CS150 - EE141/241A
Fall 2014
Digital Design and Integrated Circuits

Instructors:
John Wawrzynek and Vladimir Stojanovic

Lecture 15

Outline

- Adder review, subtraction, carry-select
- Carry-lookahead
- Bit-serial addition, summary

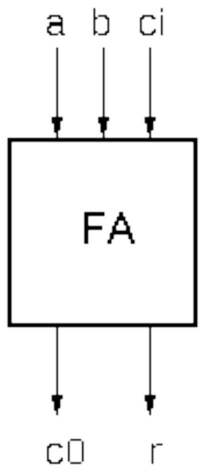
Adder review, subtraction, carry-select

Carry-ripple Adder Revisited

- Each cell:

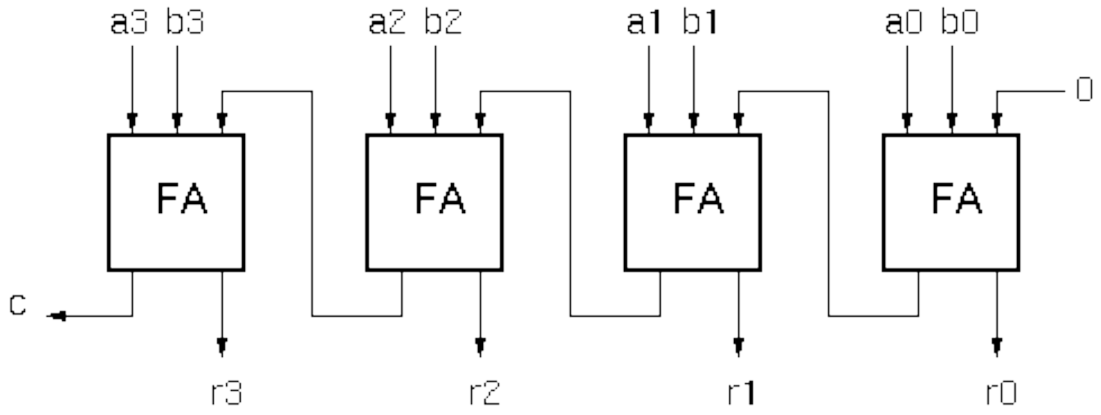
  $r_i = a_i \oplus b_i \oplus c_{in}$

  $c_{out} = a_i c_{in} + a_i b_i + b_i c_{in} = c_{in}(a_i + b_i) + a_i b_i$



"Full adder cell"

- 4-bit adder:



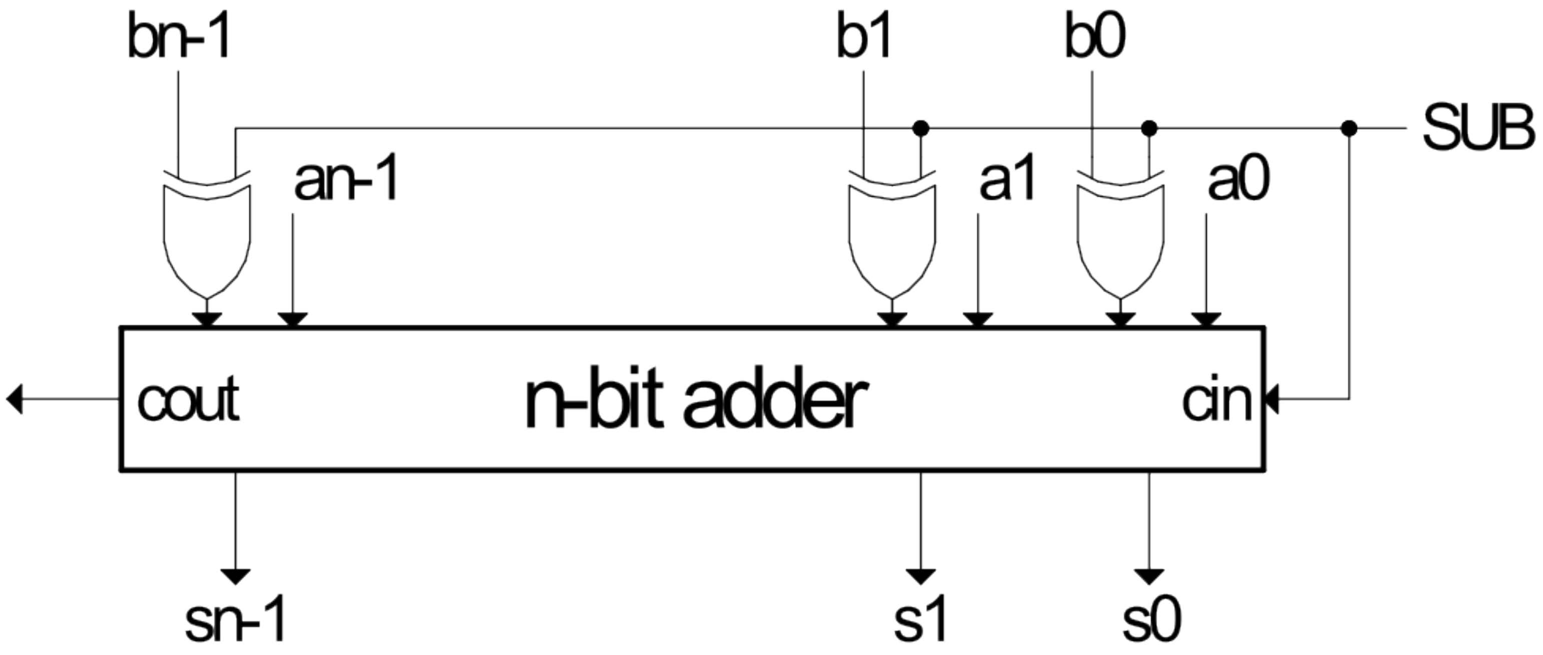
- What about subtraction?

Subtractor/Adder

A - B = A + (-B)

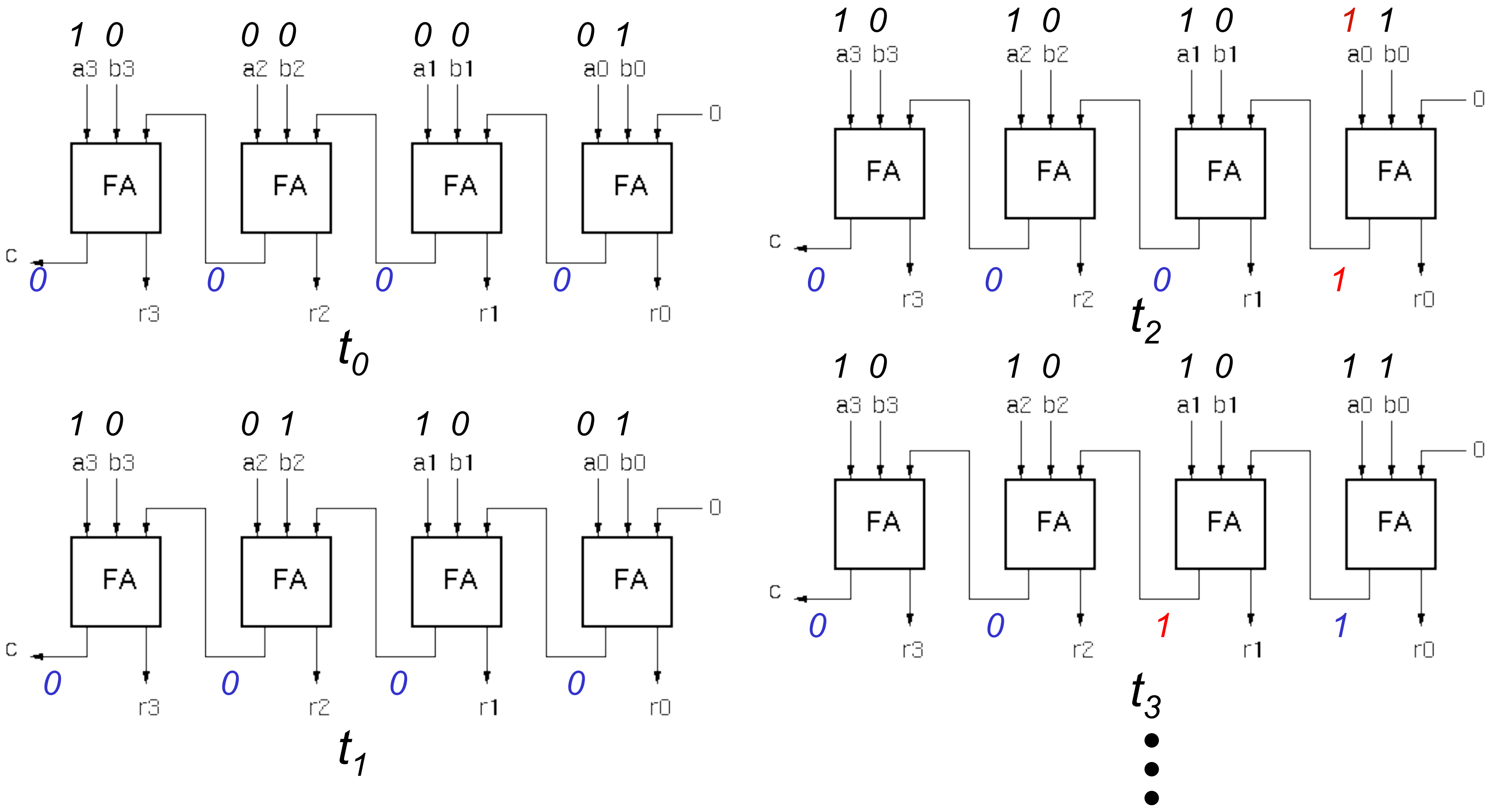
How do we form -B?

