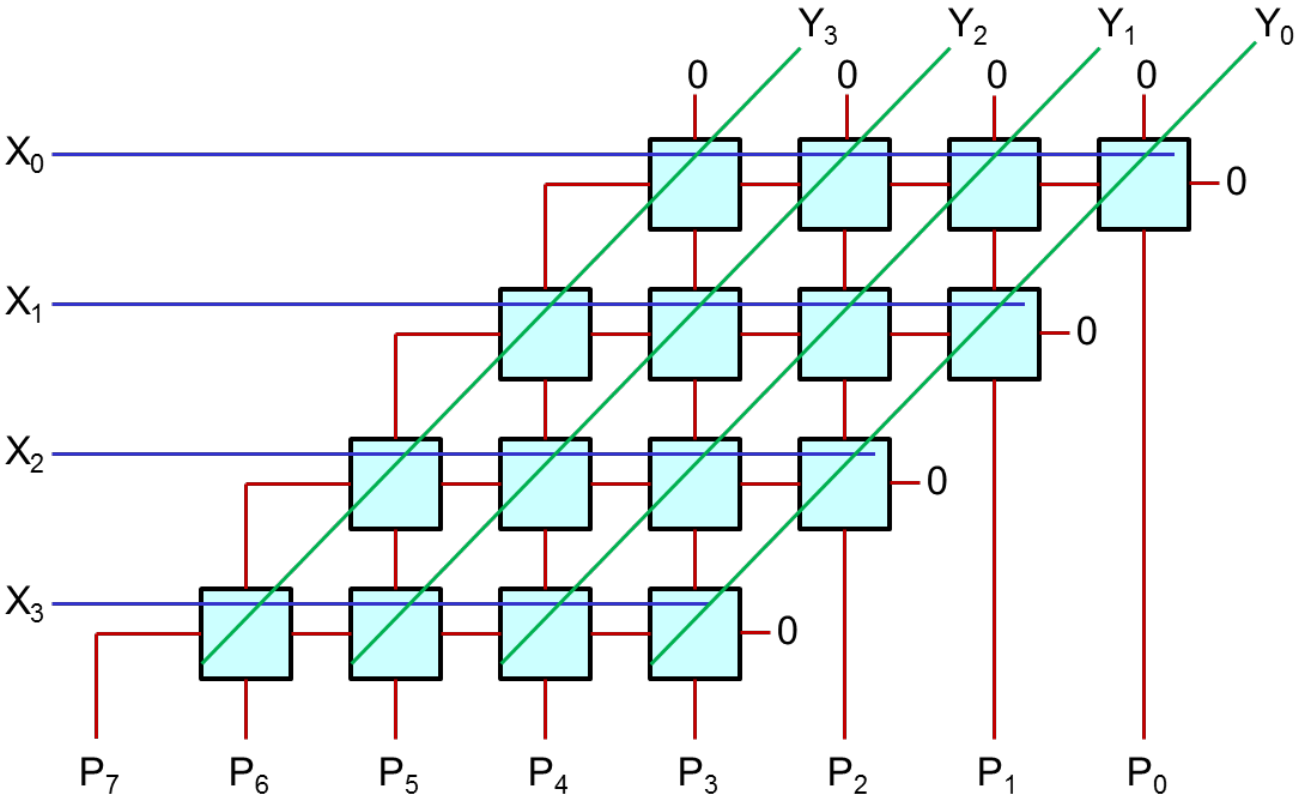
Unsigned Multiplication

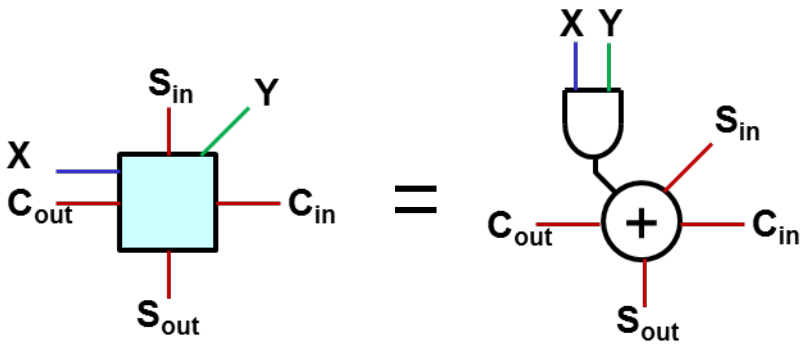
- Example:

