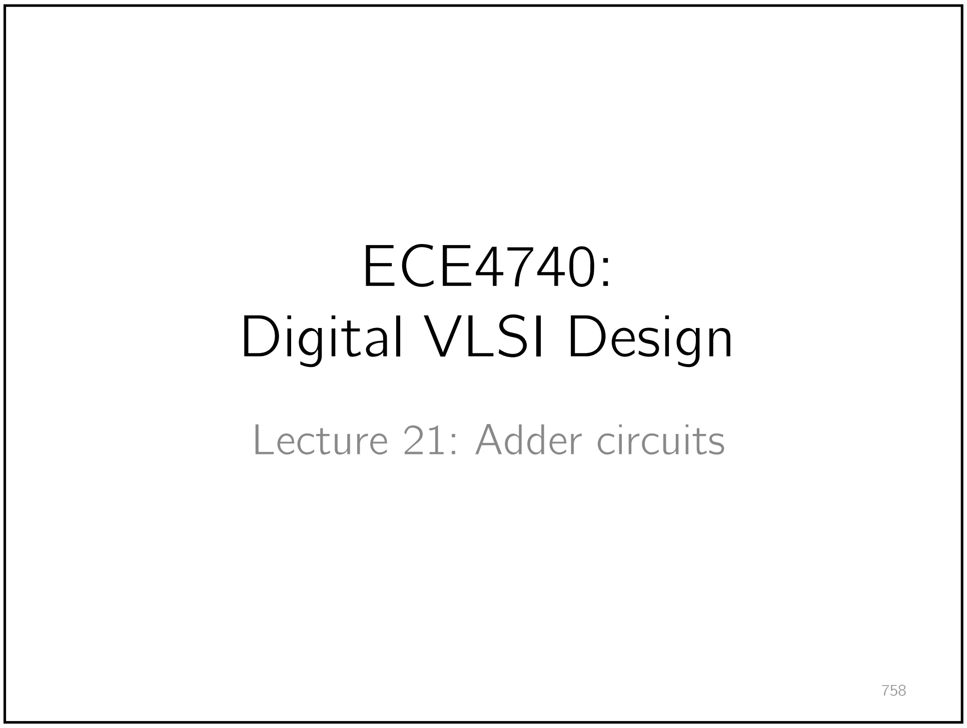
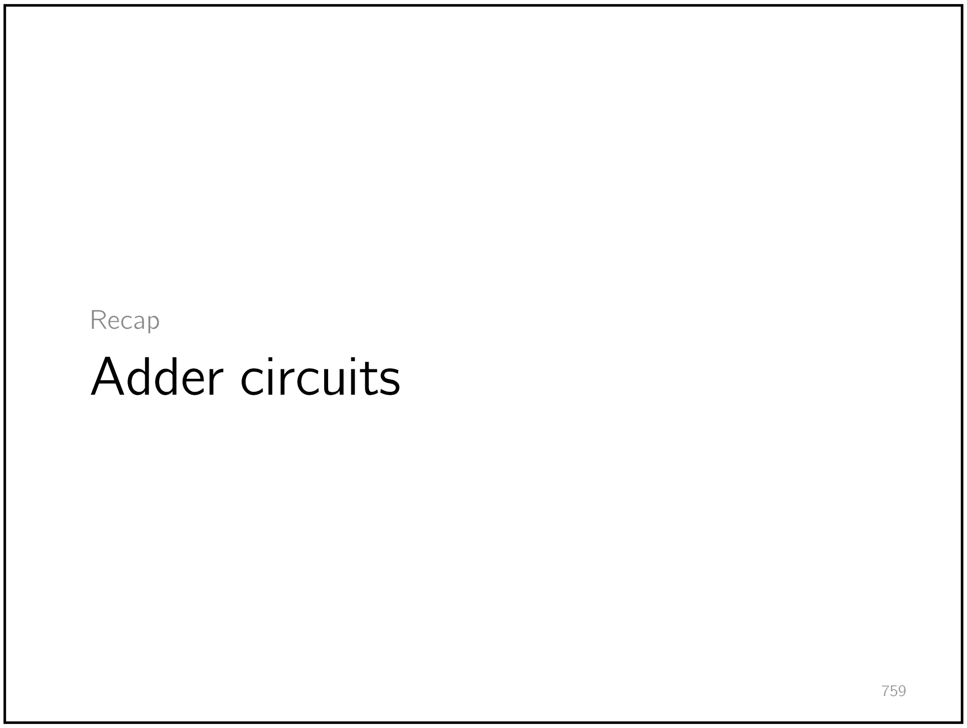


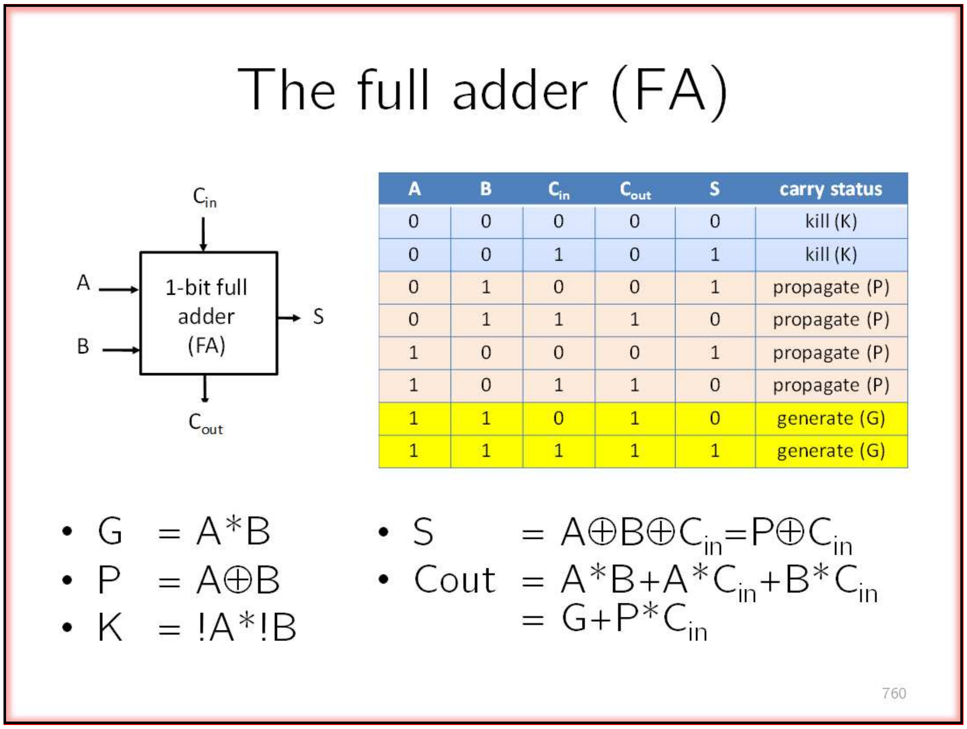
ECE4740: Digital VLSI Design

Lecture 21: Adder circuits



Recap

Adder circuits

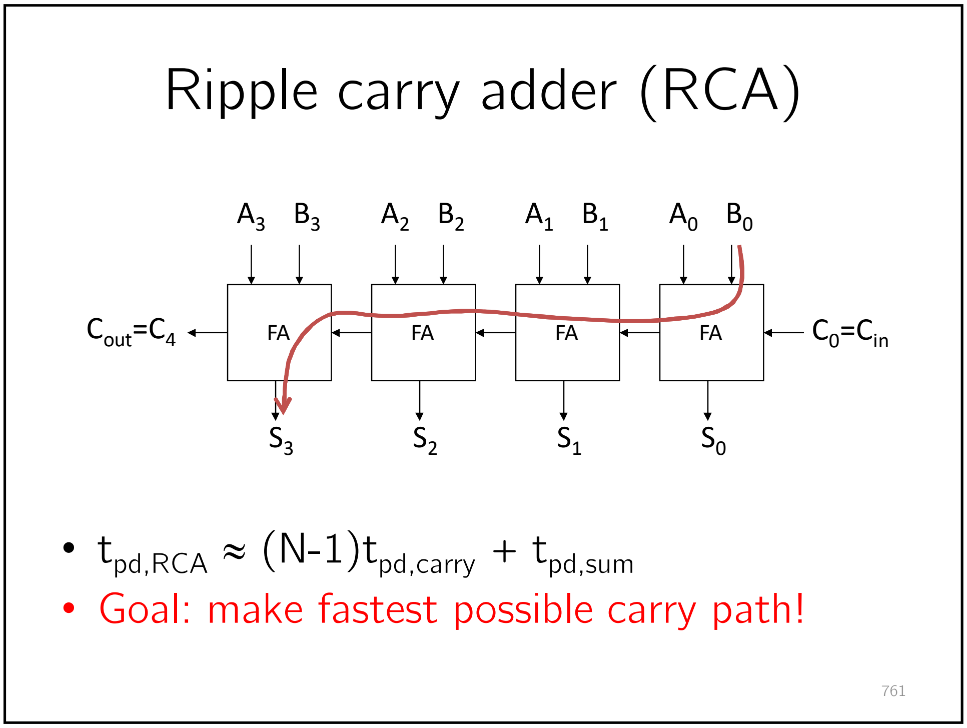


The full adder (FA)

