




EECS 270 – Introduction to Logic Design Winter 2013

17. Fast Adders

Prof. Valeria Bertacco & Prof. Kang Shin
EECS Department
University of Michigan, Ann Arbor

Copyright © 2009 Frank Vahid

Instructors of courses requiring Vahid's Digital Design textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites.. PowerPoint source (or pdf with animations) may not be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.ddvahid.com> for information.



Last lecture and Today

- 
- RTL design

TODAY

- Carry-select adders
- Carry-lookahead adders



Review: Ripple-carry adder

- Ripple-carry adder (from Ch 4)

- Similar to adding by hand, column by column

- **Con: Slow**

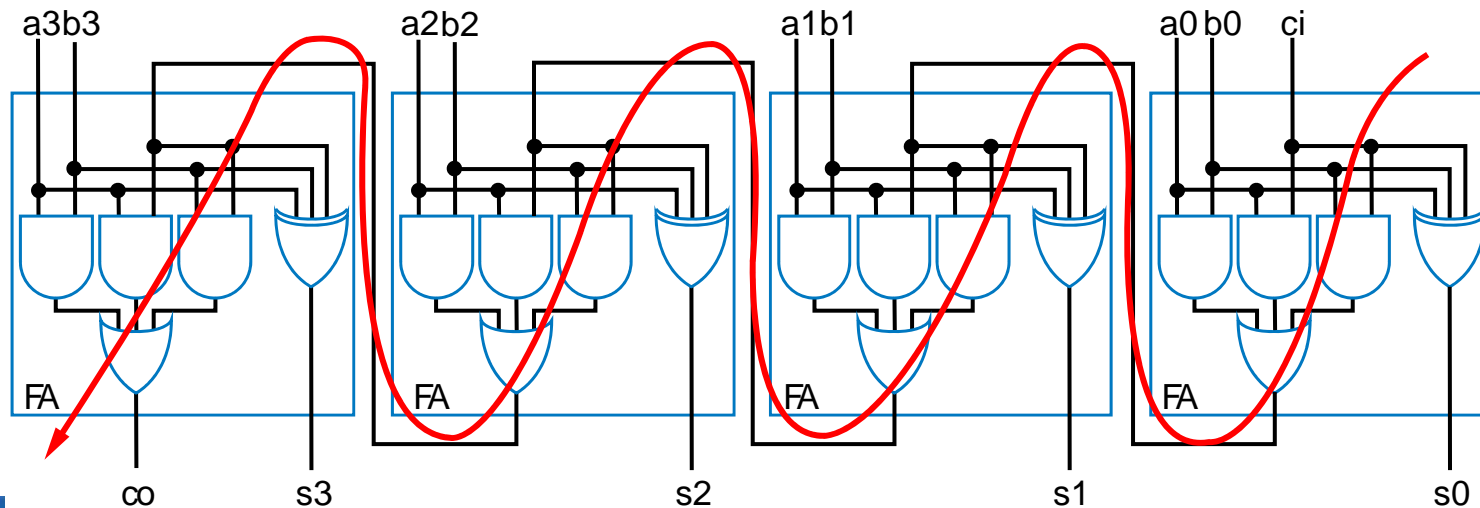
- Output is not correct until the carry have rippled to the left
- 4-bit carry-ripple adder has $4 \times 2 = 8$ gate delays

- **Pro: Small**

- 4-bit carry-ripple adder has just $4 \times 5 = 20$ gates

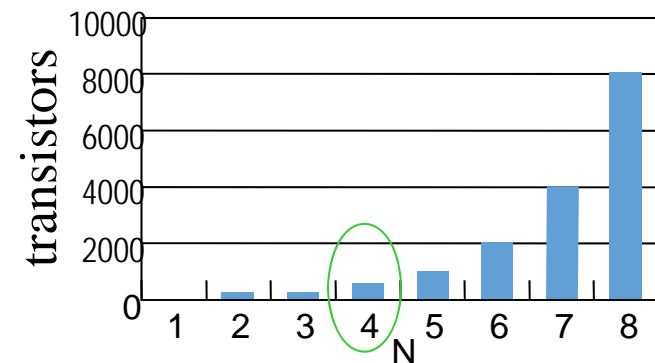
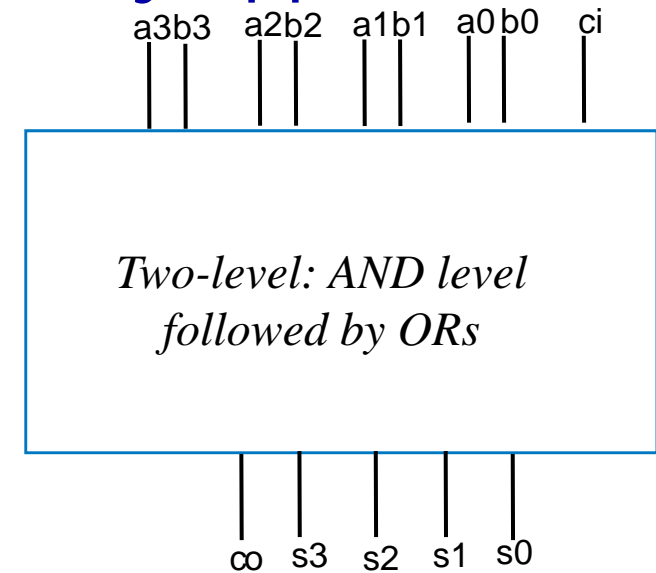
a3 b3	a2 b2	a1 b1	a0 b0	cin
4-bit adder				
cout	s3	s2	s1	s0

carry:	c3	c2	c1	cin	
B:	b3	b2	b1	b0	
A:	+	a3	a2	a1	a0
	<hr/>				
cout	s3	s2	s1	s0	



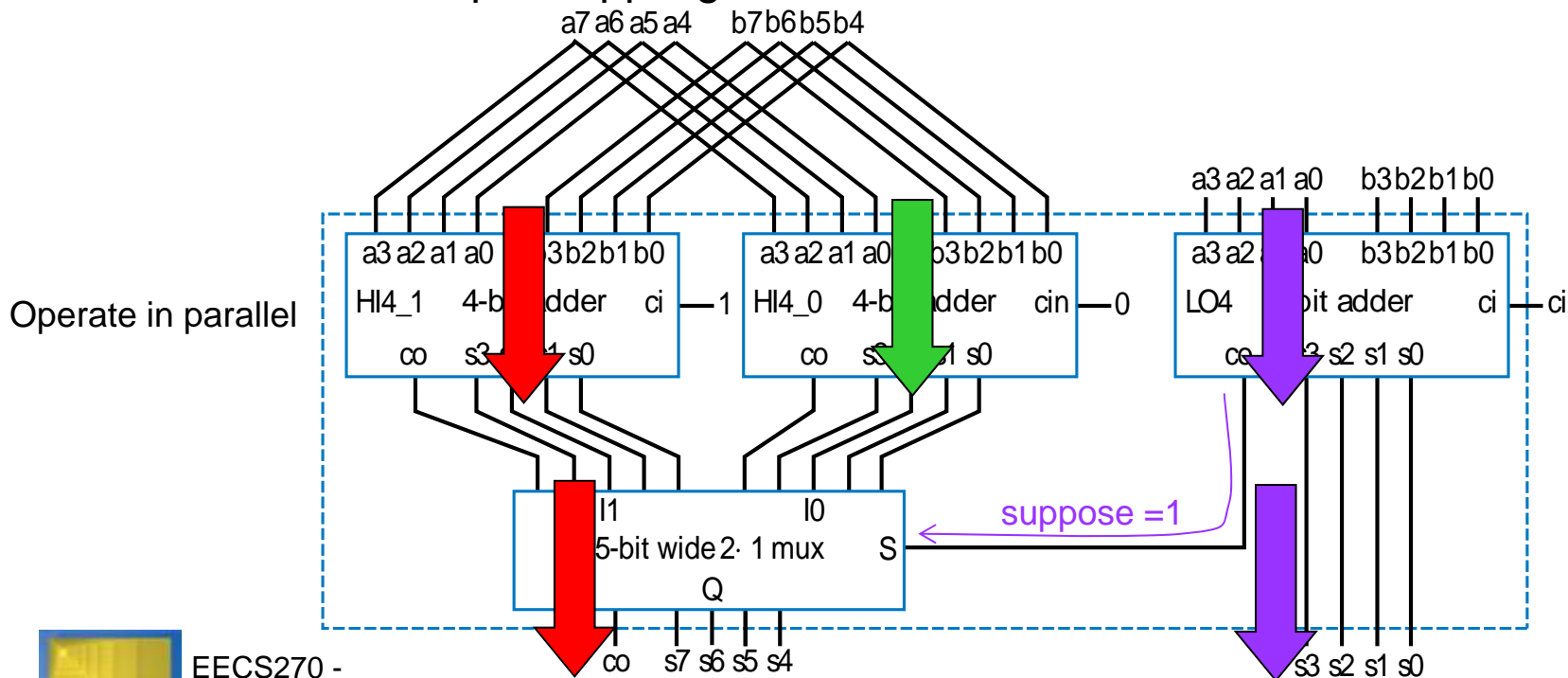
A faster adder: two-level carry-ripple

- Faster adder – Use two-level combinational logic design process
 - Gets REALLY BIG REALLY FAST
 - Pro: Fast
 - 2 gate delays
 - Con: Large
 - Truth table would have $2^{(4+4)} = 256$ rows
 - Plot shows 4-bit adder would use about 500 gates
- *Is there a compromise design?*
 - Between 2 and 8 gate delays
 - Between 20 and 500 gates