	1 1 0 0	$:12_{10}$	multiplicand
X	0 1 0 1	$: 5_{10}$	multiplier
	1 1 0 0		partial products
	0 0 0 0		
	1 1 0 0		
	0 0 0 0		
	0 0 1 1 1 1 0 0	$:60_{10}$	product

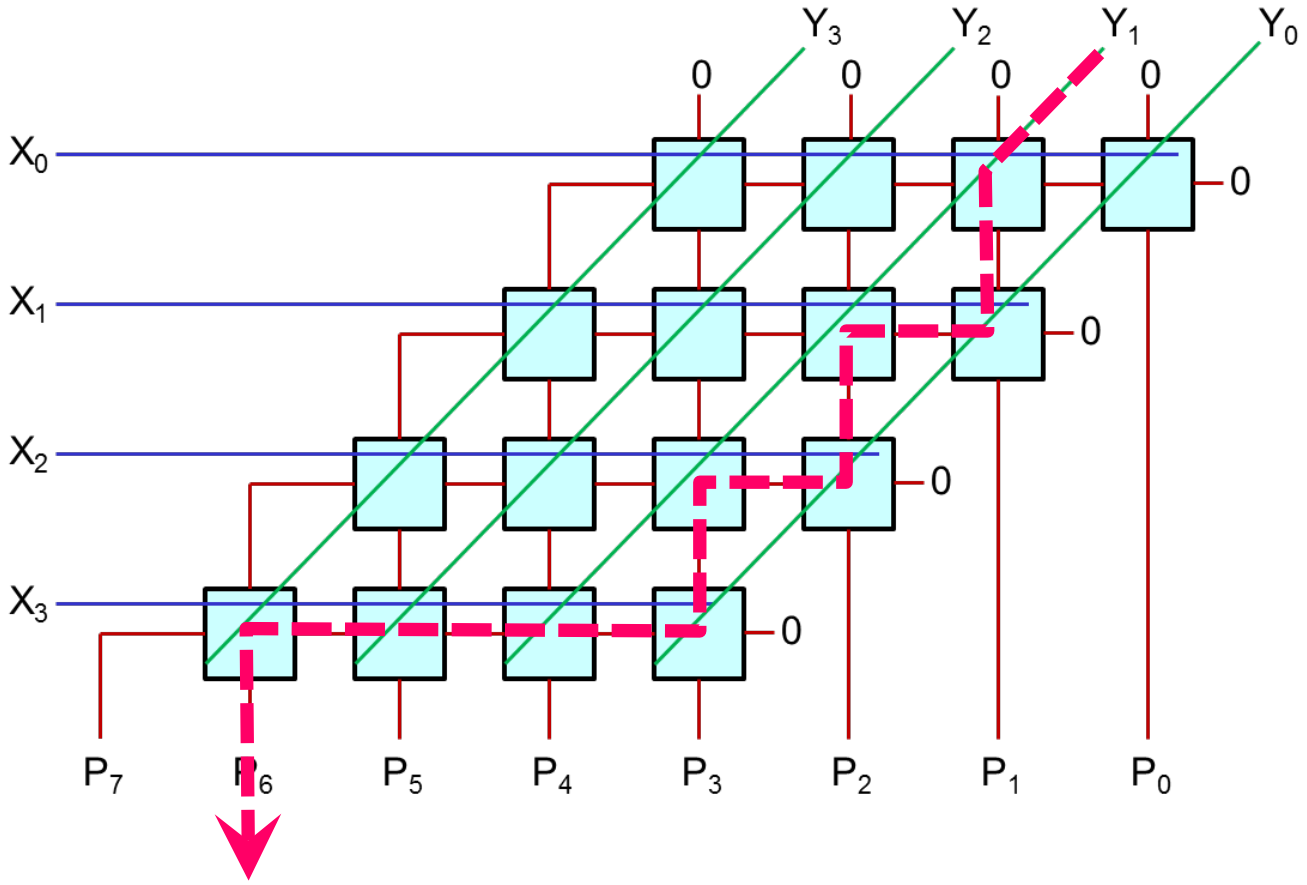
- M x N-bit multiplication
  - Produce N M-bit partial products
  - Sum these to produce (M+N)-bit product