A	B	C_{in}	C_{out}	S	carry status
0	0	0	0	0	kill (K)
0	0	1	0	1	kill (K)
0	1	0	0	1	propagate (P)
0	1	1	1	0	propagate (P)
1	0	0	0	1	propagate (P)
1	0	1	1	0	propagate (P)
1	1	0	1	0	generate (G)
1	1	1	1	1	generate (G)

- $G = A*B$
- $P = A\oplus B$
- $K = !A*!B$

- $S = A\oplus B\oplus C_{in} = P\oplus C_{in}$
- $Cout = A*B + A*C_{in} + B*C_{in}$
  $= G + P*C_{in}$



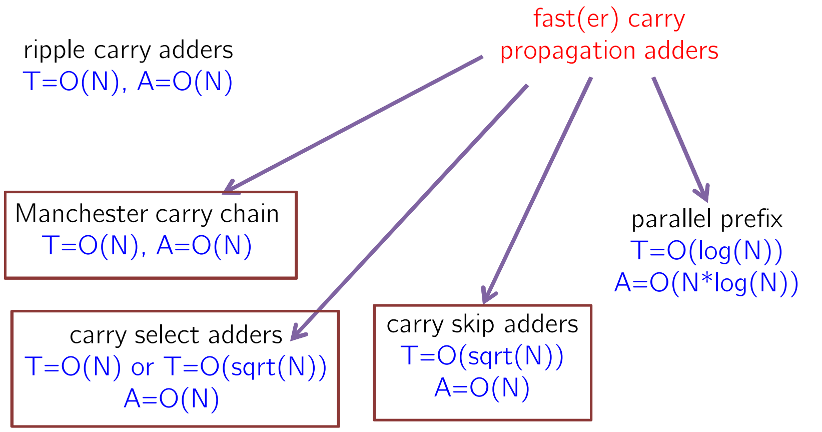
Ripple carry adder (RCA)

- $t_{pd,RCA} \approx (N\text{-}1)t_{pd,carry} + t_{pd,sum}$
- Goal: make fastest possible carry path!

We can make adders even faster!

Fast carry-chain designs

Overview: some binary adder circuits



ripple carry adders
T=O(N), A=O(N)

fast(er) carry
propagation adders

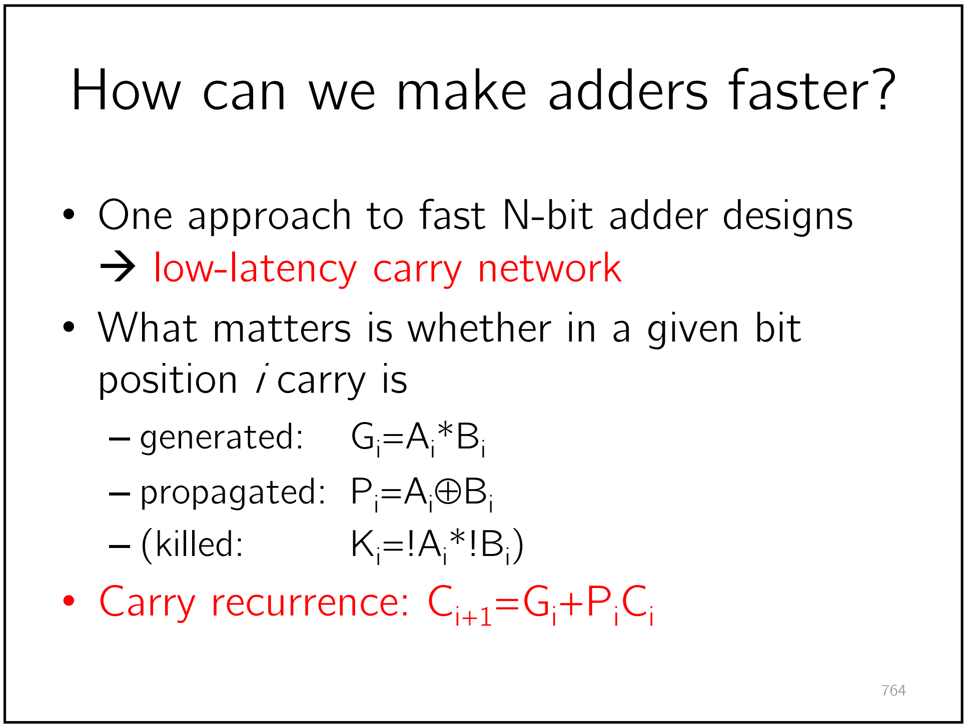
Manchester carry chain
T=O(N), A=O(N)

carry select adders
T=O(N) or T=O(sqrt(N))
A=O(N)

carry skip adders
T=O(sqrt(N))
A=O(N)

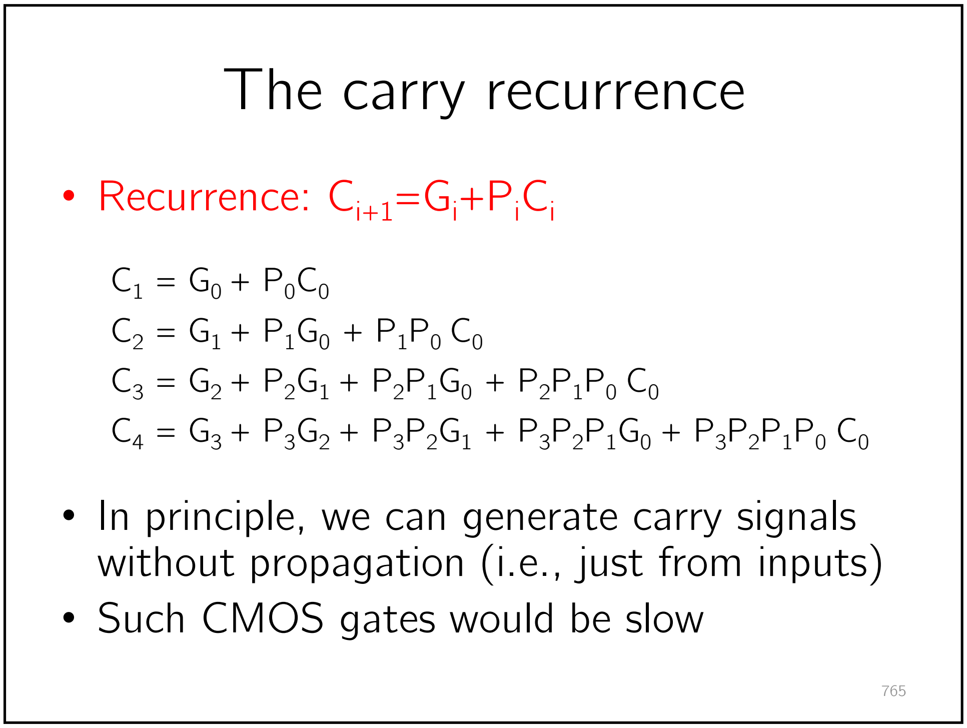
parallel prefix
T=O(log(N))
A=O(N*log(N))

- As always, there is an area/time trade-off



How can we make adders faster?