Carry-select adders (CSA)

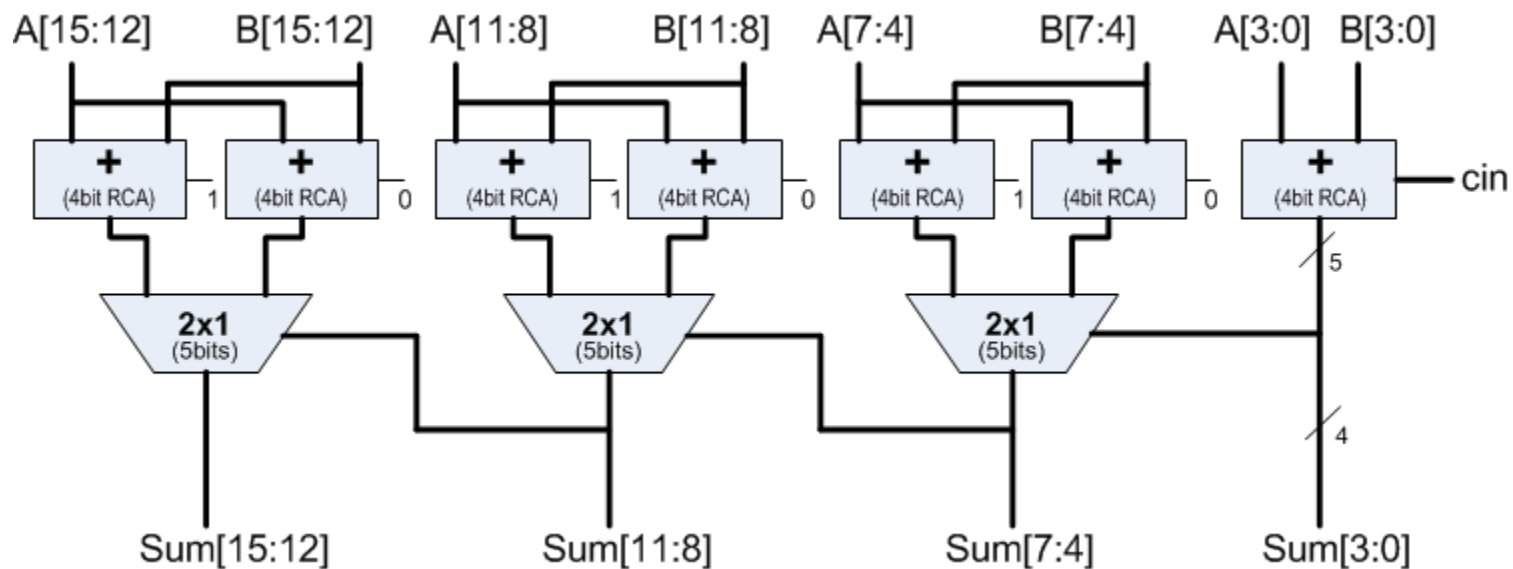
- A way to compose adders:
 - Divide the bits into blocks
 - High-order stage -- Compute result for carry in of **1** and of **0**
 - Select based on **carry-out** of low-order stage
 - Faster than pure rippling



Cascaded architecture for carry-select adders

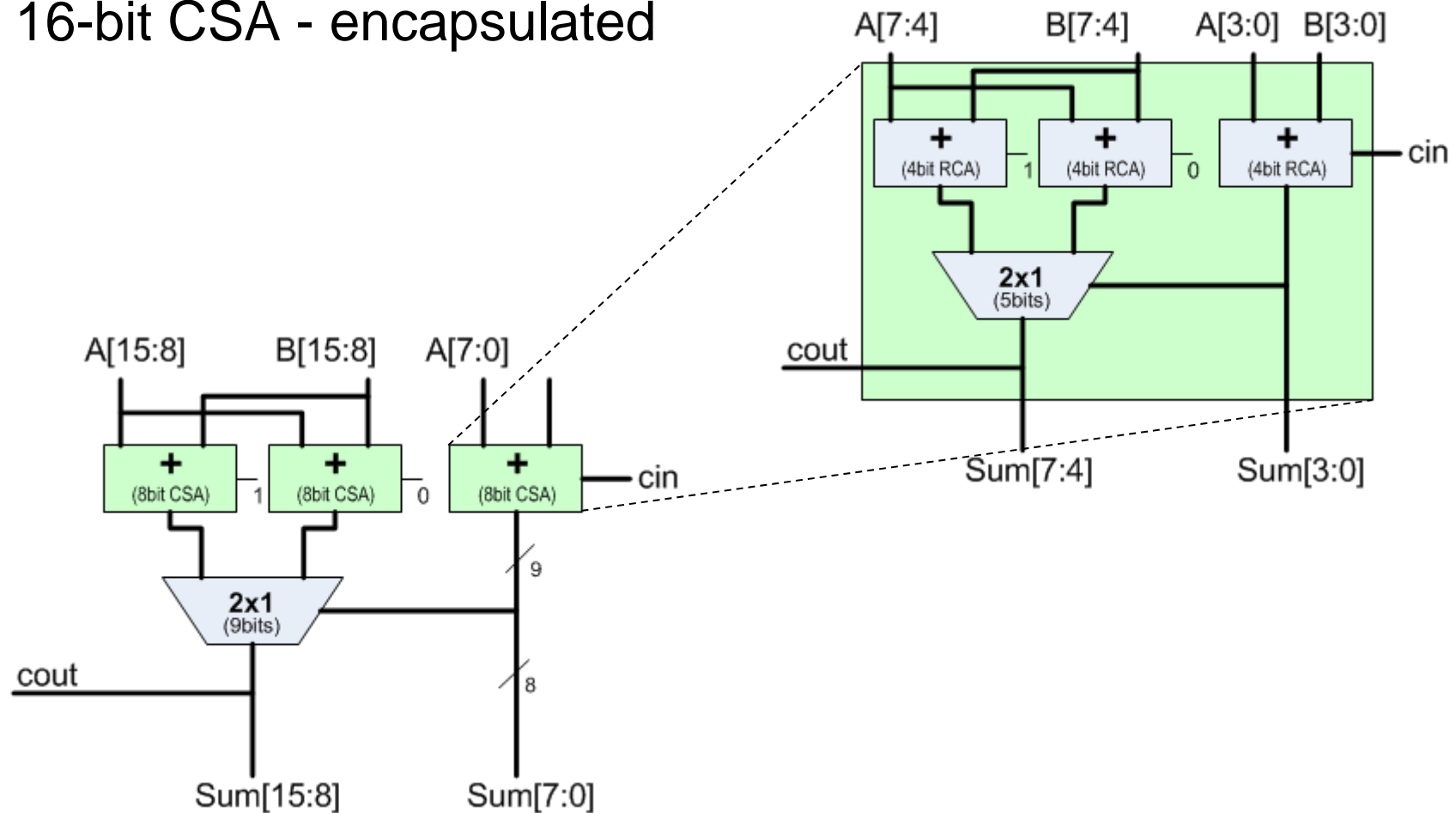
- Uses 4bits RCAs adders as basic building blocks