1. *complement B*
2. *add 1*



Delay in Ripple Adders

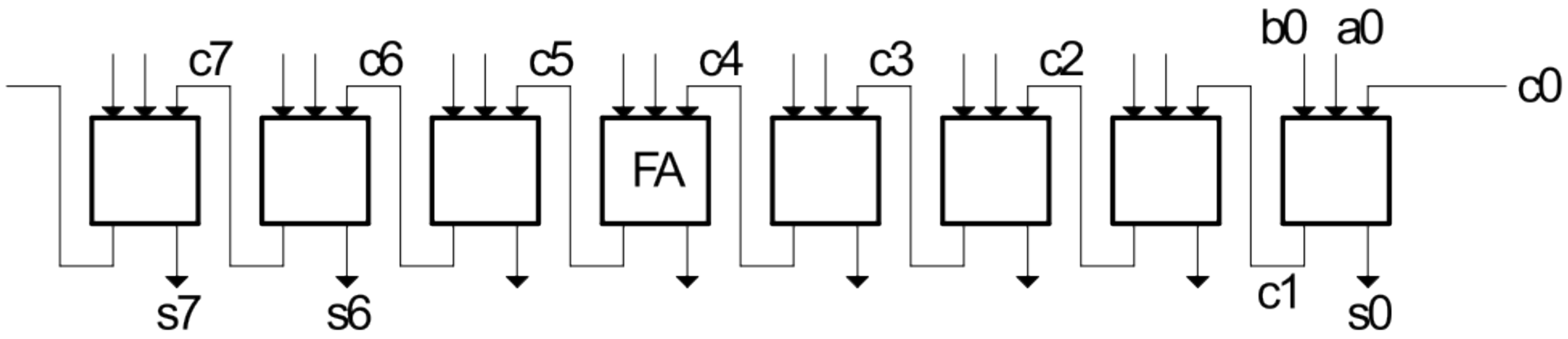
- Ripple delay amount is a function of the data inputs:



- However, we usually only consider the worst case delay on the critical path. There is usually at least one set of input data that exposes the worst case delay.

Adders (cont.)

Ripple Adder

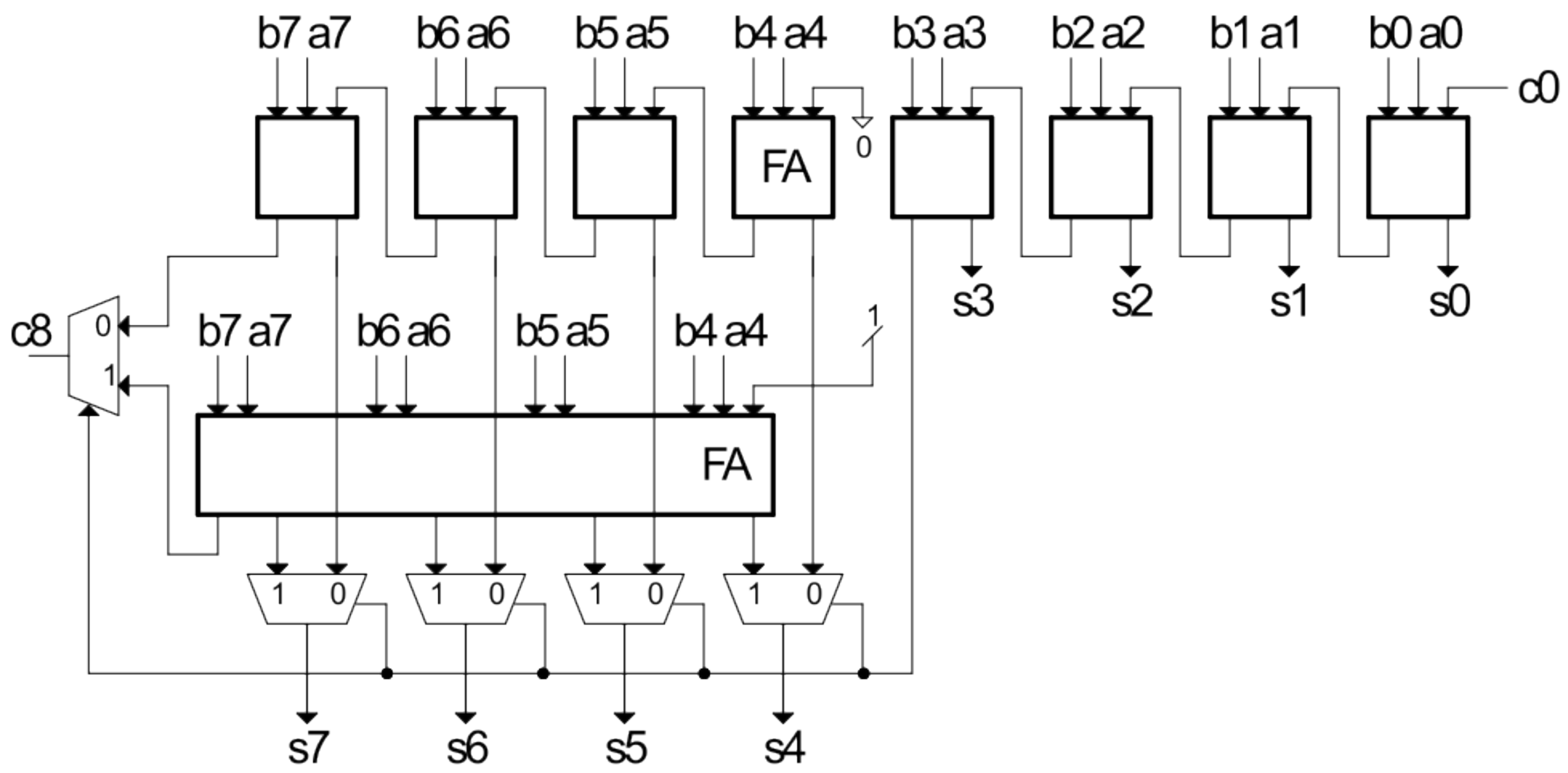


Ripple adder is inherently slow because, in worst case s7 must wait for c7 which must wait for c6 …

T α n, Cost α n

How do we make it faster, perhaps with more cost?