- One approach to fast N-bit adder designs → low-latency carry network
- What matters is whether in a given bit position *i* carry is
  - generated: $G_i=A_i*B_i$
  - propagated: $P_i=A_i \oplus B_i$
  - (killed: $K_i=!A_i*!B_i$)
- Carry recurrence: $C_{i+1}=G_i+P_iC_i$

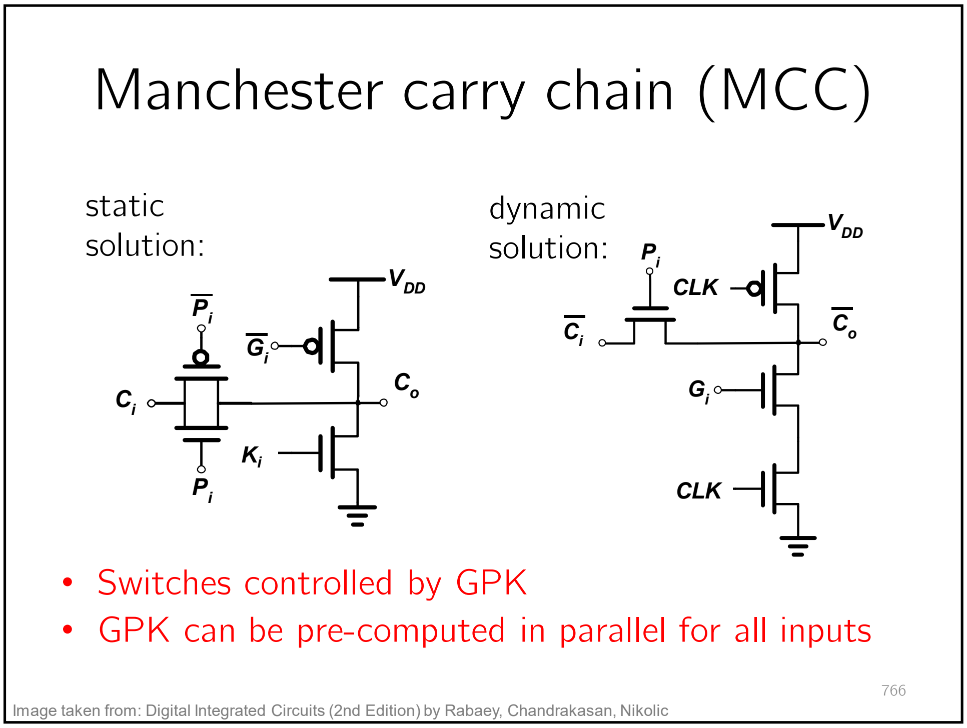


The carry recurrence

- Recurrence: $C_{i+1}=G_i+P_iC_i$

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0 C_0$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0 C_0$$
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0 C_0$$

- In principle, we can generate carry signals without propagation (i.e., just from inputs)
- Such CMOS gates would be slow



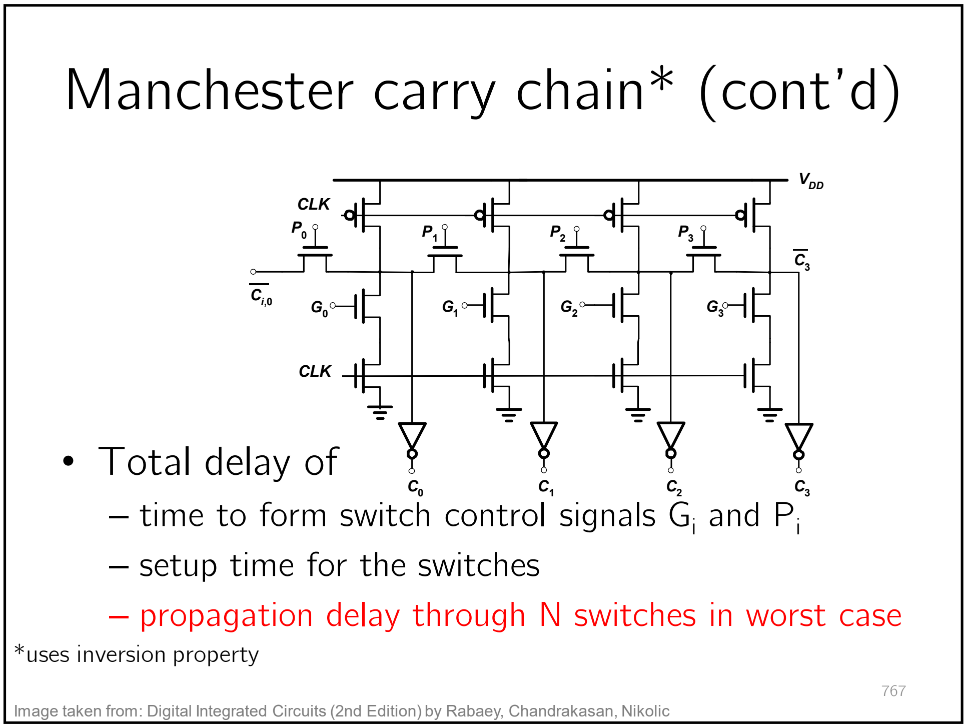
Manchester carry chain (MCC)

static solution:

dynamic solution:

- Switches controlled by GPK
- GPK can be pre-computed in parallel for all inputs

Image taken from: Digital Integrated Circuits (2nd Edition) by Rabaey, Chandrakasan, Nikolic



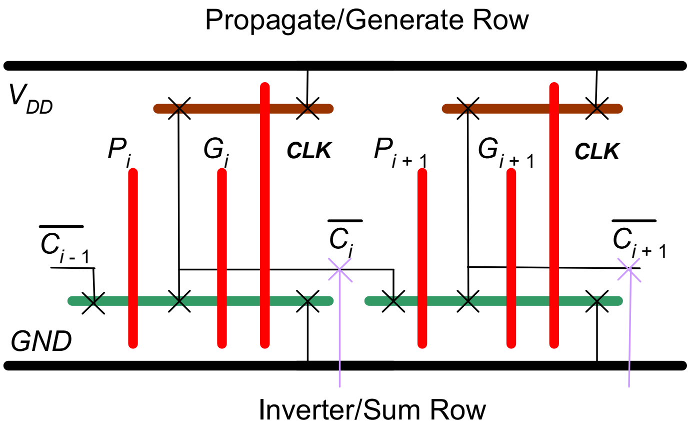
Manchester carry chain* (cont'd)

- Total delay of
  - time to form switch control signals G_i and P_i
  - setup time for the switches
  - propagation delay through N switches in worst case

*uses inversion property

Image taken from: Digital Integrated Circuits (2nd Edition) by Rabaey, Chandrakasan, Nikolic

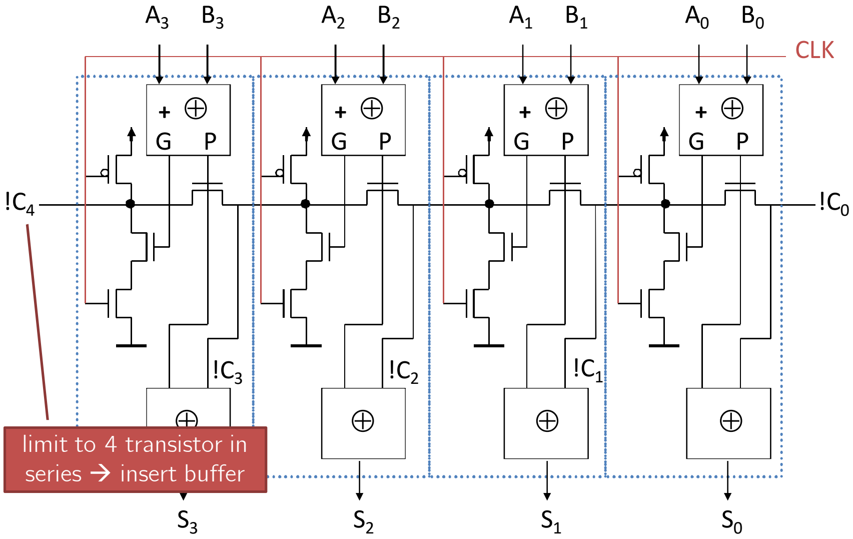
(MCC stick diagram)



768

Image taken from: Digital Integrated Circuits (2nd Edition) by Rabaey, Chandrakasan, Nikolic

Dynamic 4-bit MCC adder



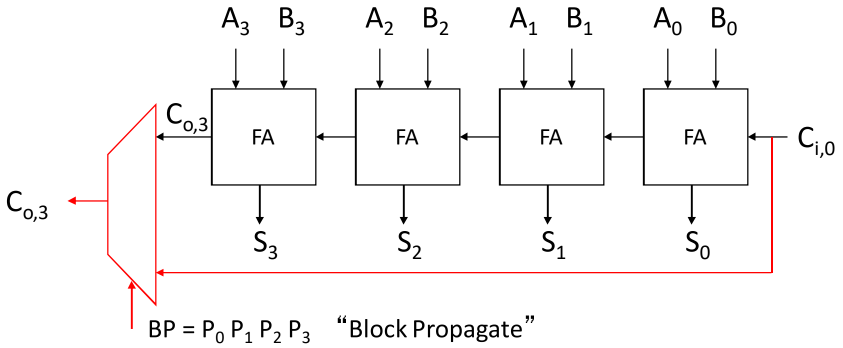
limit to 4 transistor in series → insert buffer