16-bit carry-select adder - cascaded



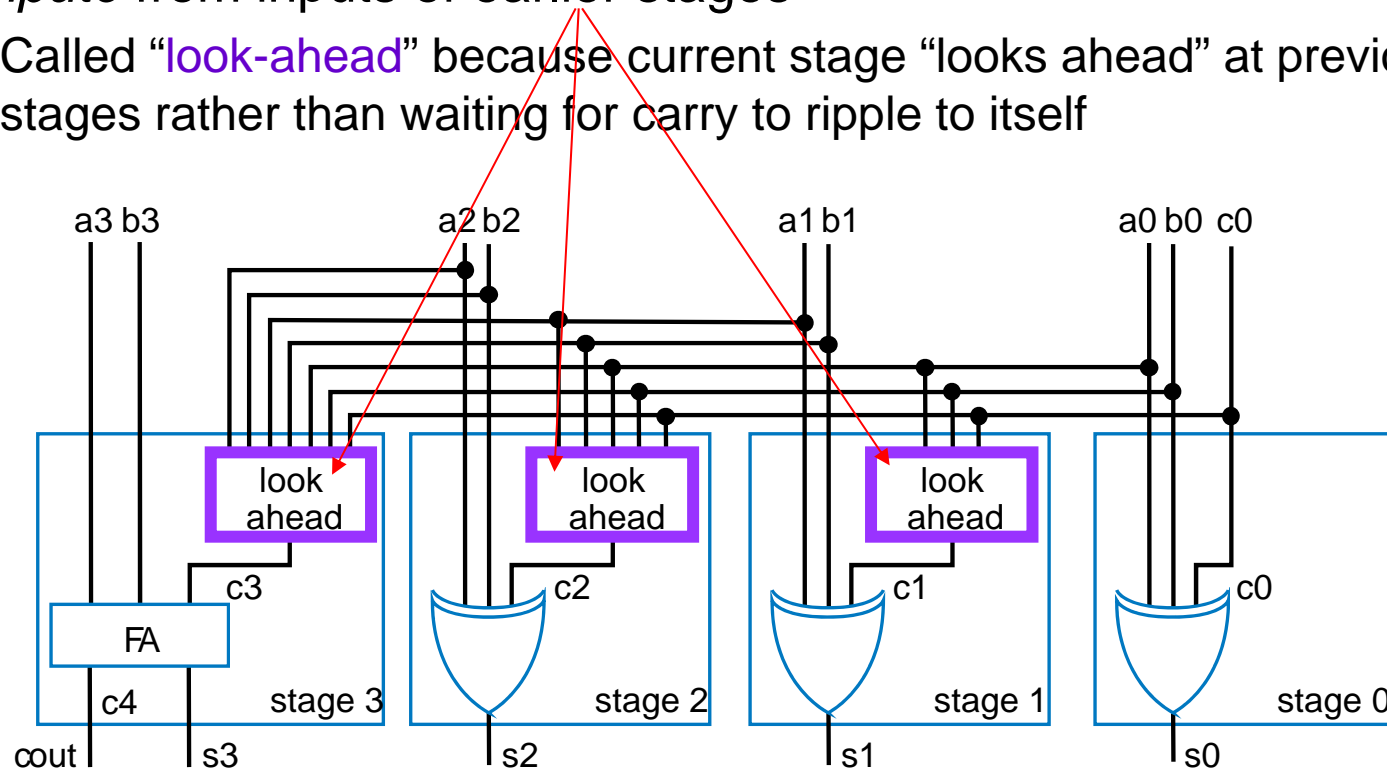
Encapsulated architecture for CSAs

- 16-bit CSA - encapsulated

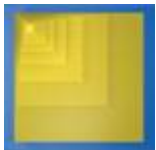


First attempt at “look-ahead” adders

- Idea: modify ripple-carry adder
 - For a stage's carry-in, don't wait for carry to ripple, but rather *directly compute* from inputs of earlier stages
 - Called “look-ahead” because current stage “looks ahead” at previous stages rather than waiting for carry to ripple to itself

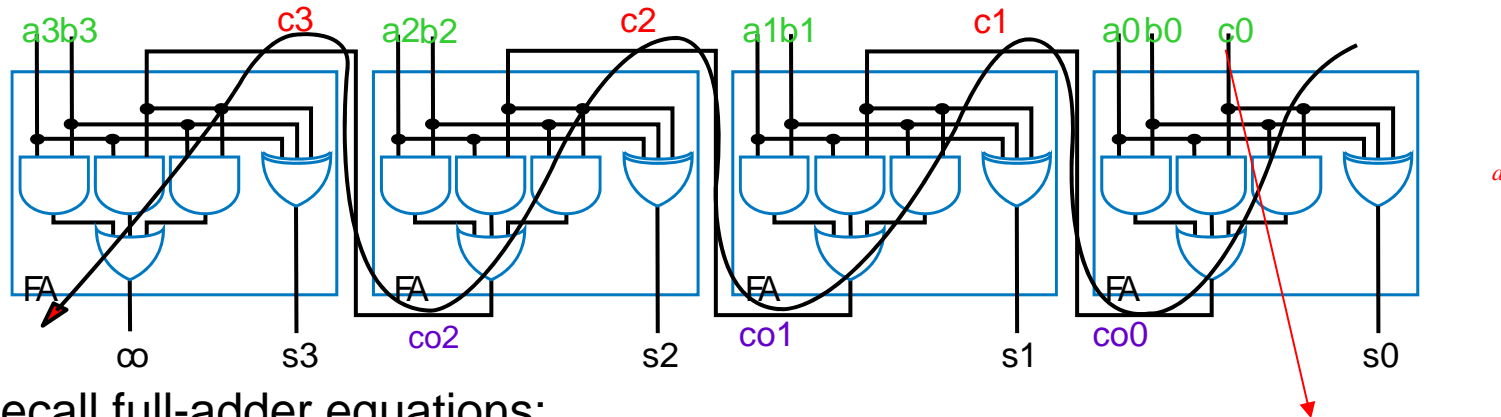


Notice – no rippling of carry

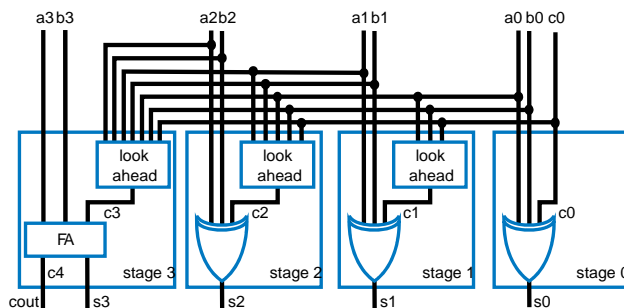


Designing the look-ahead block

- Each stage's **carry-in** bit should be a function of **external inputs** only (a's, b's, or c0)



- Recall full-adder equations:
 $s = a \text{ xor } b$ and $c = bc + ac + ab$



Stage 0: Carry-in is already an external input: **c0**

Stage 1: **c1** = c_0 ← $c_0 = b_0c_0 + a_0c_0 + a_0b_0$

$$c_1 = b_0c_0 + a_0c_0 + a_0b_0$$

Stage 2: **c2** = c_1

$$c_1 = b_1c_1 + a_1c_1 + a_1b_1$$

$$c_2 = b_1c_1 + a_1c_1 + a_1b_1$$

$$c_2 = b_1(b_0c_0 + a_0c_0 + a_0b_0) + a_1(b_0c_0 + a_0c_0 + a_0b_0) + a_1b_1$$

$$c_2 = b_1b_0c_0 + b_1a_0c_0 + b_1a_0b_0 + a_1b_0c_0 + a_1a_0c_0 + a_1a_0b_0 + a_1b_1$$