Array Multiplier



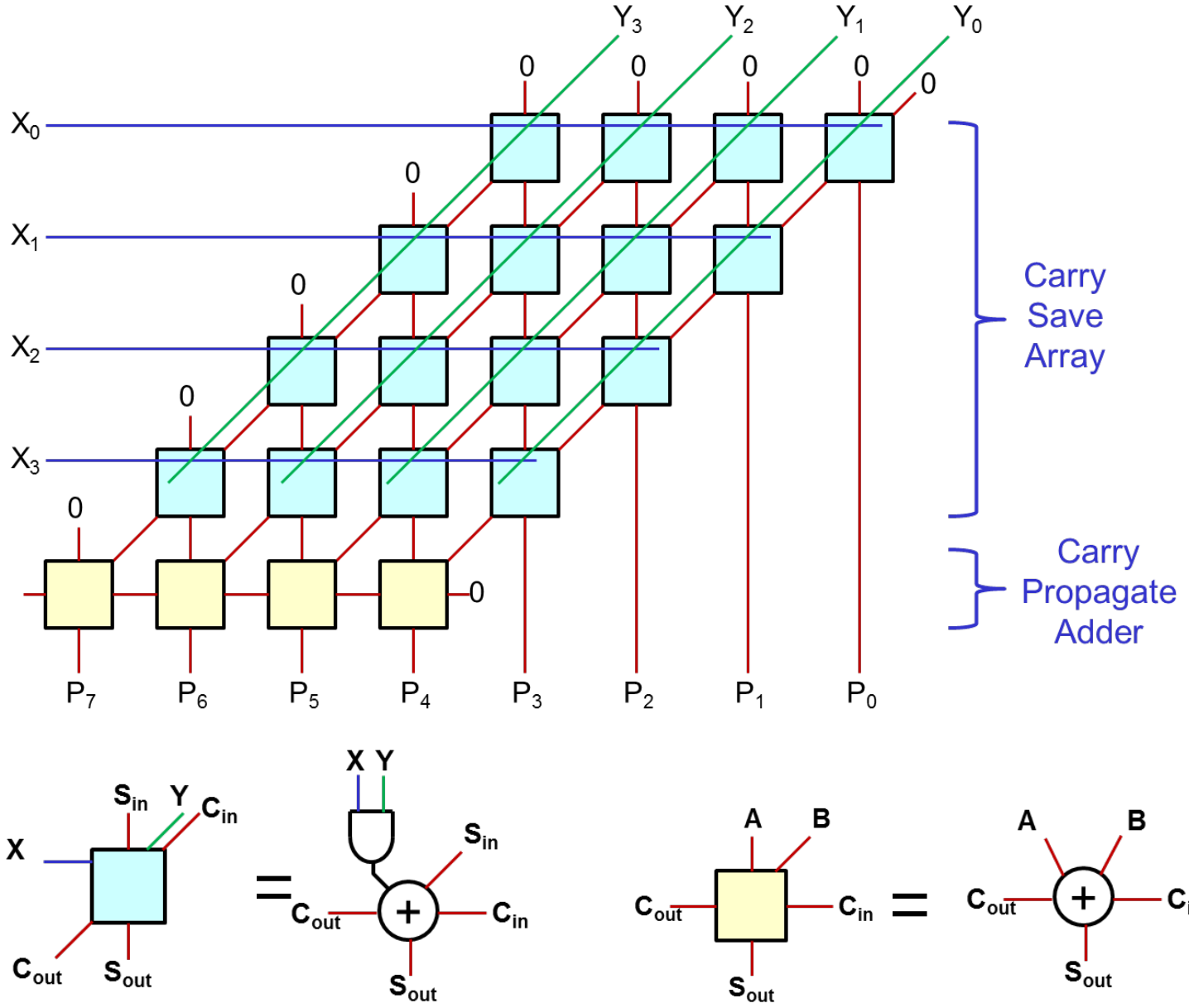


Array Multiplier – Critical Path

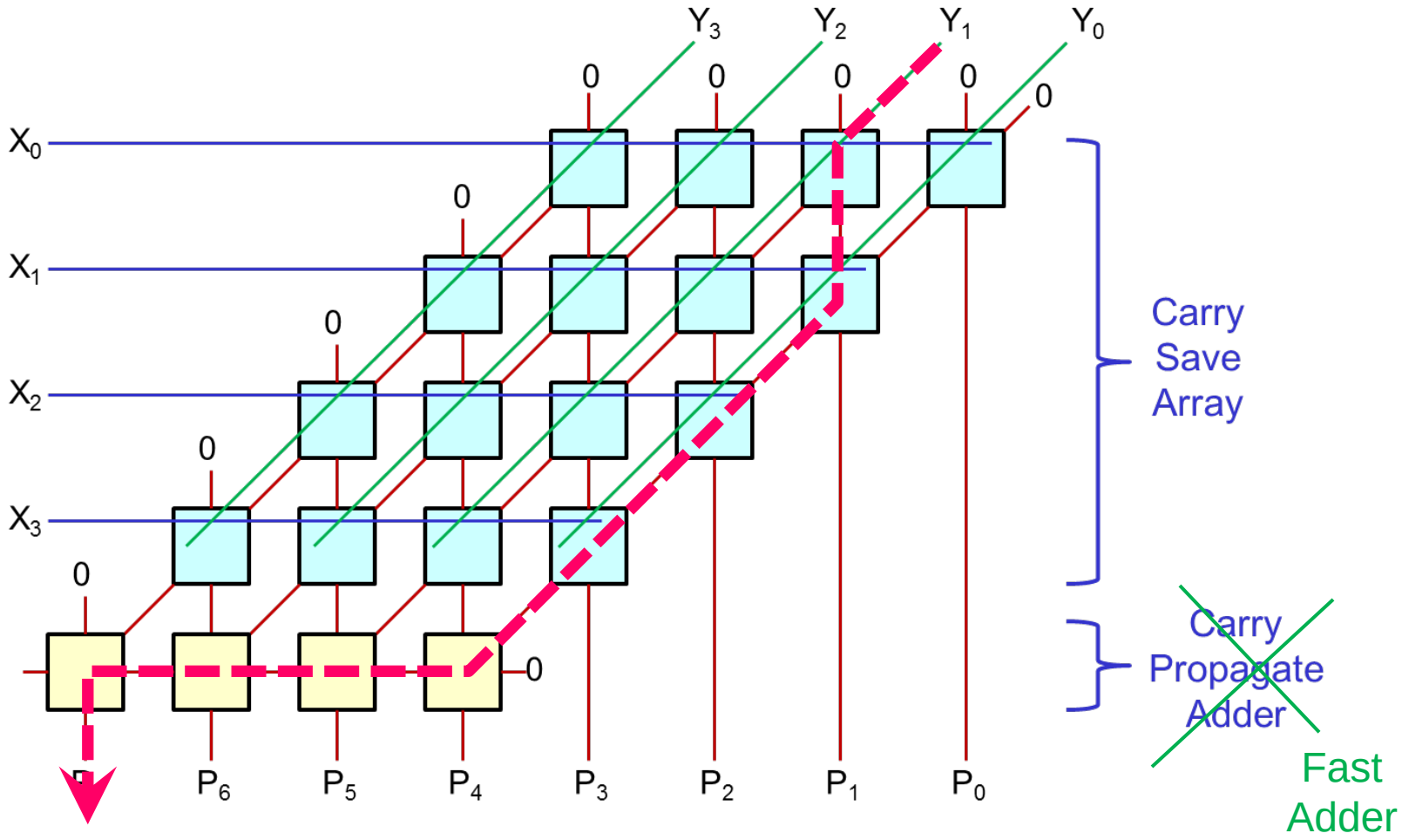


$$t_{mult} = (M+N-3).t_{carry} + N.t_{sum} + t_{AND}$$

Carry Save Multiplier



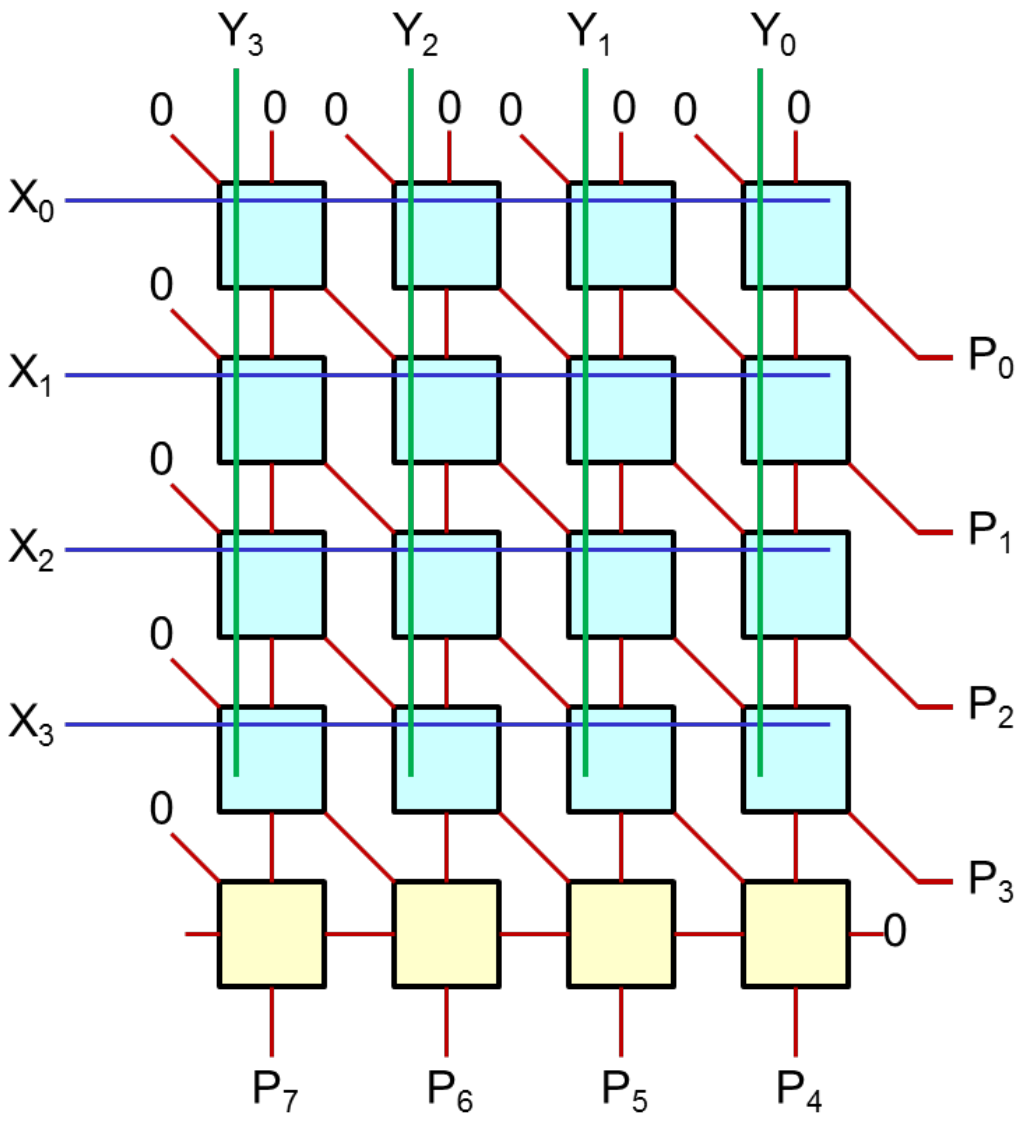
Carry Save Multiplier – Critical Path



$$t_{mult} = (M+N-2).t_{carry} + 2.t_{sum} + t_{AND}$$

OR $$t_{mult} = (N-1).t_{carry} + t_{fast\_adder} + t_{sum} + t_{AND}$$

CSA Multiplier – Compact Layout



Two's Complement (Signed) Multiplication

- In two's complement representation: $X = -x_{N-1}.2^{N-1} + \sum_{i=0}^{N-2} x_i\,2^i$

$$P = Y.X = \left[-y_{M-1}.2^{M-1} + \sum_{j=0}^{M-2} y_j\,2^j\right].\left[-x_{N-1}.2^{N-1} + \sum_{i=0}^{N-2} x_i\,2^i\right]$$