769

Another (often confusing) trick

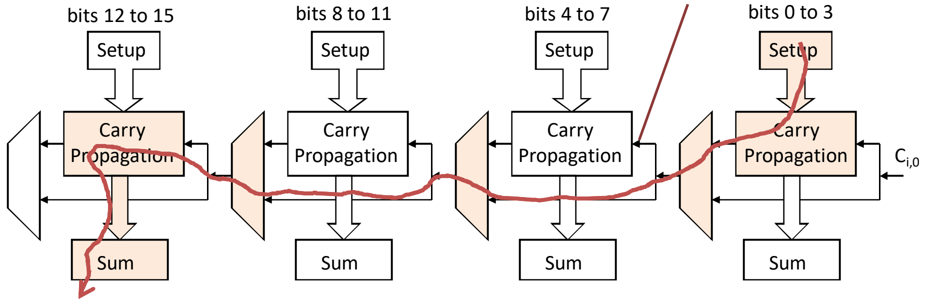
Carry-skip adder

Carry-skip (or bypass) adder



- If $BP=P_0P_1P_2P_3=1$ then $C_{O,3}=C_{i,0}$, otherwise block itself generates (or kills) carry internally

N-bit adder with groups of B=4 bit



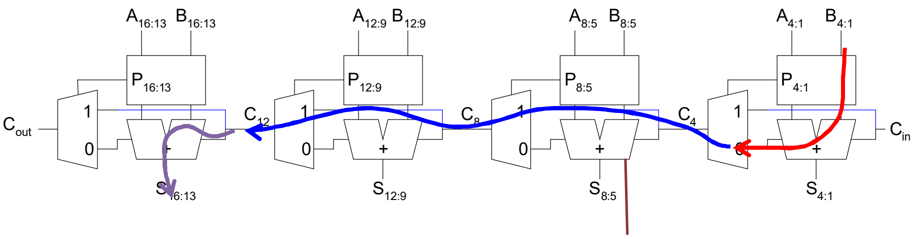
no direct path to carry out

- Worst case: ripples 0-3, skips middle two groups, ripples through last group 12-15

772

Image adapted from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris

Critical path revisited



there is no direct C_{in}-to-C_{out} path

- $t_{add}=t_{setup}+Bt_{carry}+(N/B-1)t_{skip}+Bt_{carry}+t_{sum}$

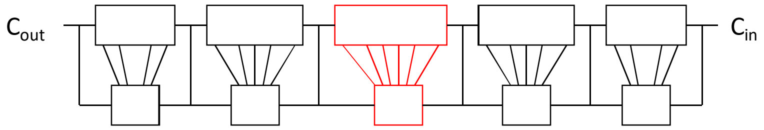
773

Optimal block size and time

- Assume one stage of ripple (t_{carry}) has same delay as skip stage (t_{skip}) and both =1
- $T_{add}=t_{setup}+Bt_{carry}+(N/B\text{-}1)t_{skip}+Bt_{carry}+t_{sum}$
  $=1+2^*B+N/B$
- Optimal block size B_{opt} is: $dT_{add}/dB=0$
- Bits per stage: $B_{opt}=\text{sqrt}(N/2)$
- Optimal delay: $T_{add,opt}=1+3^*\text{sqrt}(2^*N)$

Improving the carry skip adder

- Variable block sizes:
  - Using shorter groups at the beginning and end, and longer groups in the middle
  - Reduces critical path (a little)

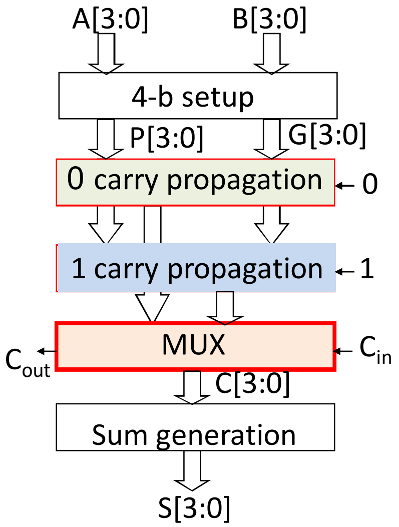


Yet another trick

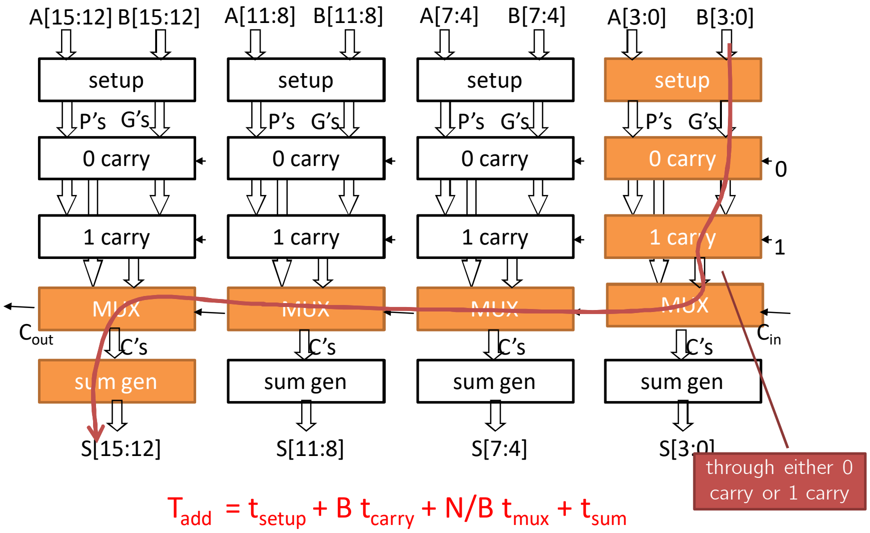
Carry-select adder (CSA)

Carry select adder

- Pre-compute carry out for each block for C_{in}=0 and C_{in}=1 (can be done in parallel)
- Select correct outputs as soon as C_{in} is ready



Critical path of carry select adder

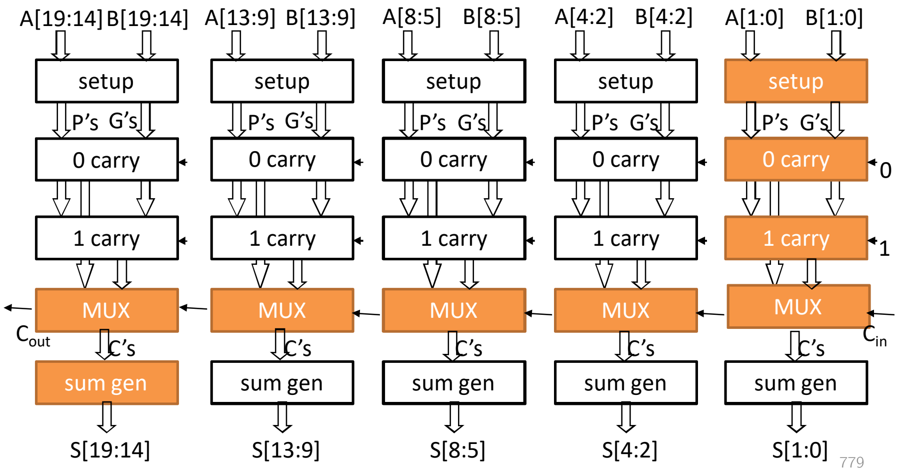


$$T_{add} = t_{setup} + B\ t_{carry} + N/B\ t_{mux} + t_{sum}$$

778

Square root carry select adder

- Use increasing group sizes (increment 1)