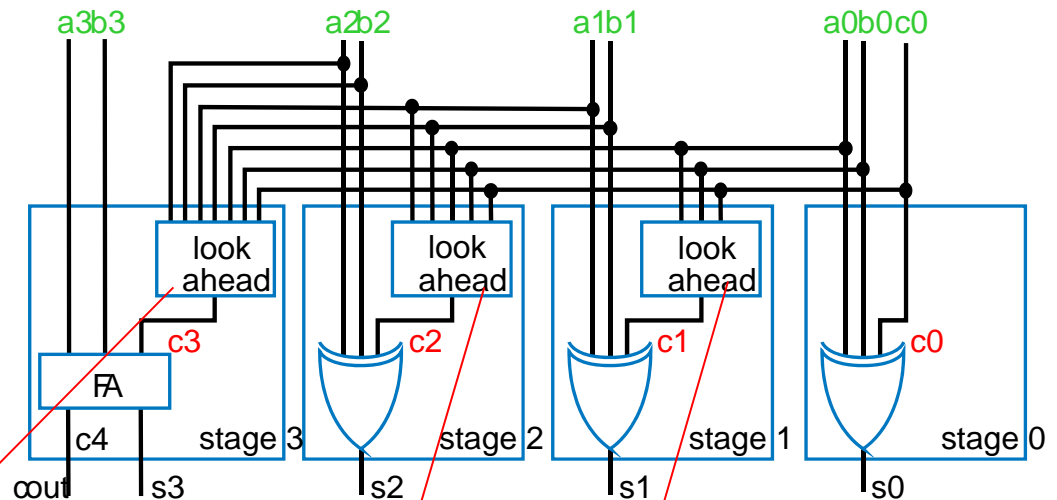


First attempt at "look-ahead" adders

- **Carry** look-ahead logic is function of **external inputs**
 - No waiting for ripple

- **Cons**

- Equations get too big (= look-ahead blocks get big)
- Some gates have many inputs



$$c1 = b0c0 + a0c0 + a0b0$$

$$c2 = b1b0c0 + b1a0c0 + b1a0b0 + a1b0c0 + a1a0c0 + a1a0b0 + a1b1$$

$$c3 = b2b1b0c0 + b2b1a0c0 + b2b1a0b0 + b2a1b0c0 + b2a1a0c0 + b2a1a0b0 + b2a1b1 + a2b1b0c0 + a2b1a0c0 + a2b1a0b0 + a2a1b0c0 + a2a1a0c0 + a2a1a0b0 + a2a1b1 + a2b2$$



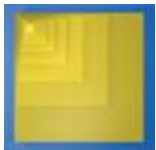
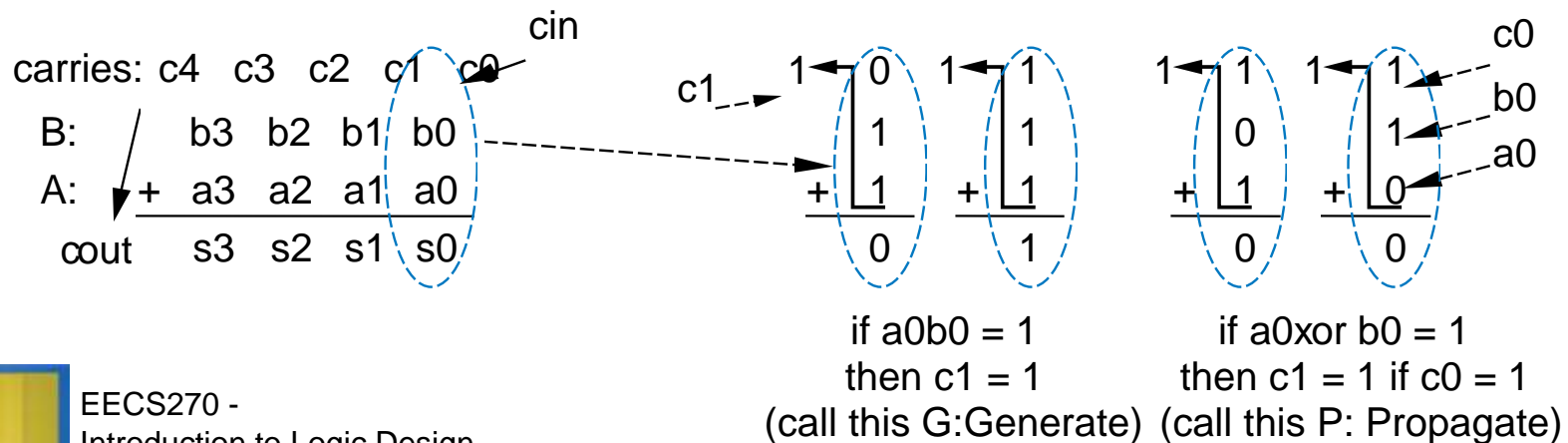
Better (and common) form of look-ahead

- Have each stage compute two terms:

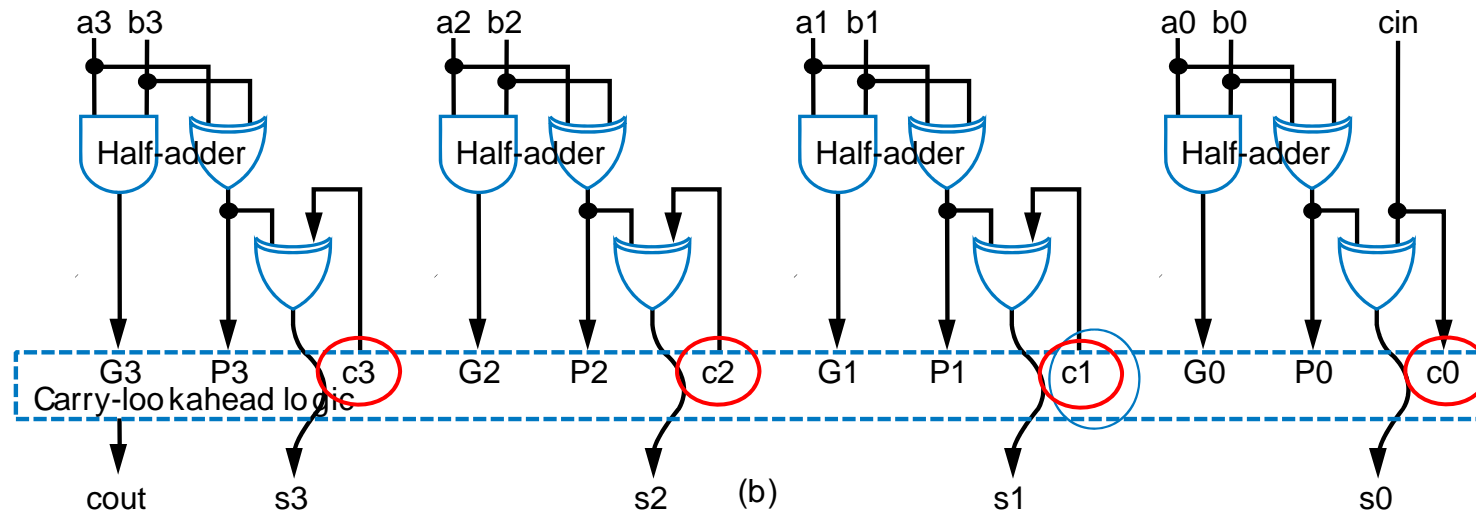
-Propagate: $P = a \text{ xor } b$

- Generate: $G = ab$

- Compute look-ahead from P and G terms, *not from external inputs*
 - Why P & G ? Because the logic comes out much simpler
 - Very clever finding; not particularly obvious though
 - Why those names?
 - G : If a and b are 1, carry-out will be 1 – “generate” a carry-out of 1 in this case
 - P : If only one of a or b is 1, then carry-out will equal the carry-in – propagate the carry-in to the carry-out in this case



Common form of look-ahead



- With P & G , the carry look-ahead equations are much simpler

– Equations before substituting

- $c_1 = G_0 + P_0c_0$
- $c_2 = G_1 + P_1c_1$
- $c_3 = G_2 + P_2c_2$
- $\text{cout} = G_3 + P_3c_3$