Carry Select Adder

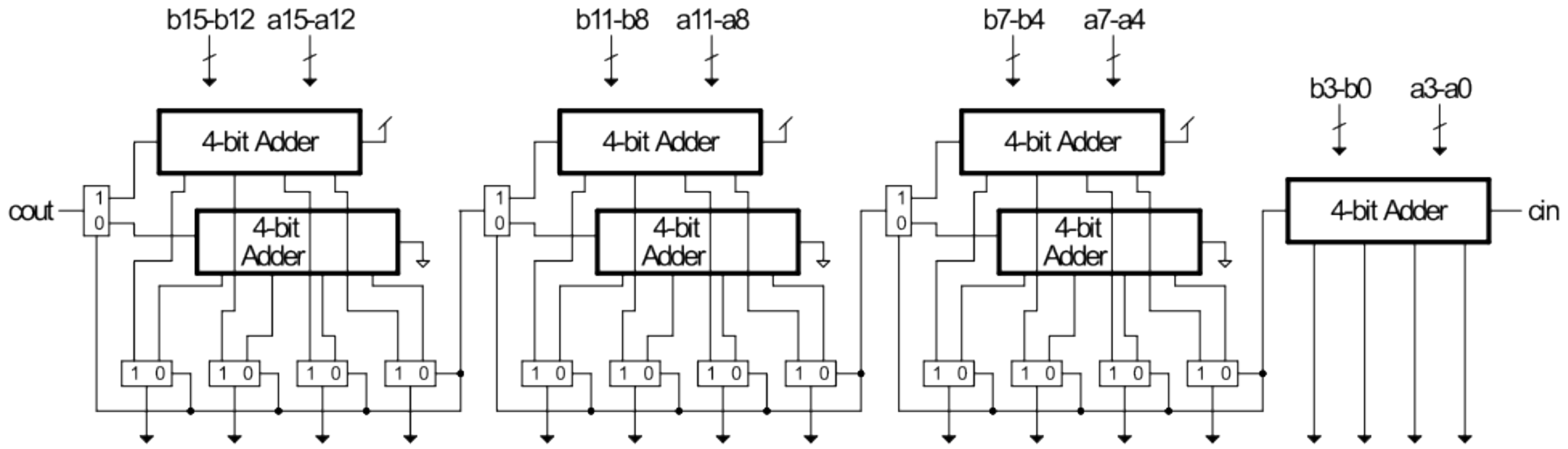


$$T = T_{ripple\_adder} / 2 + T_{MUX}$$

$$COST = 1.5 * COST_{ripple\_adder} + (n/2 + 1) * COST_{MUX}$$

Carry Select Adder

- Extending Carry-select to multiple blocks

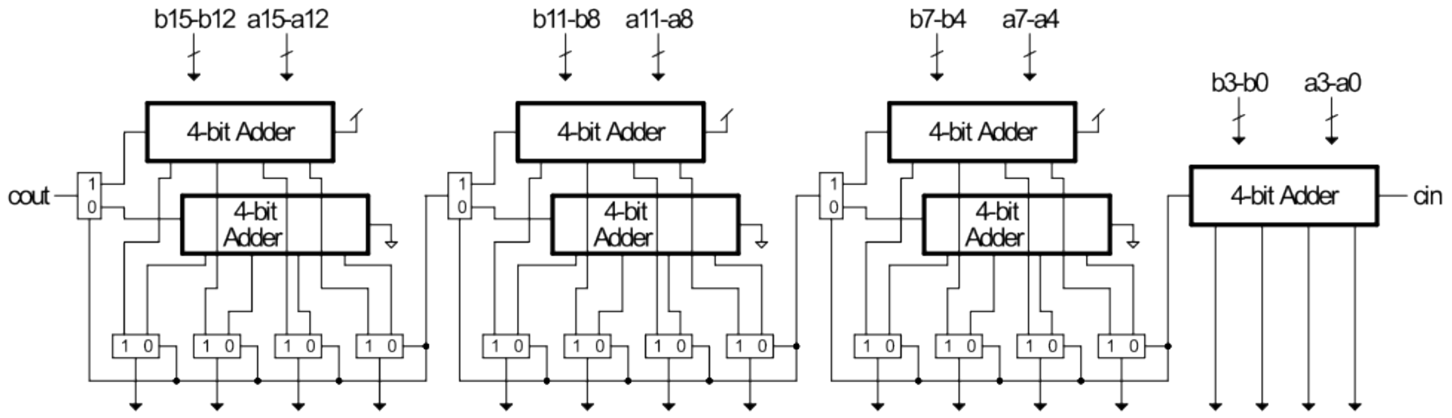


- What is the optimal # of blocks and # of bits/block?
  - If blocks too small delay dominated by total mux delay
  - If blocks too large delay dominated by adder ripple delay

$\boxed{\sqrt{N} \text{ stages of } \sqrt{N} \text{ bits}}$

T α sqrt(N),
*Cost ≈2*ripple + muxes*

Carry Select Adder



- ❑ Compare to ripple adder delay:

  $T_{total} = 2 \text{ sqrt}(N)\, T_{FA} - T_{FA},$ assuming $T_{FA} = T_{MUX}$

  For ripple adder $T_{total} = N\, T_{FA}$

  "cross-over" at N=3, Carry select faster for any value of N>3.

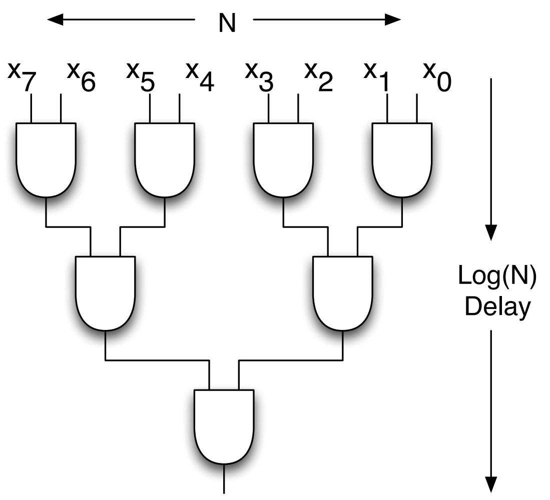
- ❑ Is sqrt(N) really the optimum?
  - From right to left increase size of each block to better match delays
  - Ex: 64-bit adder, use block sizes [12 11 10 9 8 7 7]
- ❑ How about recursively defined carry select?

Carry-lookahead

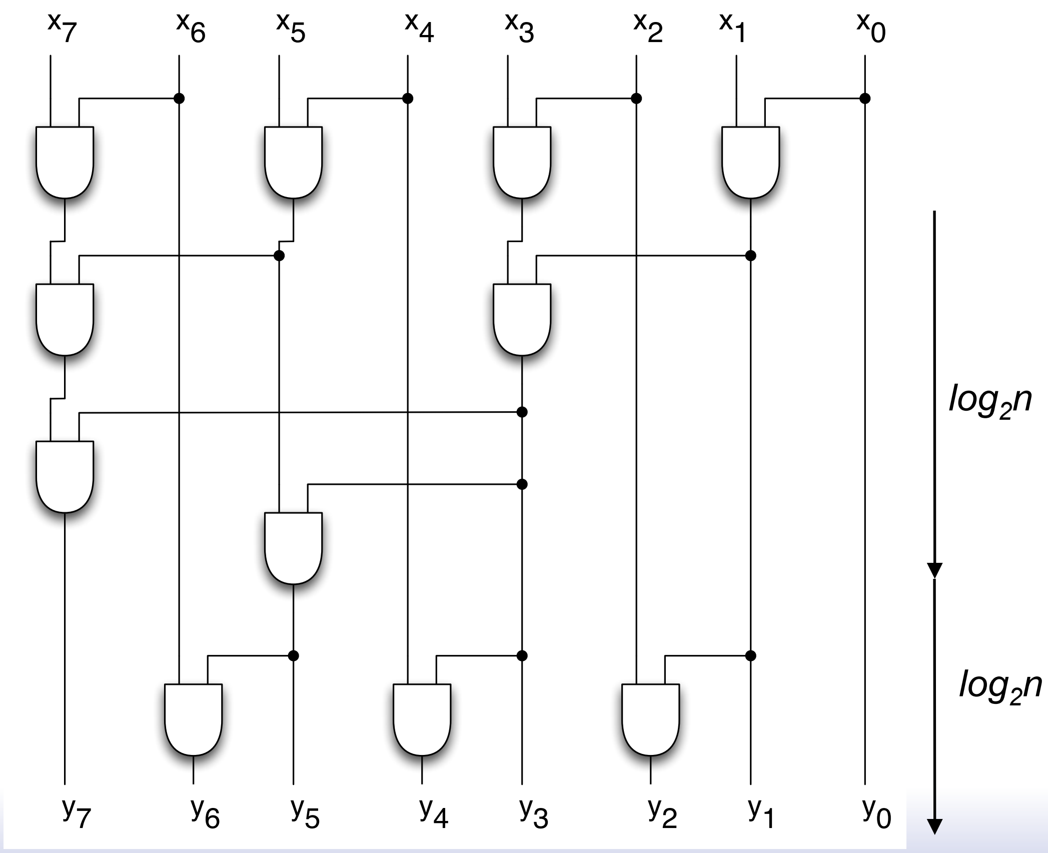
Adders with Delay α log(n)

Can carry generation can be made to be a kind of "reduction operation"?