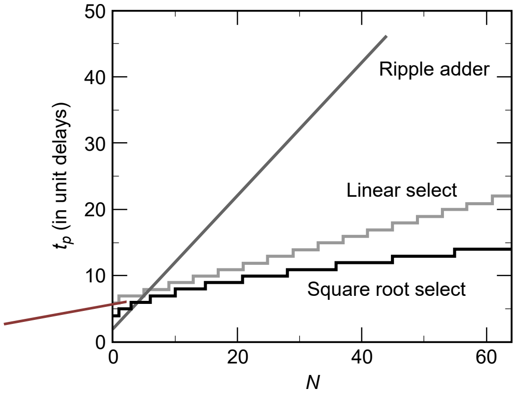


779

Delay of square root CSA

- $T_{add} = t_{setup} + 2*t_{carry} + sqrt(N)*t_{mux} + t_{sum}$

- Significantly faster than ripple adder for large N

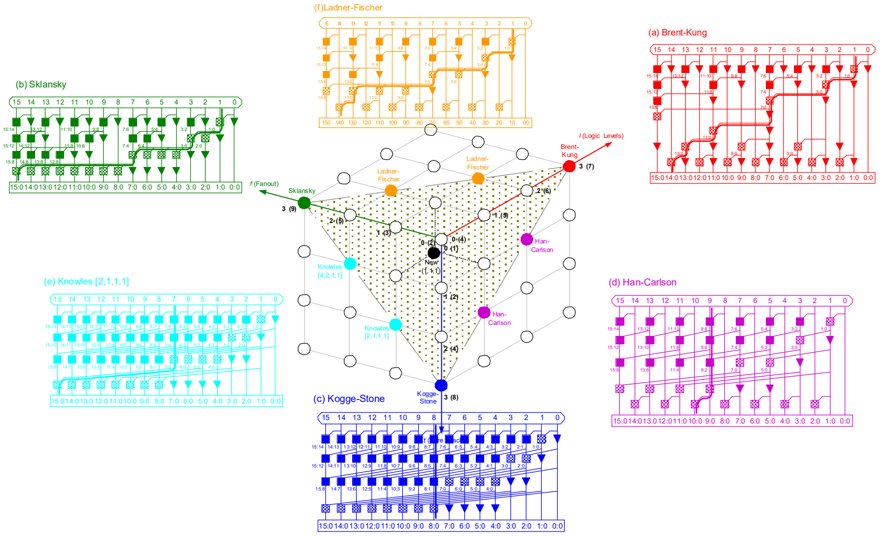


for a very small number of bits, SQRT CSA might not be better

Image taken from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris

780

Adders = science (or used to be)

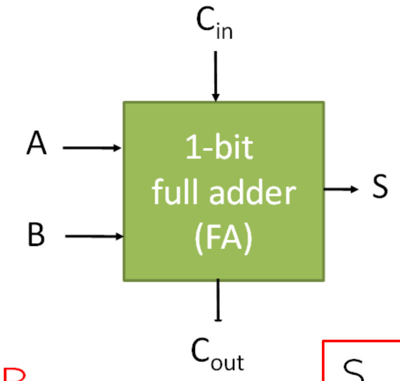


781

We can do better

Parallel prefix/tree adders

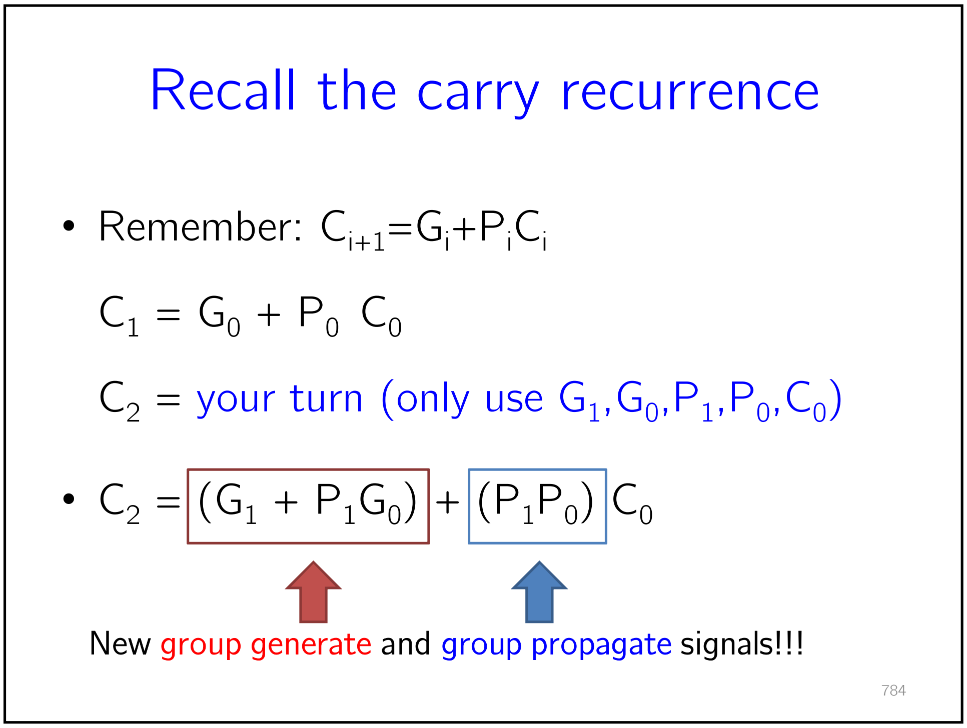
Full adder (FA)



$G = A*B$

$P = A\oplus B$

$$\boxed{\begin{aligned} S &= P\oplus C_{in} \\ C_{out} &= G+P*C_{in} \end{aligned}}$$



Recall the carry recurrence

- Remember: $C_{i+1}=G_i+P_iC_i$

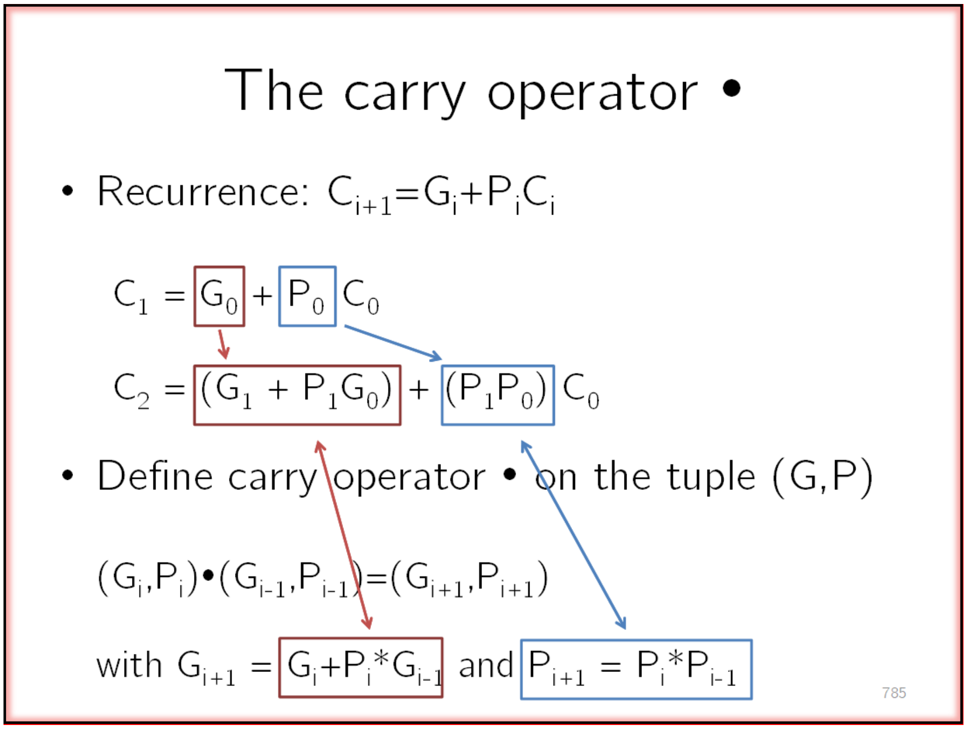
$C_1 = G_0 + P_0\ C_0$

C_2 = your turn (only use G_1,G_0,P_1,P_0,C_0)

- $C_2 = \boxed{(G_1 + P_1G_0)} + \boxed{(P_1P_0)}\ C_0$

New group generate and group propagate signals!!!