After substituting:

$$c_1 = G_0 + P_0c_0$$

$$c_2 = G_1 + P_1c_1 = G_1 + P_1(G_0 + P_0c_0)$$

$$c_2 = G_1 + P_1G_0 + P_1P_0c_0$$

$$c_3 = G_2 + P_2c_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0c_0)$$

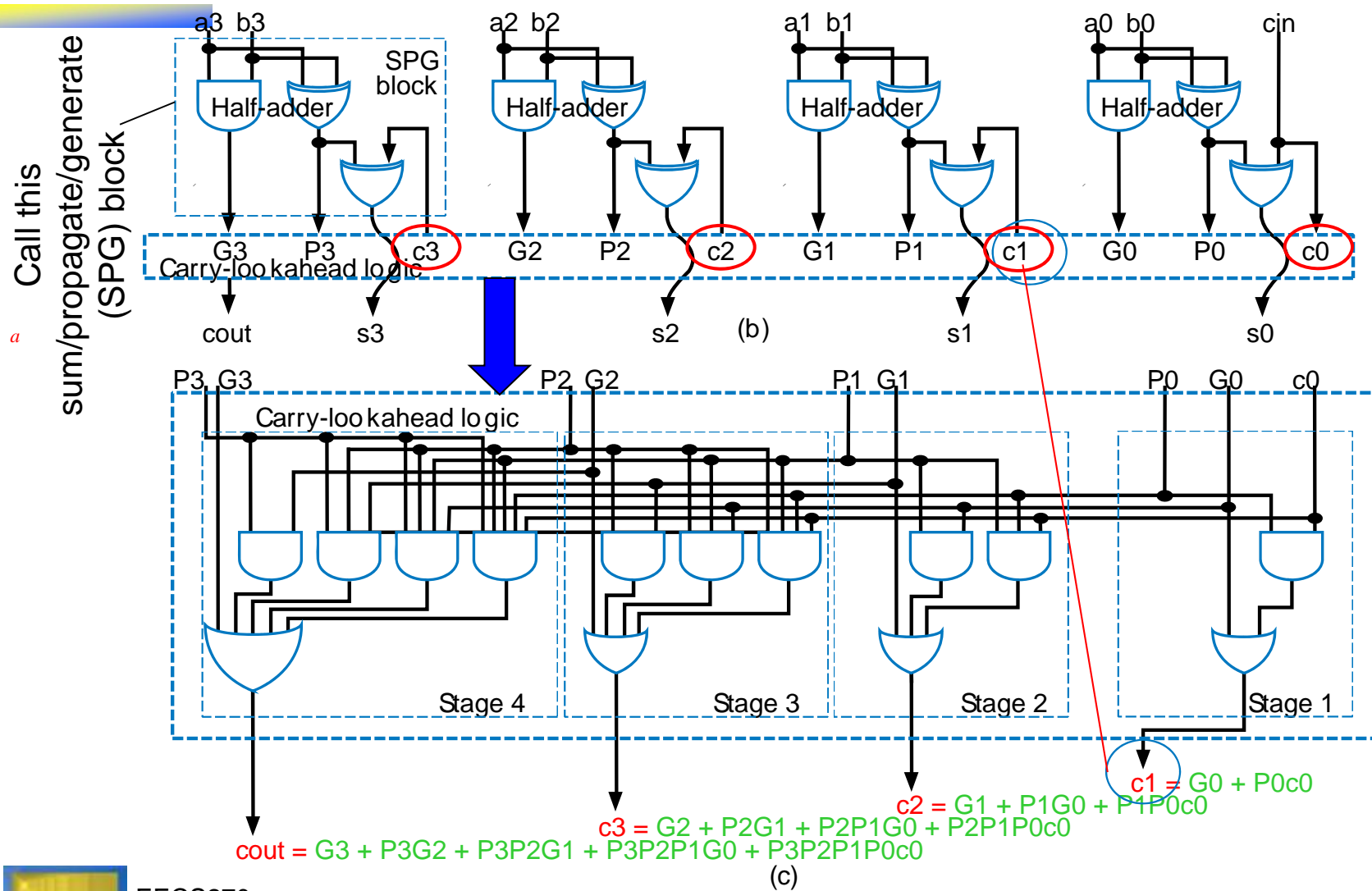
$$c_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0$$

$$\text{cout} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$$

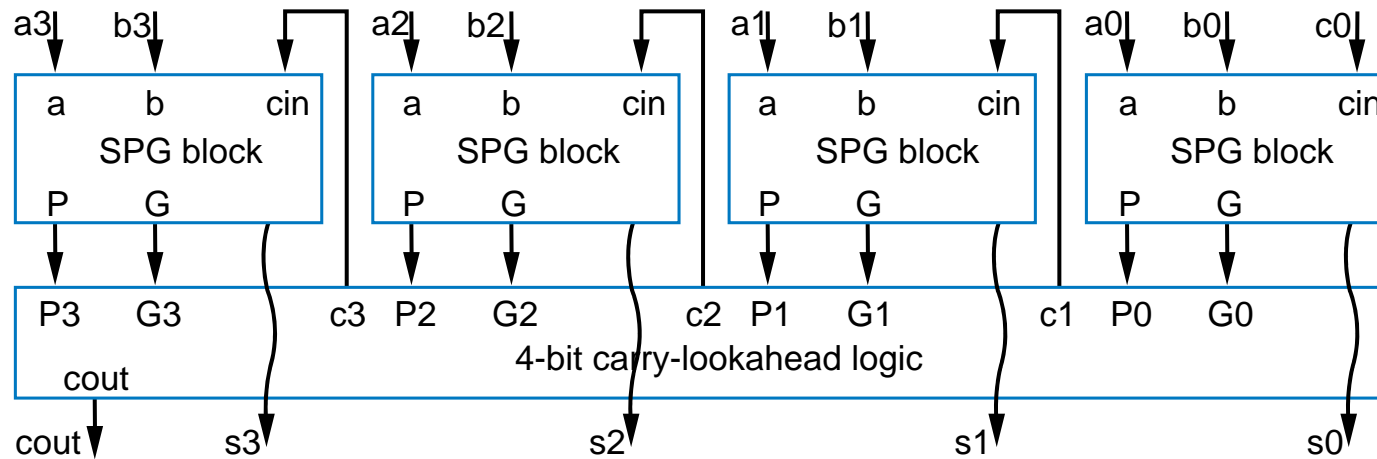
Much simpler than the “first-attempt” look-ahead



Common form of look-ahead



Carry-lookahead adder -- high-level view

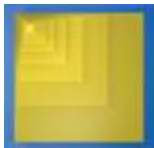
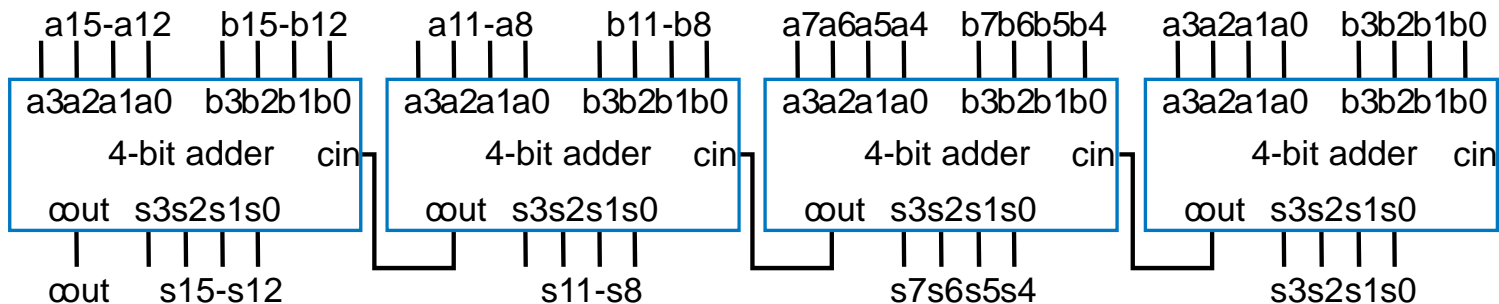
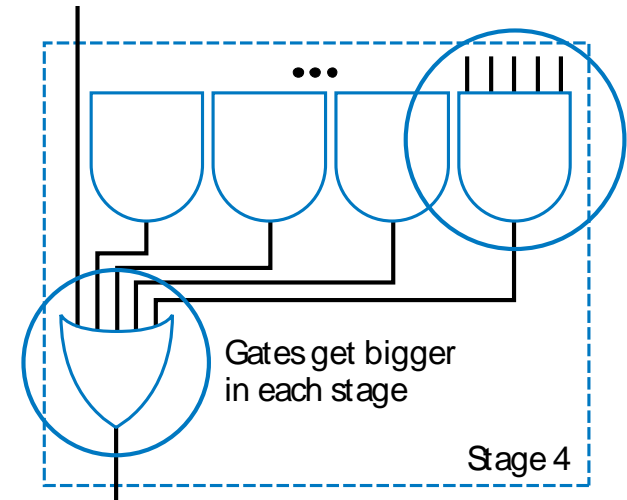


- Fast -- only **4 gate delays**
 - Each stage has SPG block with 2 gate levels
 - Carry-lookahead logic quickly computes the carry from the propagate and generate bits using 2 gate levels inside
- Reasonable number of gates -- 4-bit adder has only **26 gates**
- 4-bit adder comparison (gate delays, gates)
 - Carry-ripple: (8, 20)
 - Two-level: (2, 500)
 - CLA: (4, 26)
 - o Nice compromise

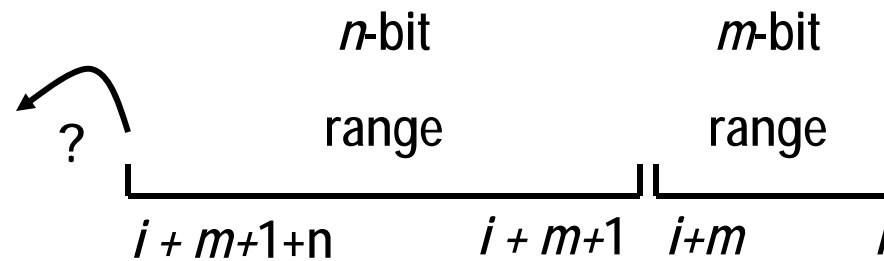


Carry-lookahead adder – 32-bit?