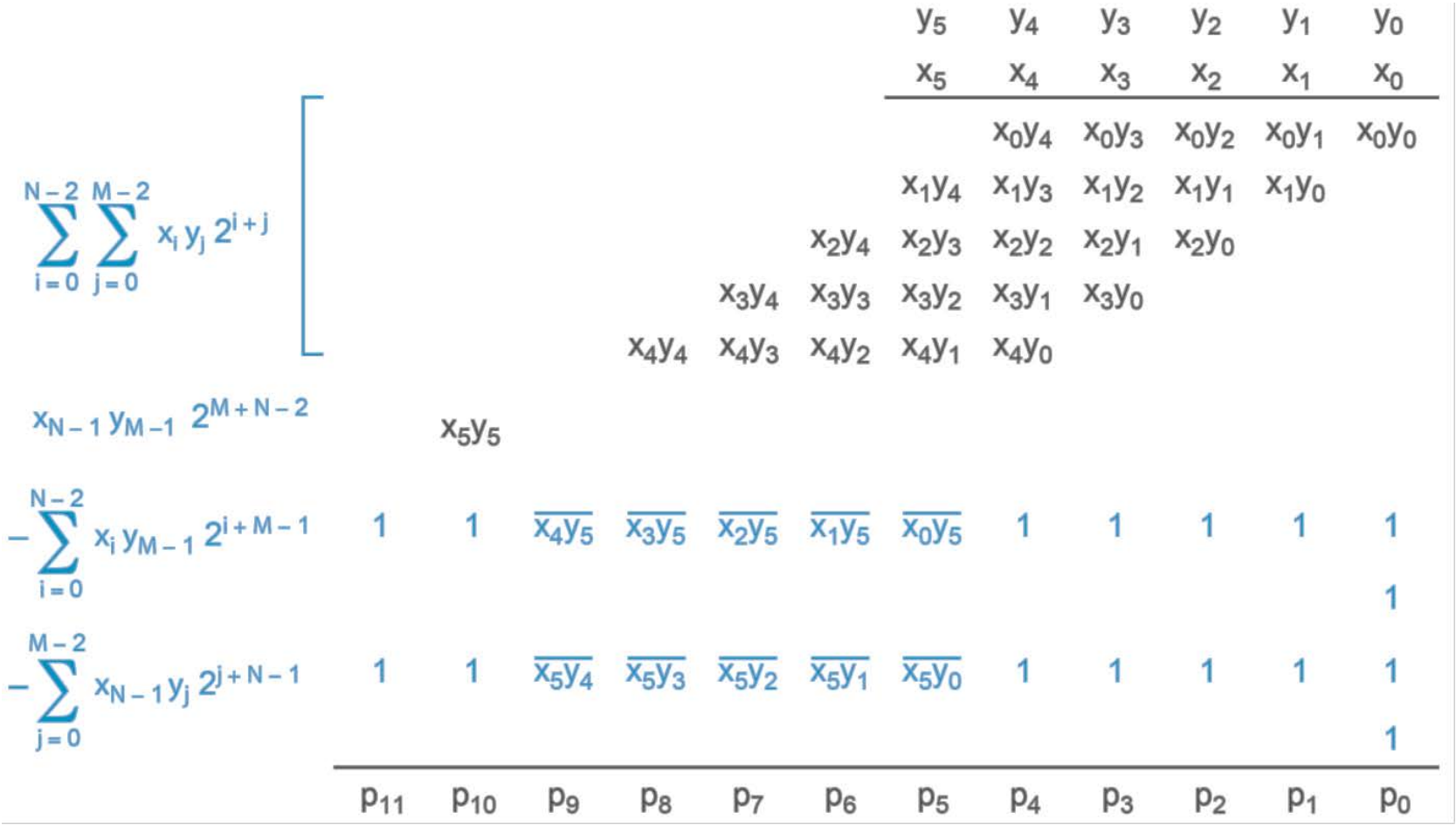
$$= \sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i.y_j.2^{i+j}$$ unsigned (N-1)x(M-1) multiply