784



The carry operator •

- Recurrence: $C_{i+1}=G_i+P_iC_i$

$C_1 = \boxed{G_0} + \boxed{P_0}\ C_0$

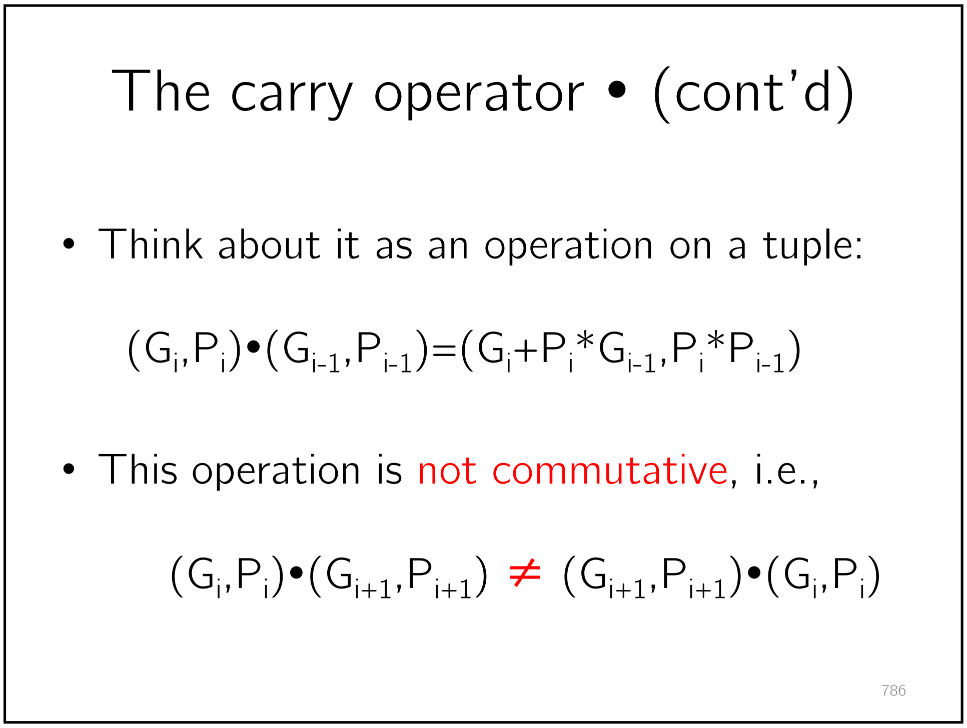
$C_2 = \boxed{(G_1 + P_1G_0)} + \boxed{(P_1P_0)}\ C_0$

- Define carry operator • on the tuple (G,P)

$(G_i,P_i)\bullet(G_{i-1},P_{i-1})=(G_{i+1},P_{i+1})$

with $G_{i+1} = \boxed{G_i+P_i*G_{i-1}}$ and $\boxed{P_{i+1} = P_i*P_{i-1}}$

785



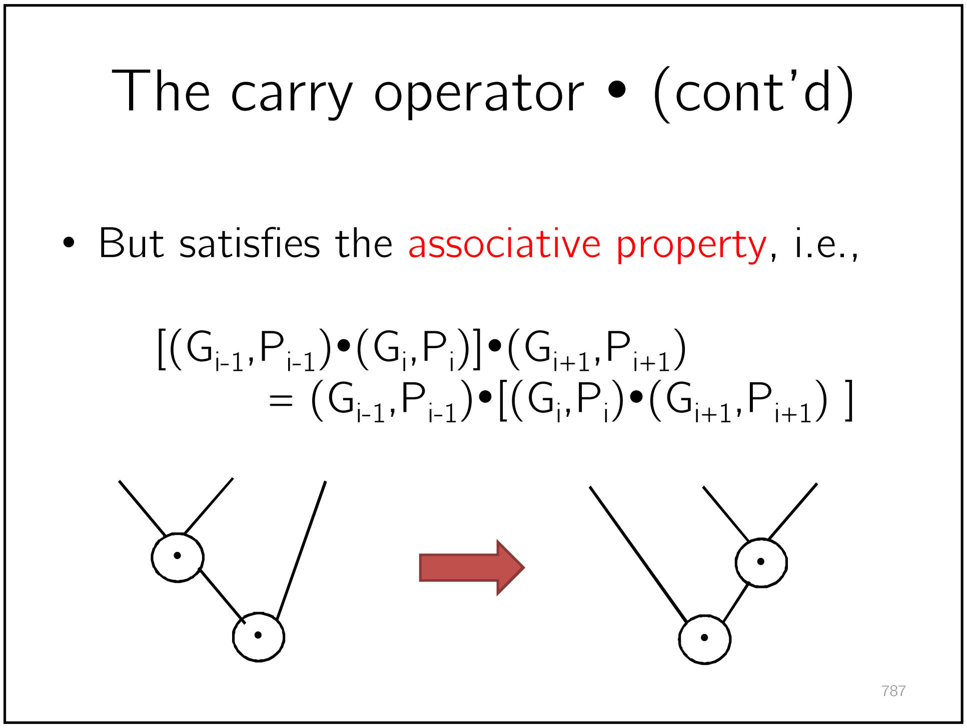
The carry operator • (cont'd)

- Think about it as an operation on a tuple:

$$(G_i,P_i)\bullet(G_{i-1},P_{i-1})=(G_i+P_i*G_{i-1},P_i*P_{i-1})$$

- This operation is not commutative, i.e.,

$$(G_i,P_i)\bullet(G_{i+1},P_{i+1}) \neq (G_{i+1},P_{i+1})\bullet(G_i,P_i)$$



The carry operator • (cont'd)

- But satisfies the associative property, i.e.,

$$[(G_{i-1},P_{i-1})\bullet(G_i,P_i)]\bullet(G_{i+1},P_{i+1})$$
$$= (G_{i-1},P_{i-1})\bullet[(G_i,P_i)\bullet(G_{i+1},P_{i+1})\ ]$$

General idea of prefix adders

- Given the G_i and P_i for each bit *i* computing all carries is equal to finding all prefixes

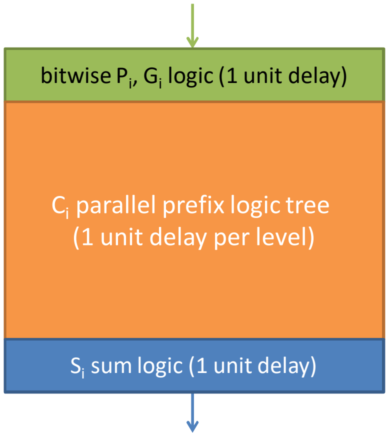
$$(C_{out},0)=(G_{N-1},P_{N-1})\bullet\ldots\bullet(G_1,P_1)\bullet(G_0,P_0)\bullet(C_{in},0)$$

- Important: since • is associative we can
  - Group the terms in any order
  - Reuse intermediate group and propagate signals

788

General adder topology

$$(C_{out},0)=(G_{N-1},P_{N-1})\bullet\ldots\bullet(G_1,P_1)\bullet(G_0,P_0)\bullet(C_{in},0)$$


bitwise P_i, G_i logic (1 unit delay)
C_i parallel prefix logic tree (1 unit delay per level)
S_i sum logic (1 unit delay)

- Metrics to consider
  - Number of • cells
  - Tree depth → delay
  - Tree cell area
  - Fan in and fan out
  - Max-wire length
  - Wiring congestion
  - Delay path variation

789

Grouping example: 4-bit adder

$$(C_{out},0)=[\underbrace{(G_3,P_3)\bullet(G_2,P_2)\bullet(G_1,P_1)\bullet(G_0,P_0)}]$$

$$(G_{3:0},P_{3:0})=[\underbrace{(G_3,P_3)\bullet(G_2,P_2)}]\bullet[\underbrace{(G_1,P_1)\bullet(G_0,P_0)}]$$

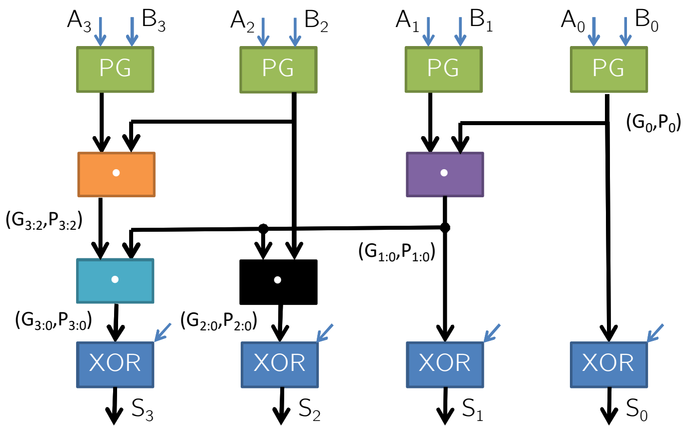
$$(G_{3:0},P_{3:0})=(G_{3:2},P_{3:2})\bullet(G_{1:0},P_{1:0})$$