- There is still one problem:
Gates get bigger in each stage
 - 4th stage has 5-input gates
 - 32nd stage would have 33-input gates
 - Too many inputs for one gate
 - Would require building a “gate-tree”, meaning more levels (slower) and more gates (bigger)
- One solution: Connect 4-bit CLA adders in ripple manner
 - But slow (4 + 4 + 4 + 4 gate delays)



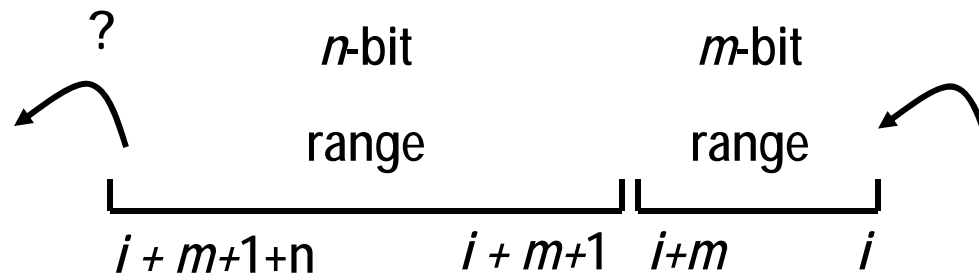
Generate for General Bit Ranges



$$G_{i+m+1+n,i} = G_{i+m+1+n,i+m+1} \mid (G_{i+m,i} \& P_{i+m+1+n,i+m+1})$$



Propagate for General Bit Ranges

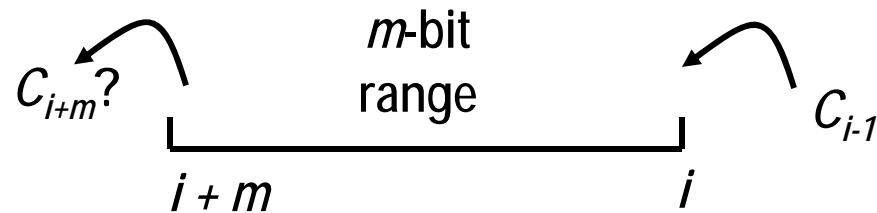


$$P_{i+m+1+n,i} = P_{i+m+1+n,i+m+1} \& P_{i+m,i}$$



So, What is My Carry?

Given a carry-in to a bit range and the generate and propagate signals associated with that bit range, how do we compute the carry-out?