$$+ \; x_{N-1}.y_{N-1}.2^{M+N-2}$$ product of MSB's

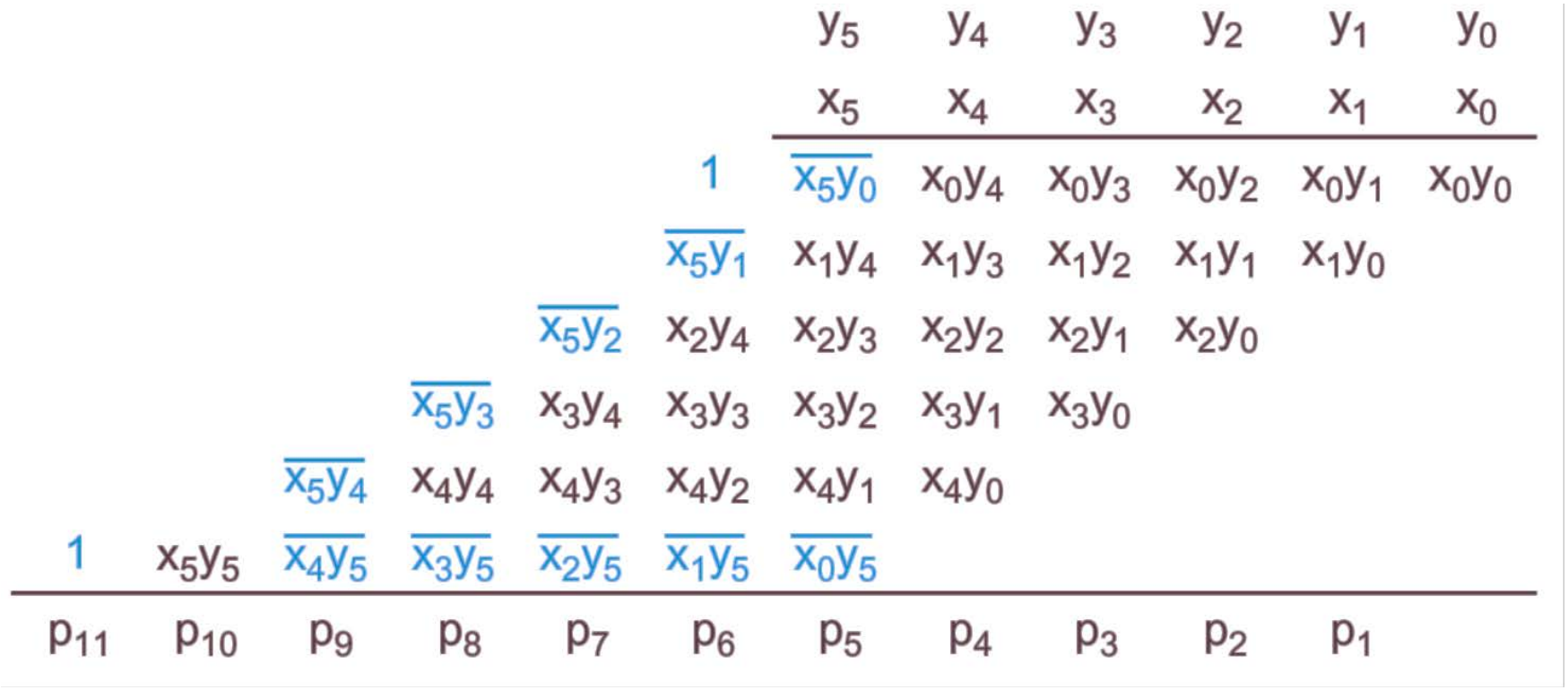
$$- \left[\sum_{i=0}^{N-2} x_i.y_{M-1}.2^{i+M-1} + \sum_{j=0}^{M-2} x_{N-1}.y_j.2^{j+N-1}\right]$$ two terms to be subtracted

Baugh-Wooley Partial Products

- Subtraction of these terms is accomplished by adding two's complement, i.e. by adding ($\overline{term}$ +1)