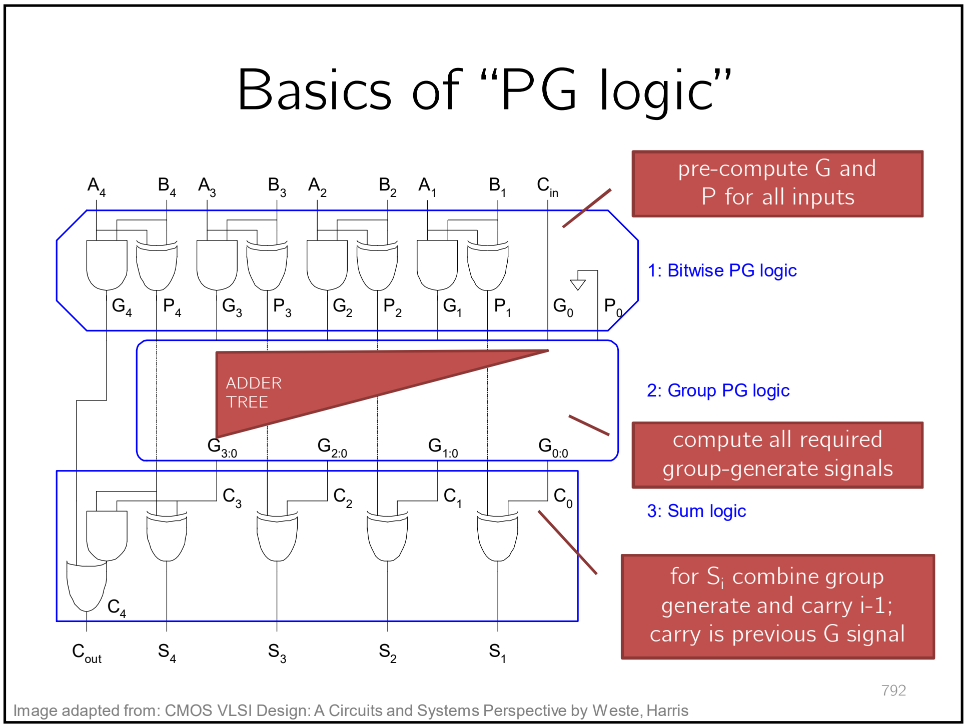
can be re-used!

$$(C_{out},0) = [(G_3,P_3)\bullet(G_2,P_2)\bullet(G_1,P_1)\bullet(G_0,P_0)]$$

$$(G_{3:0},P_{3:0}) = [(G_3,P_3)\bullet(G_2,P_2)]\bullet[(G_1,P_1)\bullet(G_0,P_0)]$$

$$(G_{3:0},P_{3:0}) = (G_{3:2},P_{3:2})\bullet(G_{1:0},P_{1:0})$$





Basics of “PG logic”

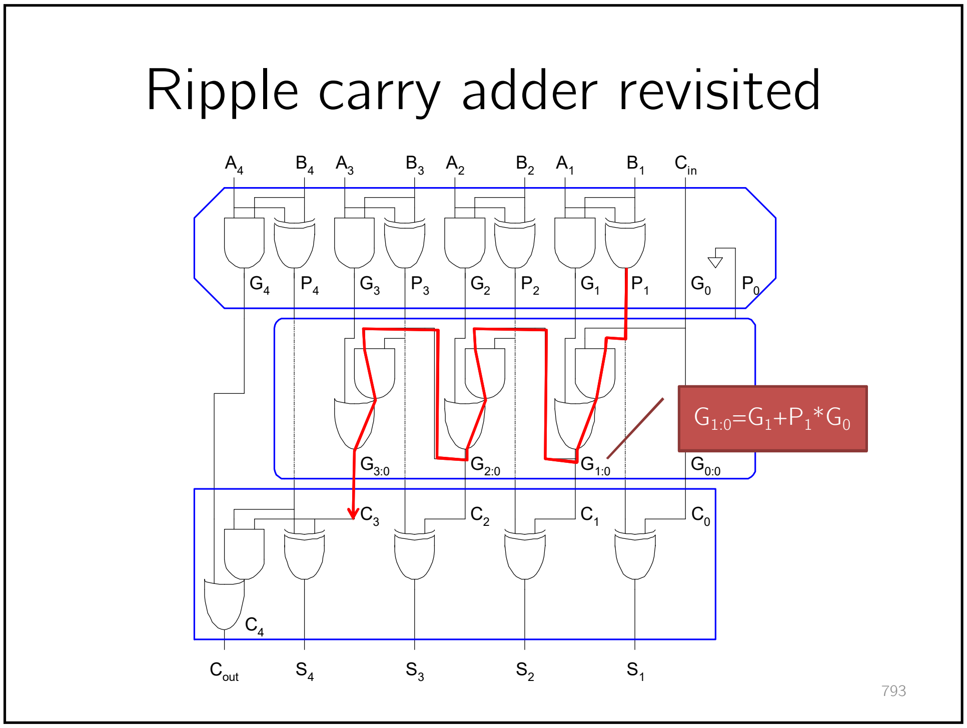
pre-compute G and P for all inputs

compute all required group-generate signals

for S_i combine group generate and carry i-1; carry is previous G signal

792

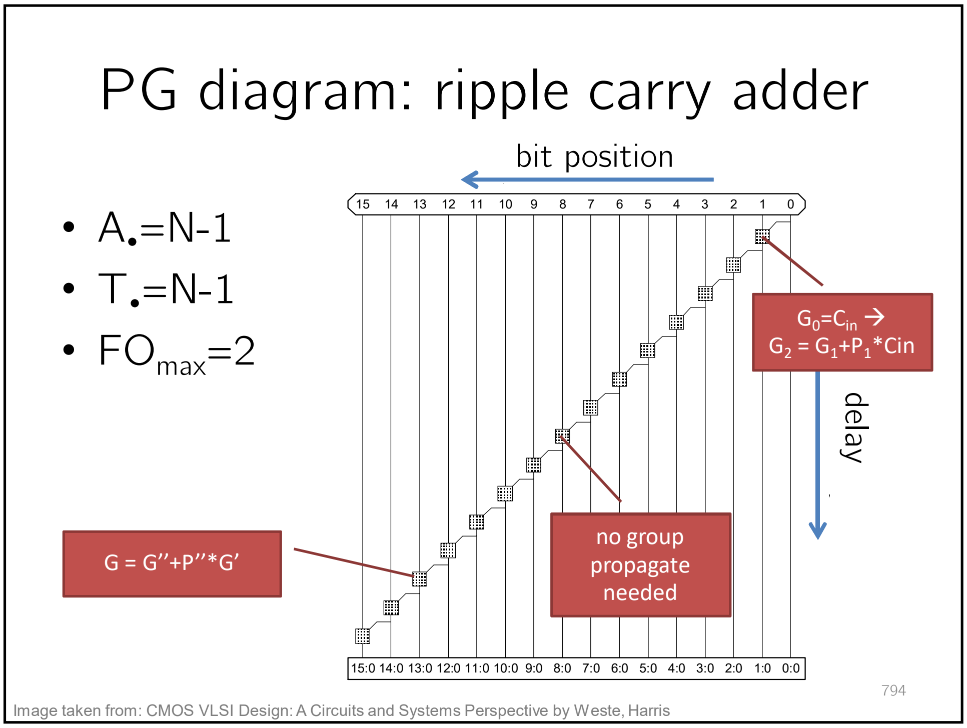
Image adapted from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris



Ripple carry adder revisited

$G_{1:0}=G_1+P_1*G_0$

793



PG diagram: ripple carry adder

- $A_\bullet$=N-1
- $T_\bullet$=N-1
- FO_{max}=2

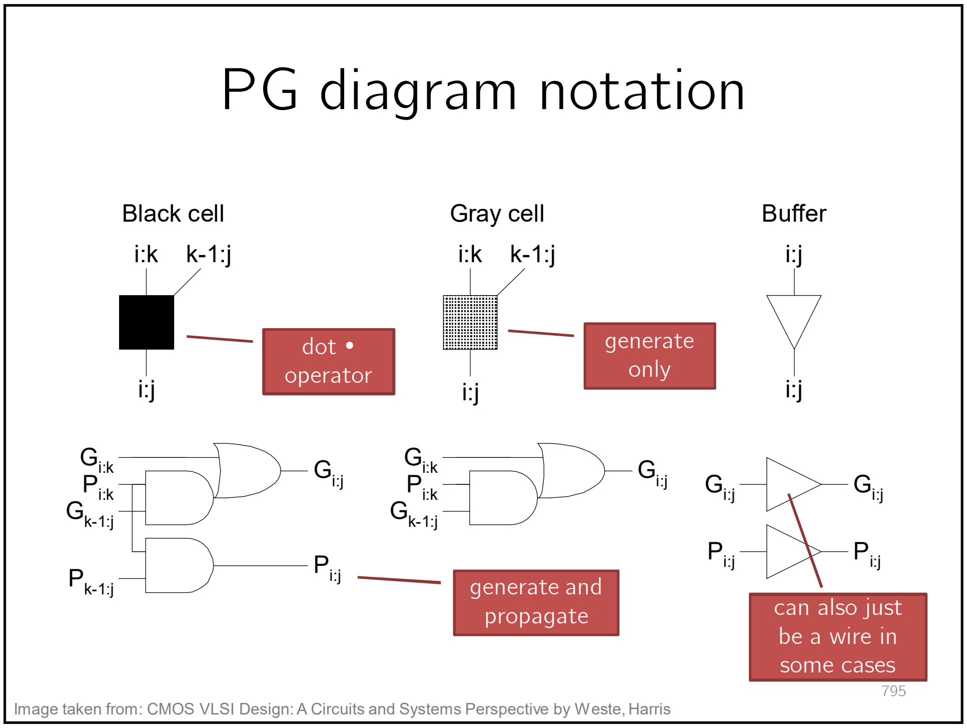
$G_0=C_{in}$ → $G_2 = G_1+P_1*Cin$

no group propagate needed

G = G''+P''*G'