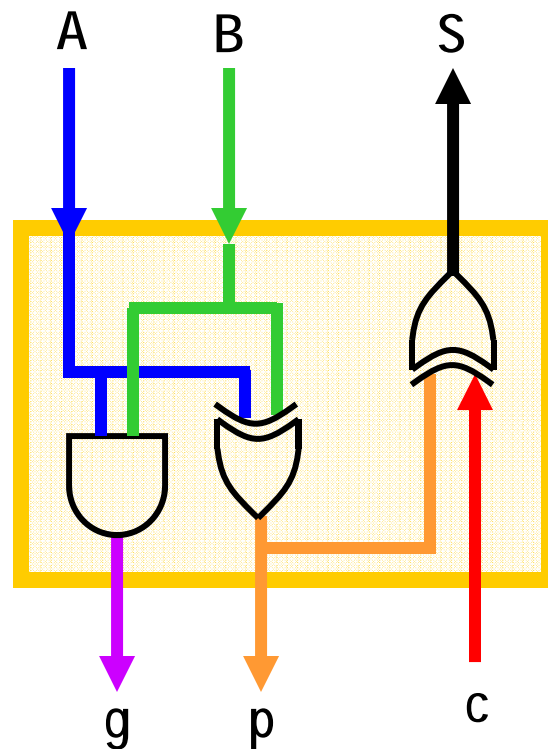


$$C_{i+m} = G_{i+m,i} \mid (P_{i+m,i} \& C_{i-1})$$

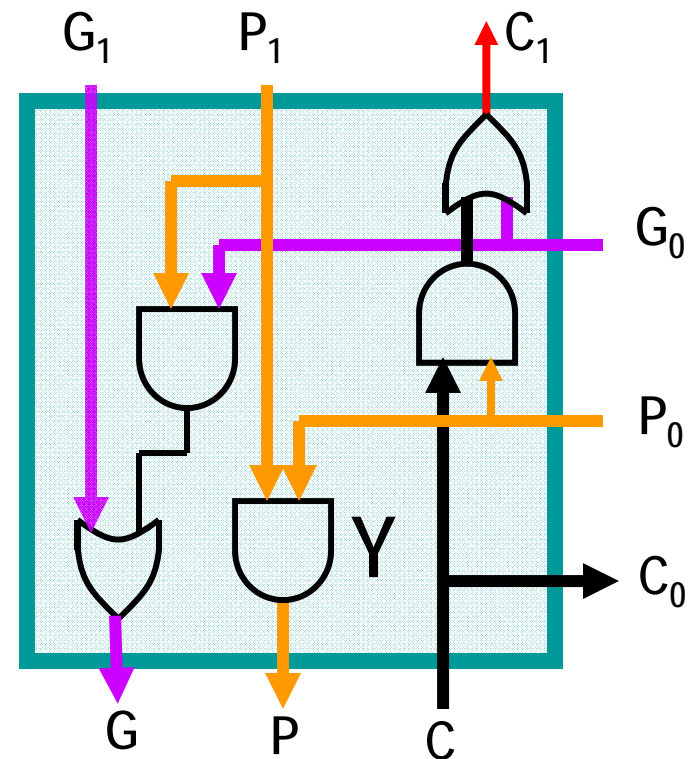


Composing SPG blocks for multi-bit ranges

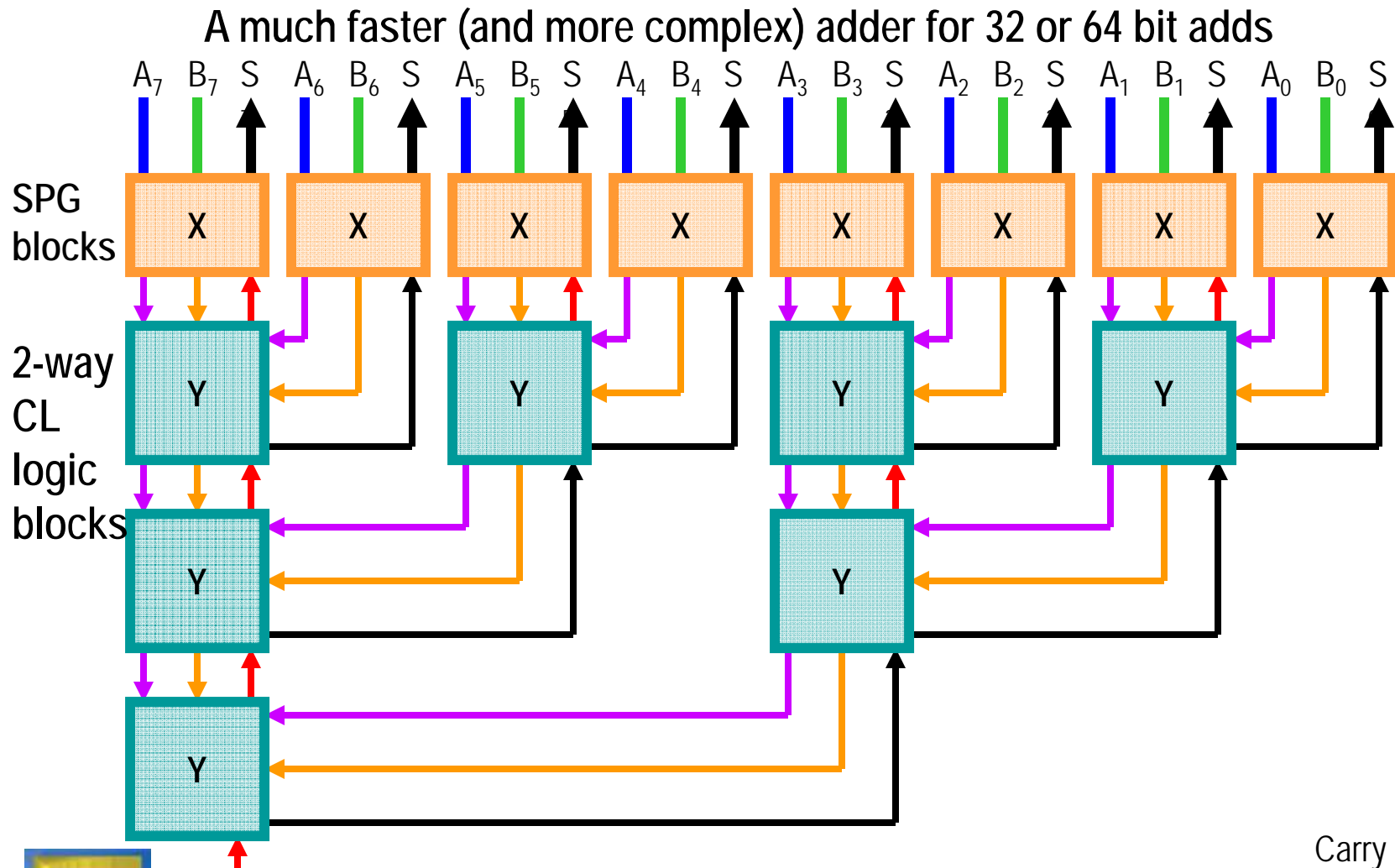
SPG block



2-way carry look-ahead logic



8-bit Carry Look-ahead Adder

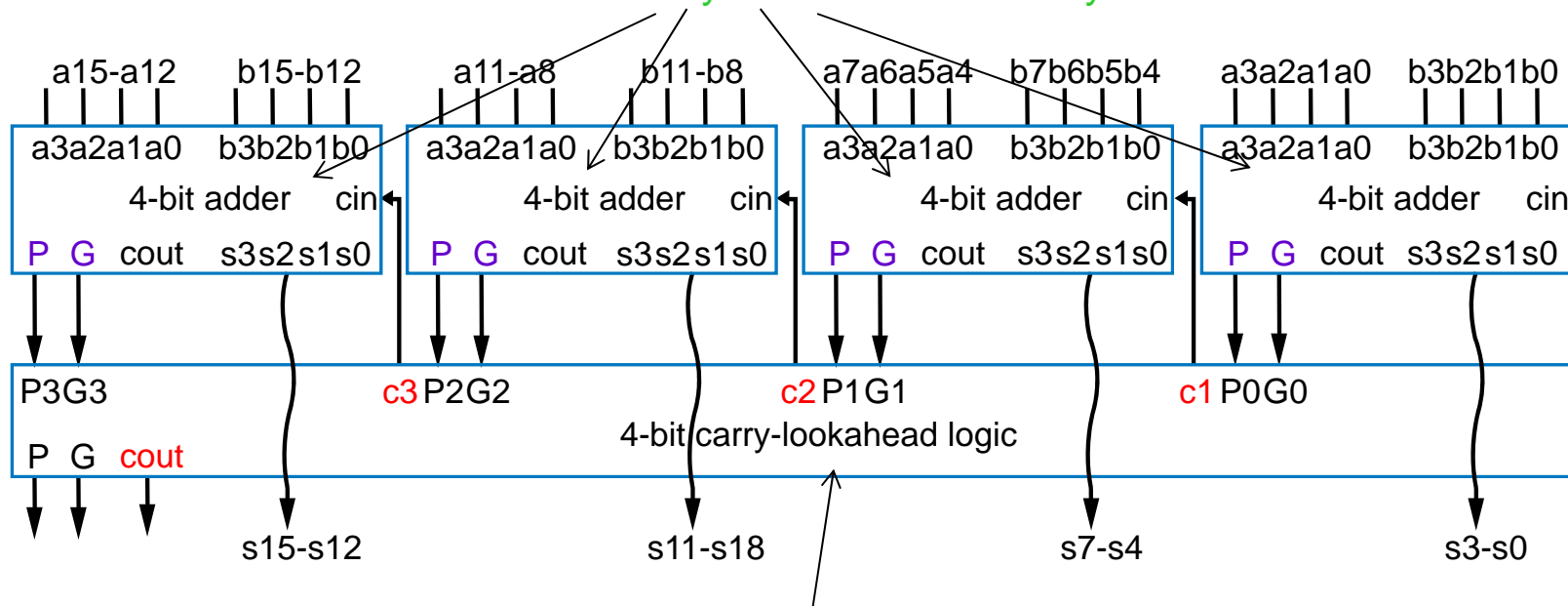




Hierarchical carry-lookahead adders

- Better solution -- Rather than rippling the carries, just *repeat* the carry-lookahead concept
 - Requires minor modification of 4-bit CLA adder to **output P and G**

These use carry-lookahead internally

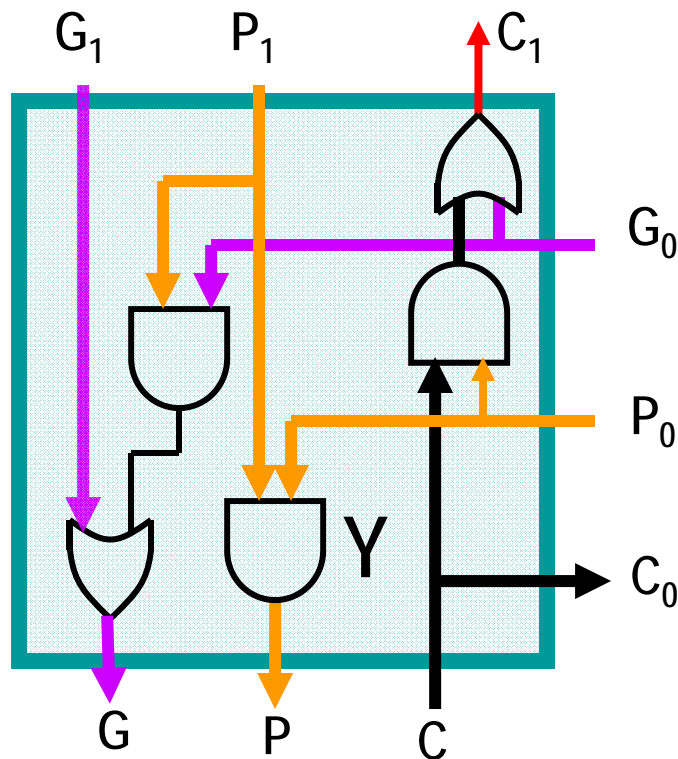


Second level of carry-lookahead

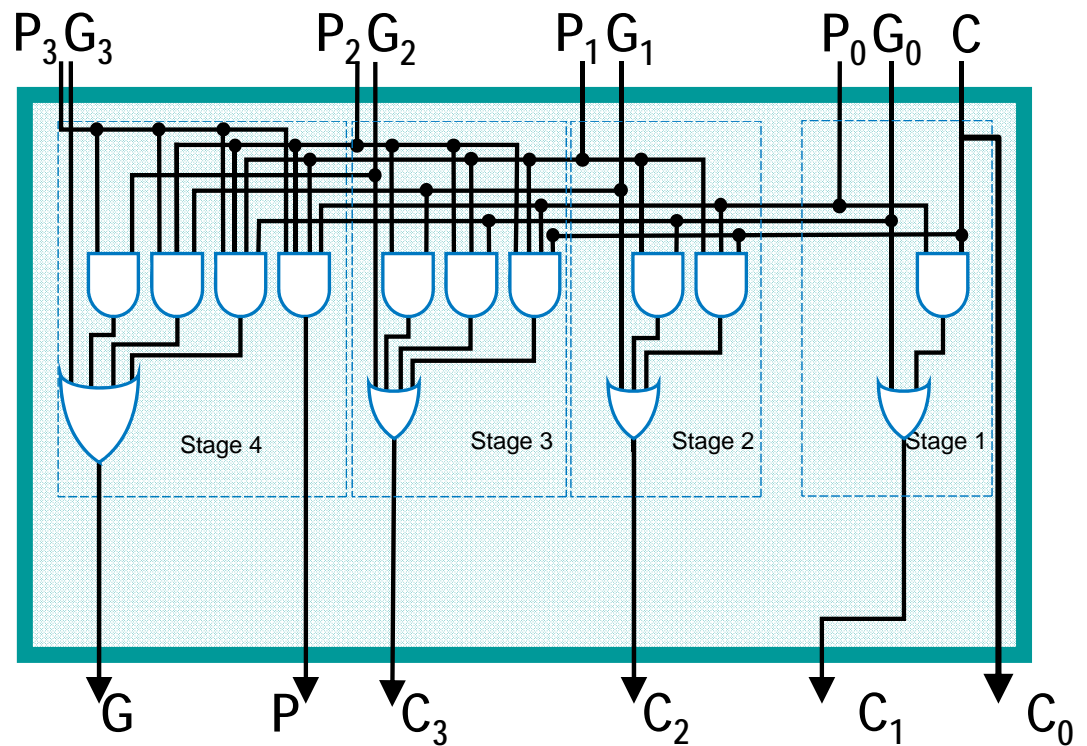


4-way carry look-ahead logic (4-way CLA)