Lowest delay for a reduction is a balanced tree.



- *But in this case all intermediate values are required.*
- *Idea is to use "Parallel Prefix" to compute the carries.*


x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
log_2n
log_2n
y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0

$y_0 = x_0$

$y_1 = x_0x_1$

$y_2 = x_0x_1x_2$

.

.

.

Parallel Prefix generally requires that the operation be commutative and associative.

Carry Look-ahead Adders

- How do we arrange carry generation to be commutative and associative?
- Reformulate basic adder stage:

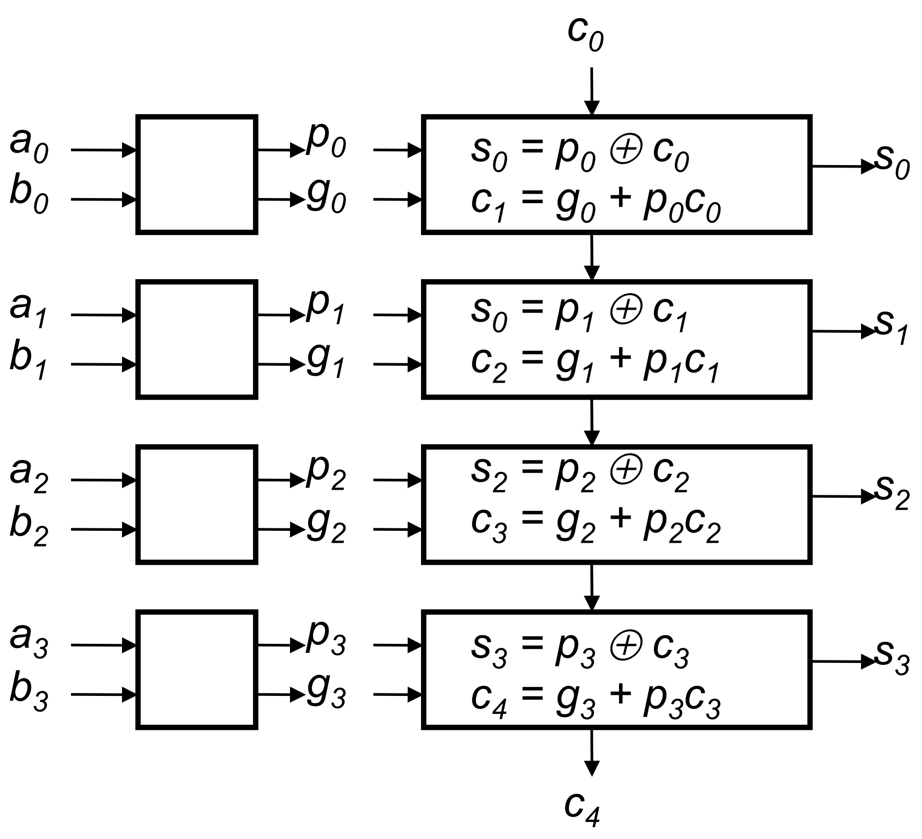
a b c_i	c_{i+1}	s	
0 0 0	0	0	*carry "kill"*
0 0 1	0	1	$k_i = a_i' \, b_i'$
0 1 0	0	1	
0 1 1	1	0	*carry "propagate"*
1 0 0	0	1	$p_i = a_i \oplus b_i$
1 0 1	1	0	
1 1 0	1	0	*carry "generate"*
1 1 1	1	1	$g_i = a_i \, b_i$

$$\boxed{\begin{aligned} c_{i+1} &= g_i + p_i c_i \\ s_i &= p_i \oplus c_i \end{aligned}}$$

Carry Look-ahead Adders

- Ripple adder using p and g signals:

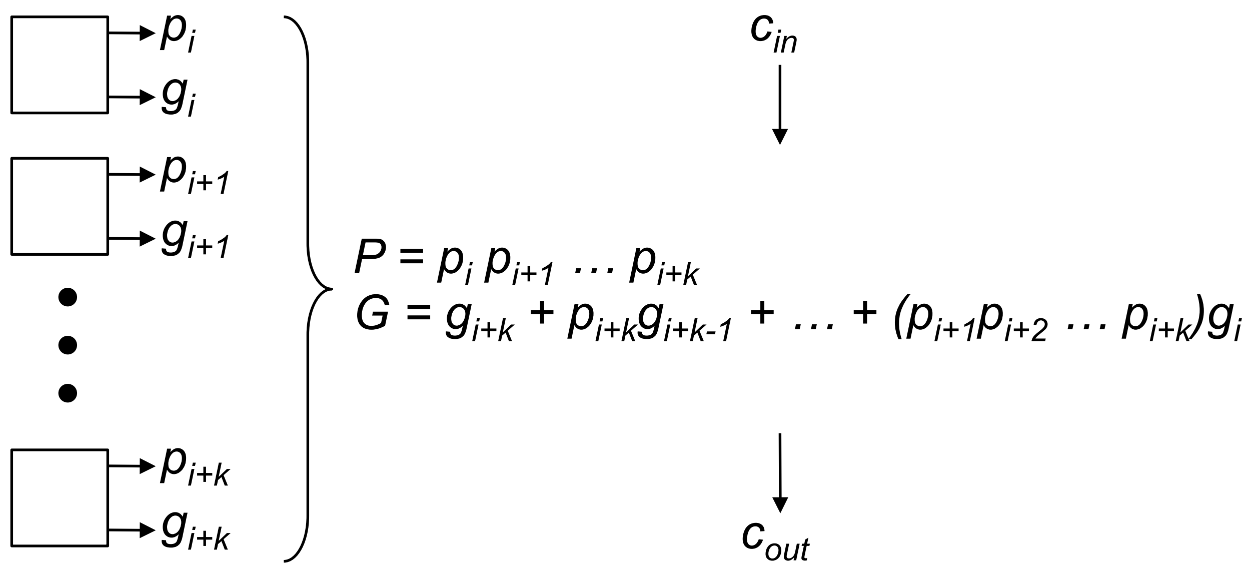
$$p_i = a_i \oplus b_i$$
$$g_i = a_i\, b_i$$



- So far, no advantage over ripple adder: $T \alpha N$

Carry Look-ahead Adders

- "Group" propagate and generate signals:



$$P = p_i\, p_{i+1} \dots p_{i+k}$$
$$G = g_{i+k} + p_{i+k}g_{i+k-1} + \dots + (p_{i+1}p_{i+2} \dots p_{i+k})g_i$$