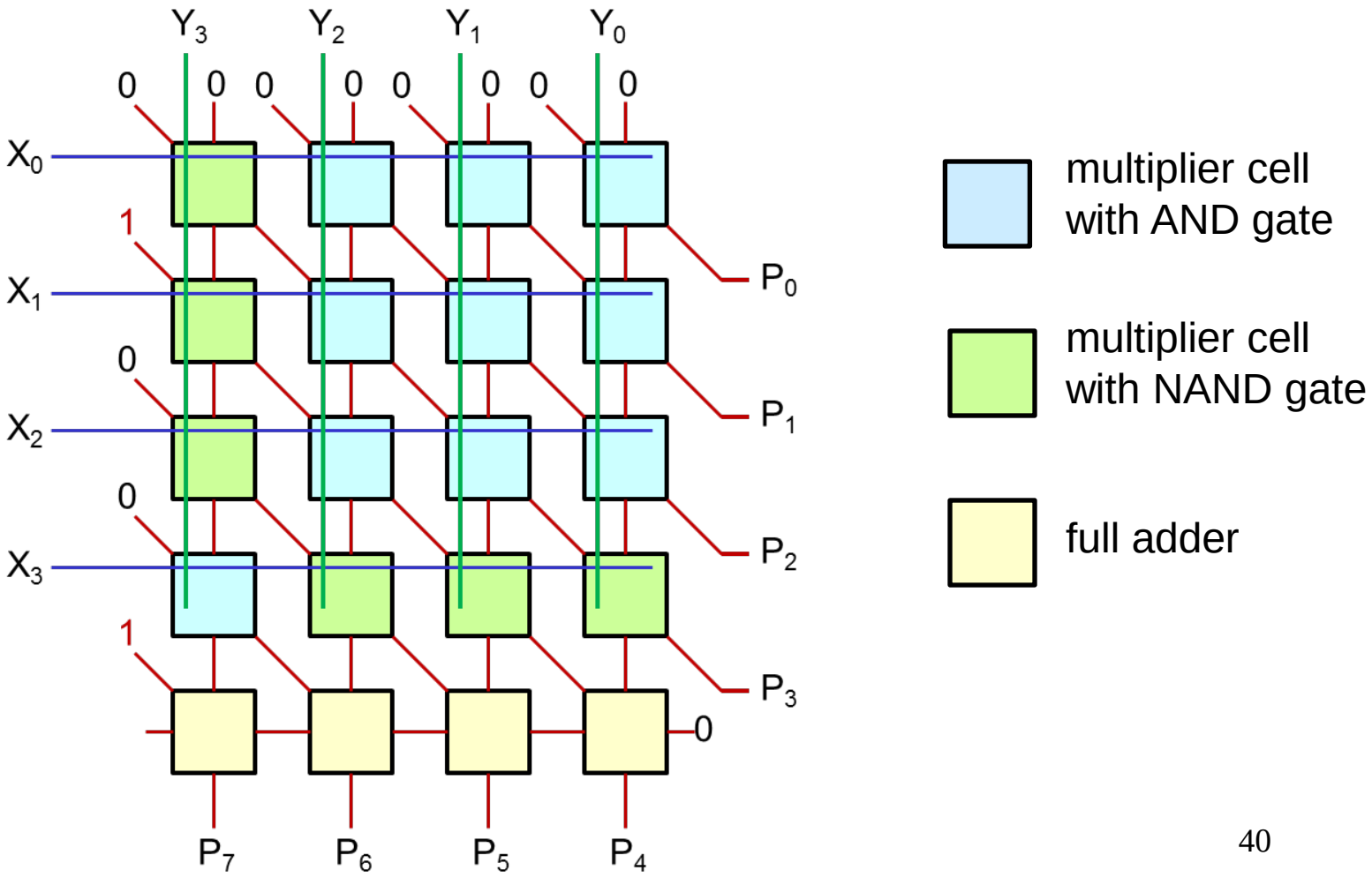


Baugh-Wooley Multiplication Array



Modified Baugh-Wooley Multiplier

- Simply replace AND gate in these cells with NAND gate
  - and set two of the carry-in constants to '1'



Faster Multipliers

- Multiplication is key element in many DSP applications
  - Digital Filters
  - Transforms
  - Modulation & Correlation
- Many architectures have been proposed to speed up multiplication
  - radix-4 Booth encoding
  - Wallace tree
  - Compressor trees
  - Pipelining
  - Various combinations of above
- Each starts by examining critical path and looking for ways to “short-circuit” computation
- Each provides improved speed at cost of area & power