2-way carry look-ahead logic



4-way carry look-ahead logic



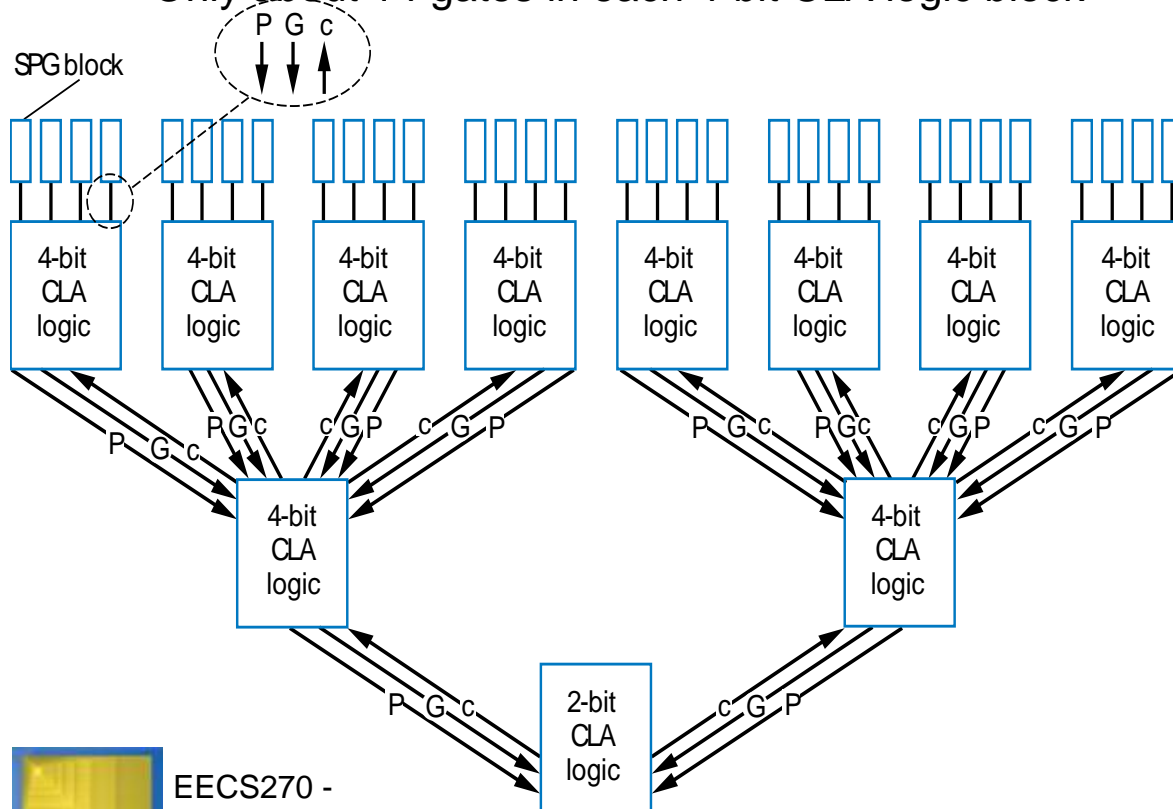
Carry-lookahead blocks

- SPG block
 - 3 gates
 - 1 gate-delay going down, and 1 gate-delay coming up
- 2way carry-lookahead logic
 - 5 gates
 - 2 gate-delays going down, and 2 gate-delays coming up
- 4way carry-lookahead logic
 - 9 gates to compute the 4 carry-bits, 5 gates for overall P&G =14
 - 2 gate-delays going down, and 2 gate-delays coming up



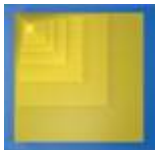
Hierarchical carry-lookahead adders

- Hierarchical CLA concept can be applied to larger adders
- 32-bit hierarchical CLA
 - Only about 12 gate delays (down:1 for SPG block, then 2 per CLA level, excluding last – up: 2 per CLA level, 1 per SPG block)
 - Only about 14 gates in each 4-bit CLA logic block

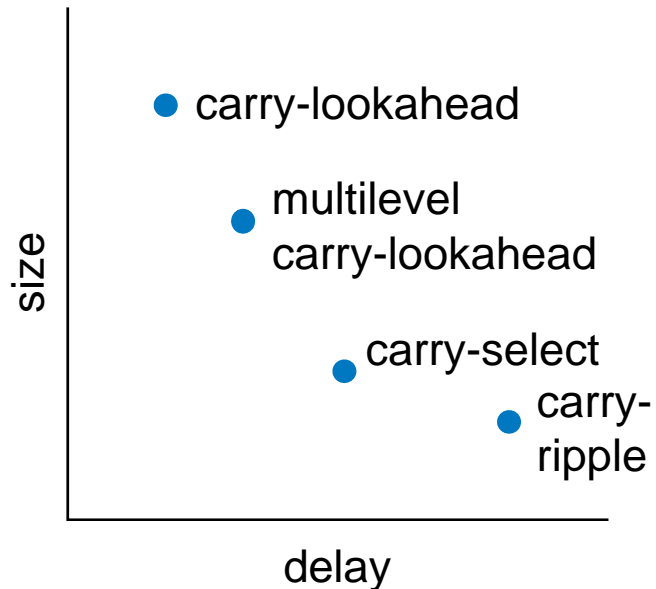


How many gate delays for 64-bit hierarchical CLA, using 4-bit CLA logic?

-16 CLA-logic blocks in 1st level,
-4 in 2nd,
-1 in 3rd
--- so still just 12 gate delays



Adder tradeoffs



-Hierarchical carry look-ahead

adders are among the fastest adders of practical size

-Very common in practice

-Developed by Kogge&Stone in the '70s

-a.k.a Kogge-Stone adders

- Designer picks the adder that satisfies particular delay and size requirements
 - May use different adder types in different parts of same design
 - Faster adders on critical path, smaller adders on non-critical path



32-bit adders – sizes & delays summary

32-bit adder design	Size	Delay
Ripple carry (RCA)	$32 \times 5g = \mathbf{160 \text{ gates}}$	$32 \times 2gd = \mathbf{64 \text{ gate-delays}}$
2-level synthesis	~2B gates	2 gate-delays
Carry-select (CSA)* cascaded	15adders+7muxes(2x1-5bits) $15 \times 20g + 7 \times 15 = \mathbf{405 \text{ gates}}$	$8 + 7 \times 2 = \mathbf{22 \text{ gate-delays}}$
Carry-select (CSA)* encapsulated	8-bitCSA: 3adders+1MUX=75 16-bitCSA: 3[8bitCSA]+1MUX=252 32-bitCSA: 3[16bitCSA]+1MUX=807 807 gates	8-bitCSA: adder+MUX=10 16-bitCSA: 10+2=12gd 32-bitCSA: 12+2=14gd 14 gate-delays
Carry look-ahead (CLA) quad-tree [slide 24]	32 SPG + 10 4-way + 1 2-way = 291 gates	12 gate-delays (SPG + 3 levels of CL logic – last stage only 2 gd)
Carry look-ahead (CLA) binary-tree [slide 19]	32 SPG + 31 2-way = 251 gates	20 gate-delays (SPG + 5 levels of CL logic – last stage only 2 gd)