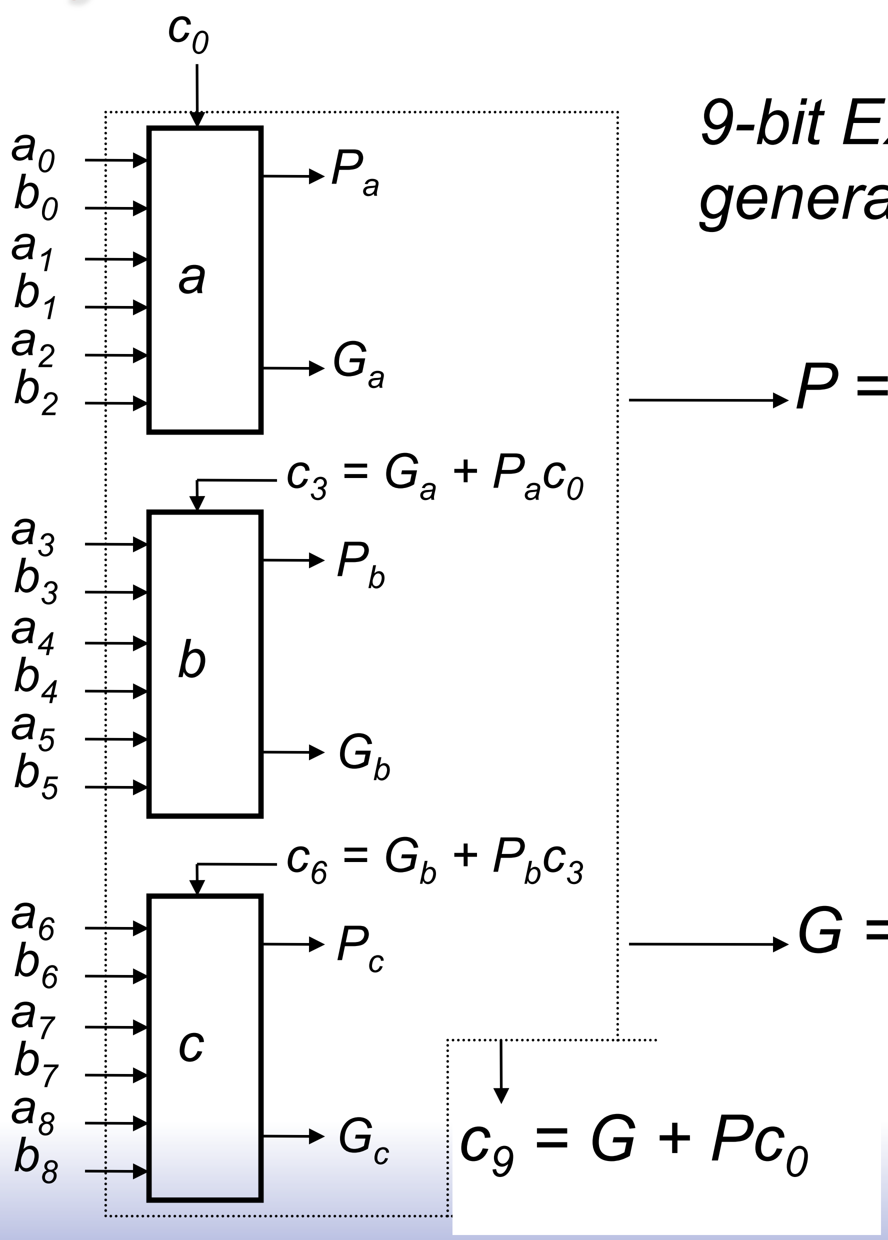
- P true if the group as a whole propagates a carry to c_{out}
- G true if the group as a whole generates a carry

$$\boxed{c_{out} = G + Pc_{in}}$$

- Group P and G can be generated hierarchically.

Carry Look-ahead Adders

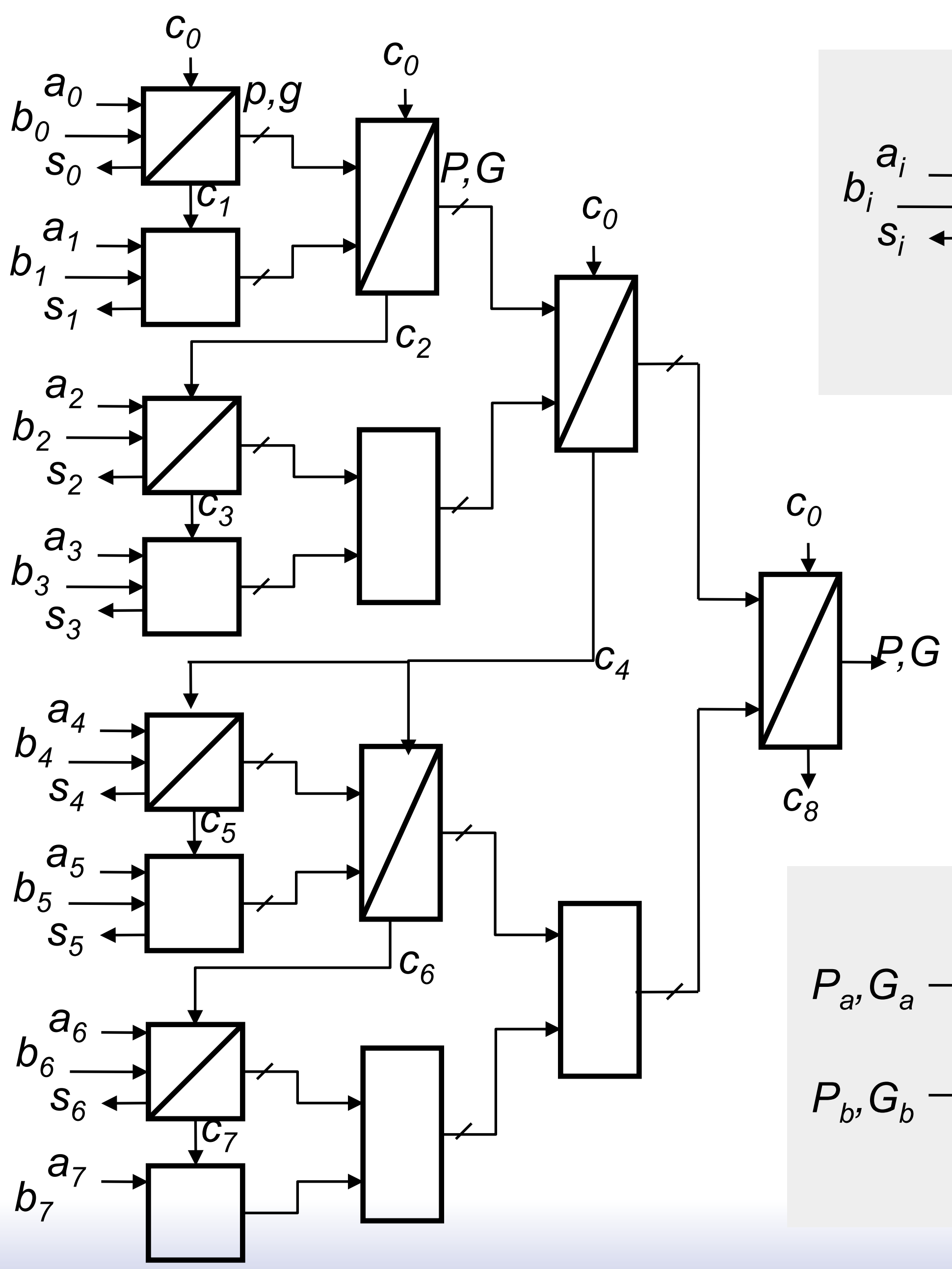
9-bit Example of hierarchically generated P and G signals:

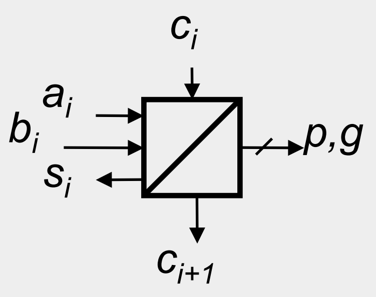


$$P = P_a P_b P_c$$

$$G = G_c + P_c G_b + P_b P_c G_a$$

$$c_9 = G + P c_0$$



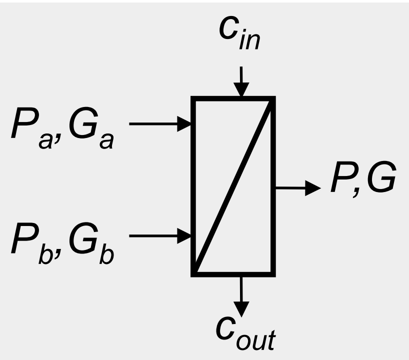


$p = a \oplus b$
$g = ab$

$s = p \oplus c_i$

$c_{i+1} = g + c_i p$

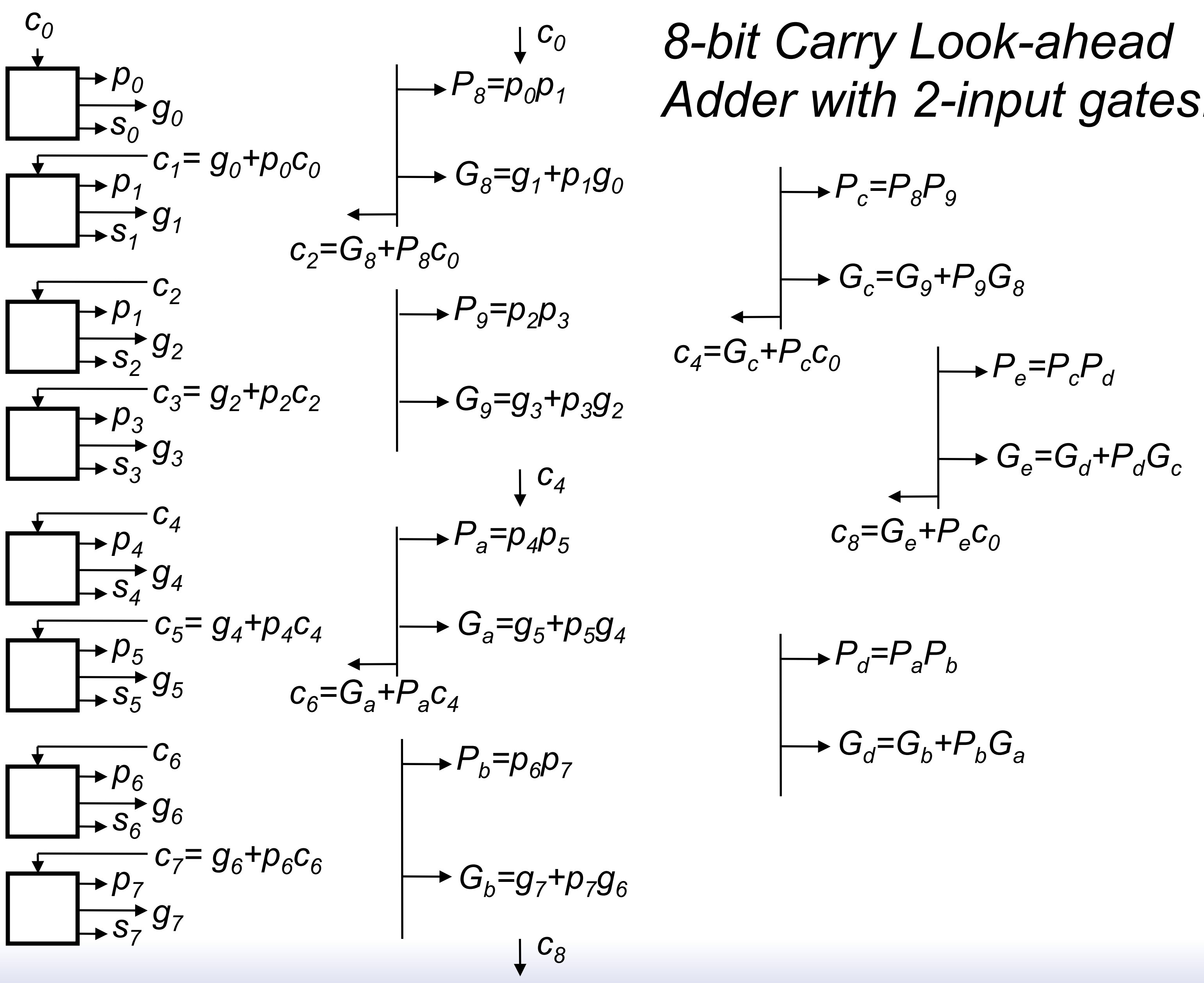
8-bit Carry Look-ahead Adder



$P = P_a P_b$
$G = G_b + G_a P_b$

$C_{out} = G + c_{in} P$

8-bit Carry Look-ahead Adder with 2-input gates.



Full Parallel-Prefix Carry Look-ahead Adders

- Generate all carries directly (no grouping):

$c_0 = 0$

$c_1 = g_0$

$c_2 = g_1 + p_1c_1 = g_1 + p_1g_0$

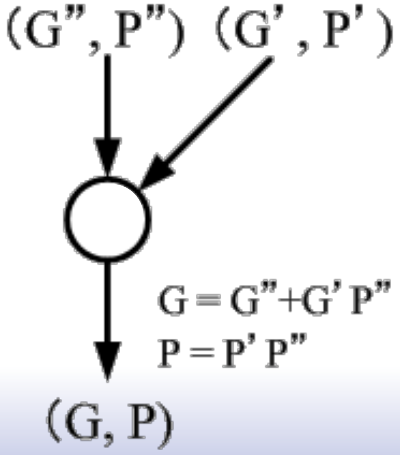
$c_3 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_1p_2g_0$

$c_4 = g_3 + p_3c_3 = g_3 + p_3g_2 + p_3p_2g_1 + p_4p_3p_2g_0$

.

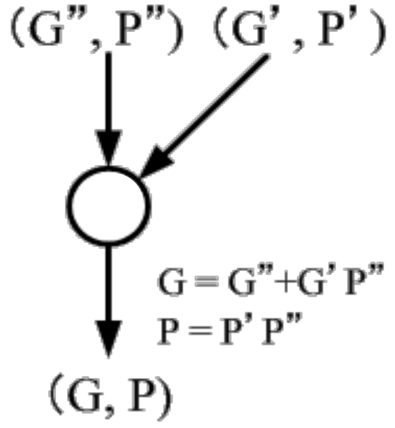
.

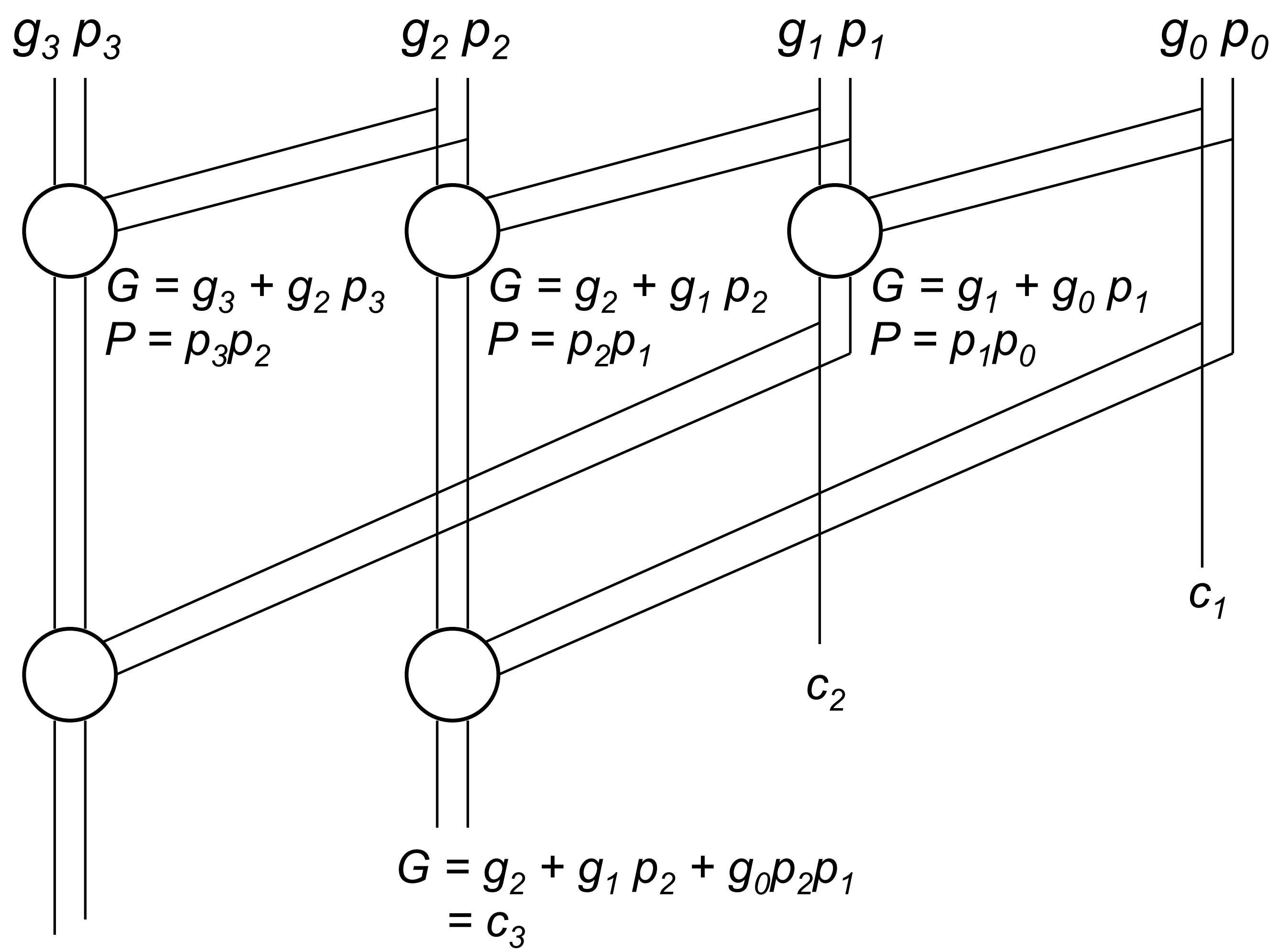
.