Image taken from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris

794



PG diagram notation

dot •
operator

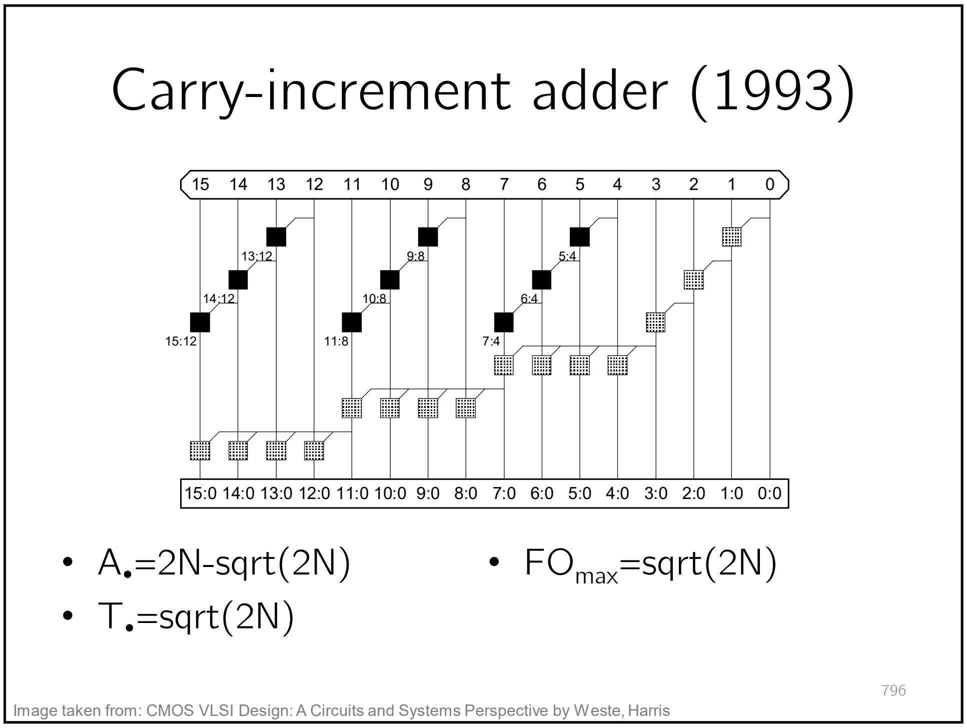
generate
only

generate and
propagate

can also just
be a wire in
some cases

Image taken from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris

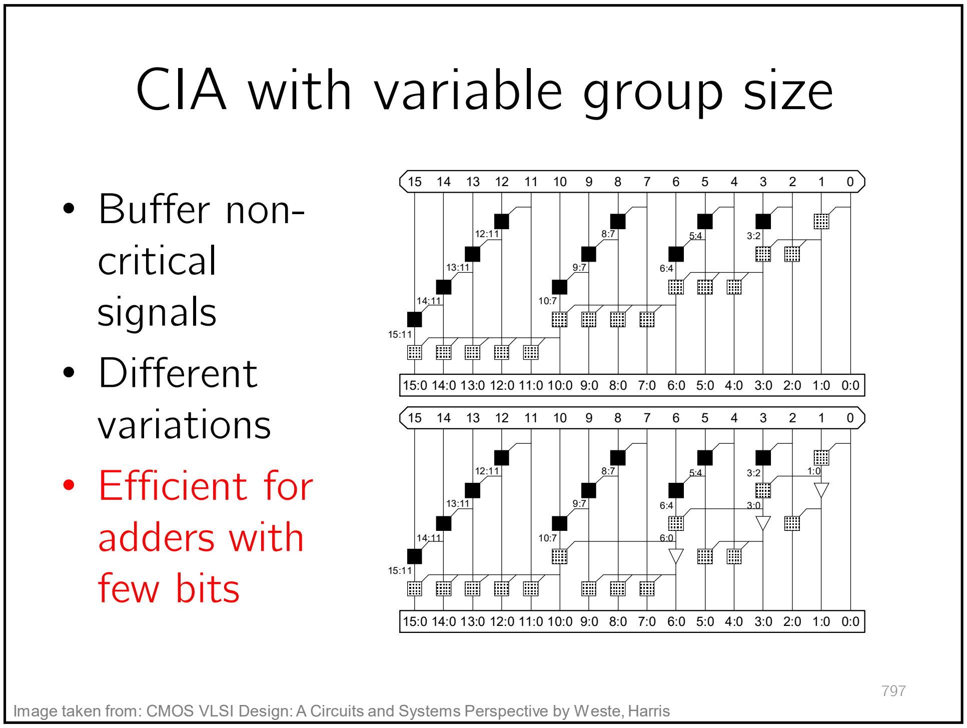
795



Carry-increment adder (1993)

- $A_\bullet$=2N-sqrt(2N)
- $T_\bullet$=sqrt(2N)
- FO_{max}=sqrt(2N)

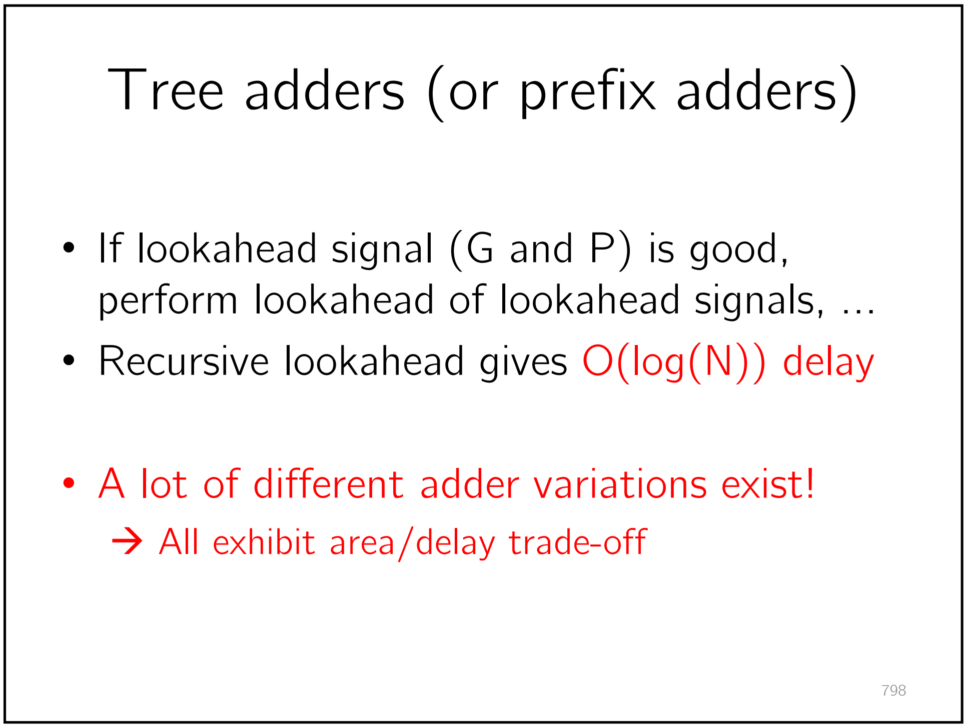
Image taken from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris



CIA with variable group size

- Buffer non-critical signals
- Different variations
- Efficient for adders with few bits

Image taken from: CMOS VLSI Design: A Circuits and Systems Perspective by Weste, Harris



Tree adders (or prefix adders)

- If lookahead signal (G and P) is good, perform lookahead of lookahead signals, …
- Recursive lookahead gives O(log(N)) delay

- A lot of different adder variations exist!
  → All exhibit area/delay trade-off