Binary (G, P) operator

Parallel Prefix Adder Example



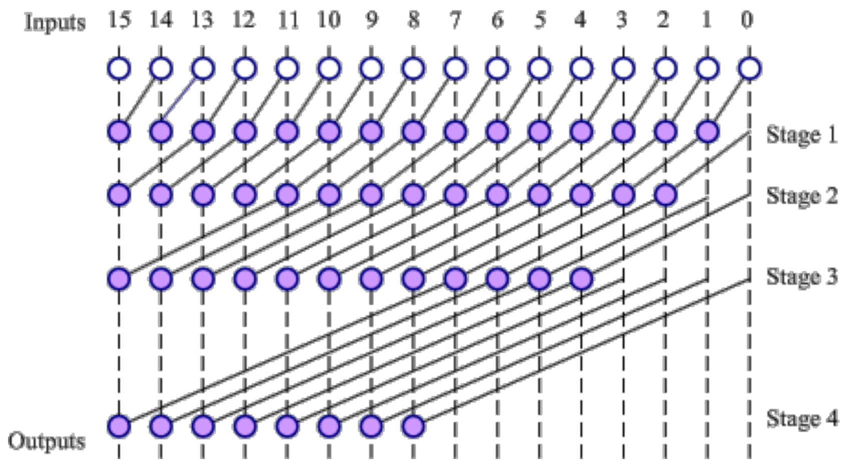


$G = g_2 + g_1\ p_2 + g_0 p_2 p_1$
$= c_3$

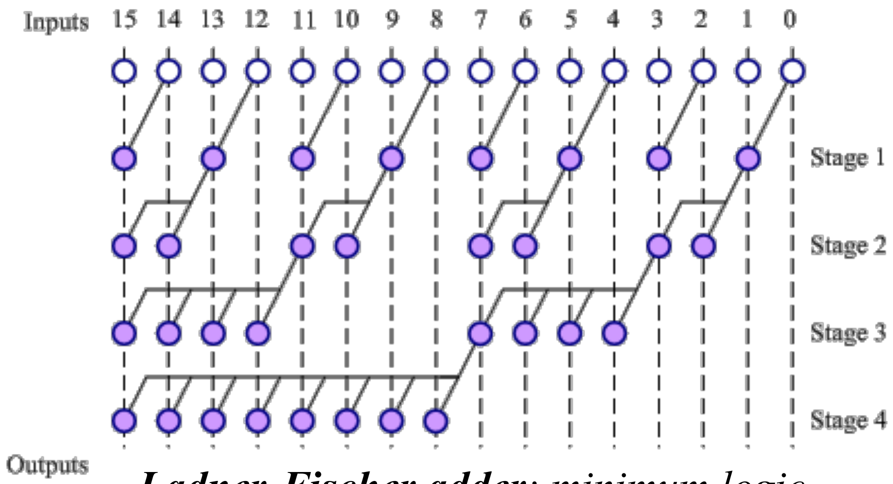
$G = g_3 + g_2\ p_3 + (g_1 + g_0 p_1) p_3 p_2$
$= g_3 + g_2 p_3 + g_1 p_3 p_2 + g_0 p_3 p_2 p_1$
$= c_4$

$$\boxed{s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i}$$

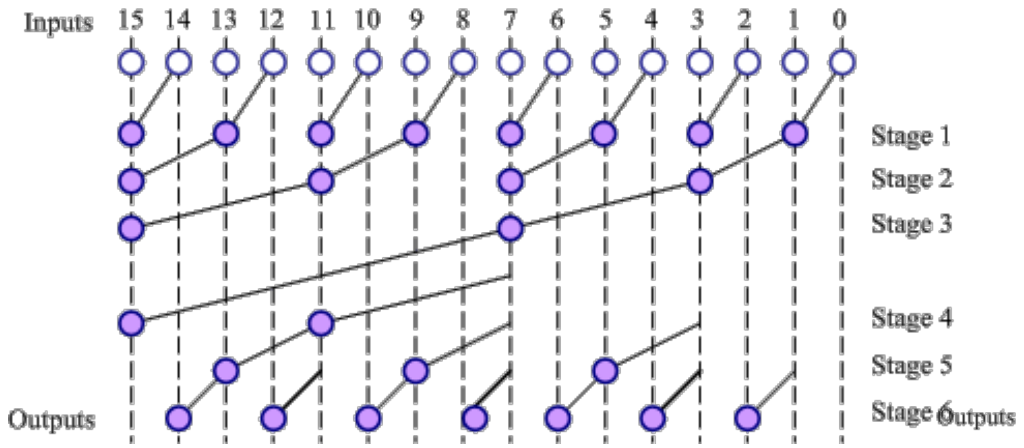
Other CLA Parallel Prefix Architectures



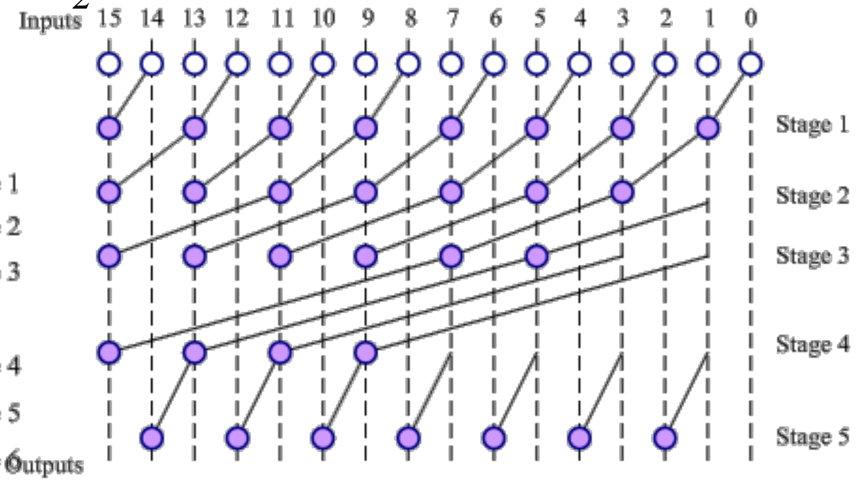
Kogge-Stone adder: *minimum logic depth, and full binary tree with minimum fan-out, resulting in a fast adder but with a large area*



Ladner-Fischer adder: *minimum logic depth, large fan-out requirement up to n/2*



Brent-Kung adder: *minimum area, but high logic depth*



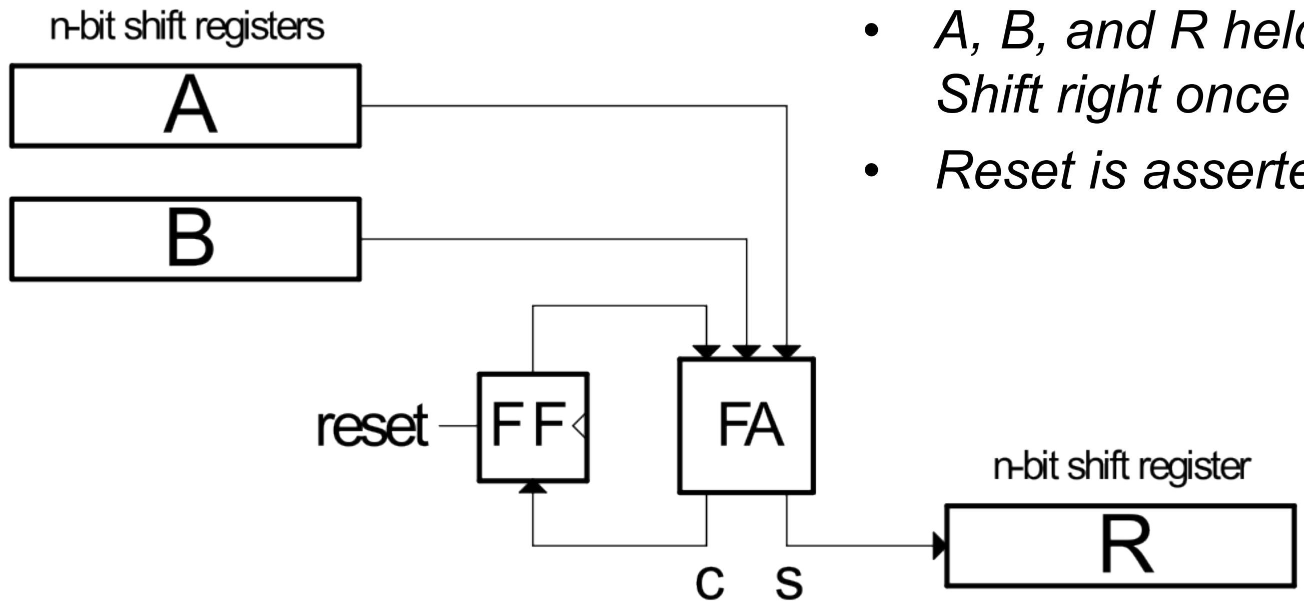
Han-Carlson adder: *hybrid design combining stages from the Brent-Kung and Kogge-Stone adder*

Carry look-ahead Wrap-up

- Adder delay O(logN).
- Cost?
- Can be applied with other techniques. Group P & G signals can be generated for sub-adders, but another carry propagation technique (for instance ripple) used within the group.
    - For instance on FPGA. Ripple carry up to 32 bits is fast (1.25ns), CLA used to extend to large adders. CLA tree quickly generates carry-in for upper blocks.
- Other more complex techniques exist that can bring the delay down below O(logN), but are only efficient for very wide adders.

Bit-serial Addition, Adder summar

Bit-serial Adder

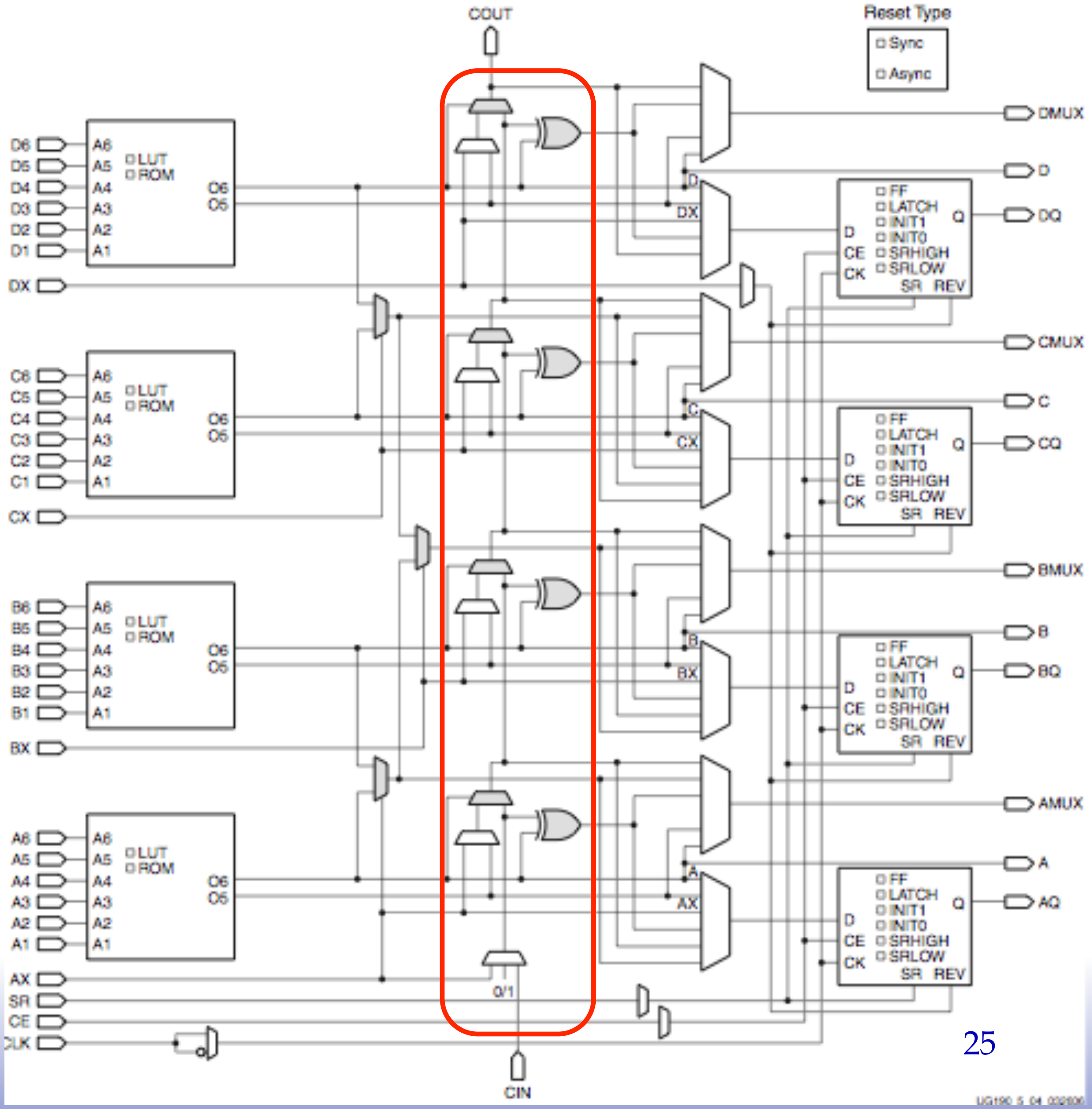


- *A, B, and R held in shift-registers. Shift right once per clock cycle.*
- *Reset is asserted by controller.*

- Addition of 2 n-bit numbers:
  - takes n clock cycles,
  - uses 1 FF, 1 FA cell, plus registers
  - the bit streams may come from or go to other circuits, therefore the registers might not be needed.

Adders on the Xilinx Virtex-5

- Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions.
- Cin to Cout (per bit) delay = 40ps, versus 900ps for F to X delay.
- 64-bit add delay = 2.5ns.



Adder Final Words

Type	Cost	Delay
Ripple	O(N)	O(N)
Carry-select	O(N)	O(sqrt(N))
Carry-lookahead	O(N)	O(log(N))
Bit-serial	O(1)	O(N)

- Dynamic energy per addition for all of these is O(n).
- "O" notation hides the constants. Watch out for this!
- The "real" cost of the carry-select is at least 2X the "real" cost of the ripple. "Real" cost of the CLA is probably at least 2X the "real" cost of the carry-select.
- The actual multiplicative constants depend on the implementation details and technology.
- FPGA and ASIC synthesis tools will try to choose the best adder architecture automatically - assuming you specify addition using the "+" operator, as in "assign A = B + C"