EE 457 Unit 2b

Fast Adders
(Carry-Lookahead Adder)

Carry-Lookahead Adders

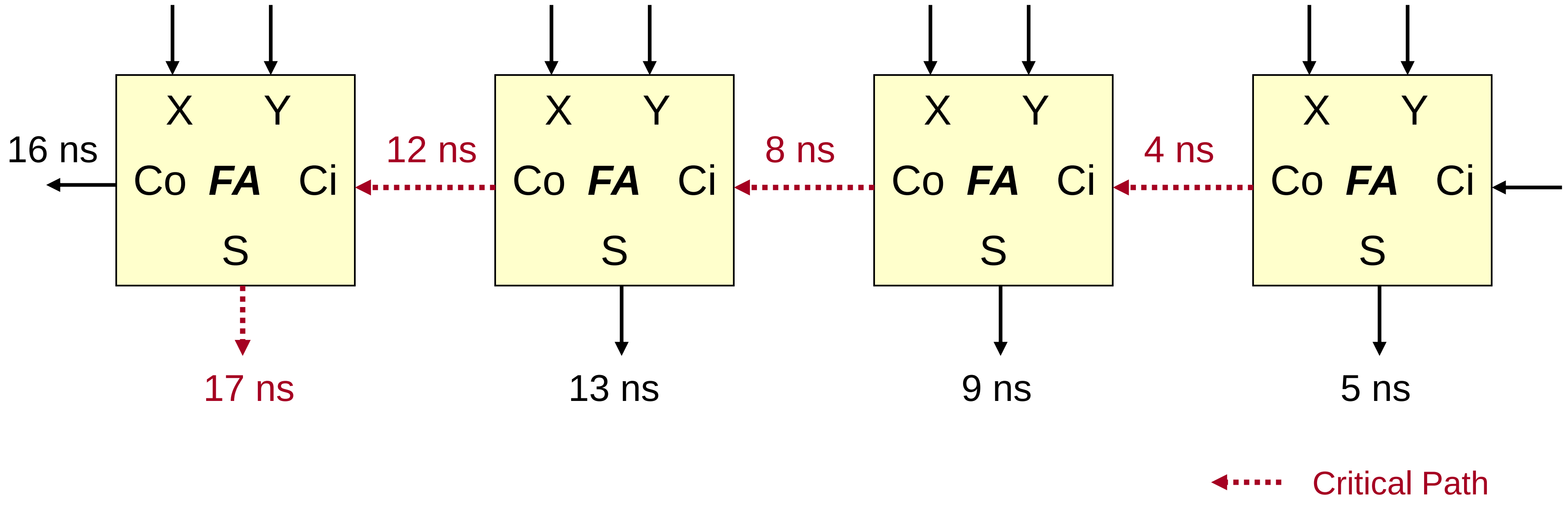
FAST ADDERS

Ripple Carry Adder Critical Path

- Critical Path = Longest possible delay path

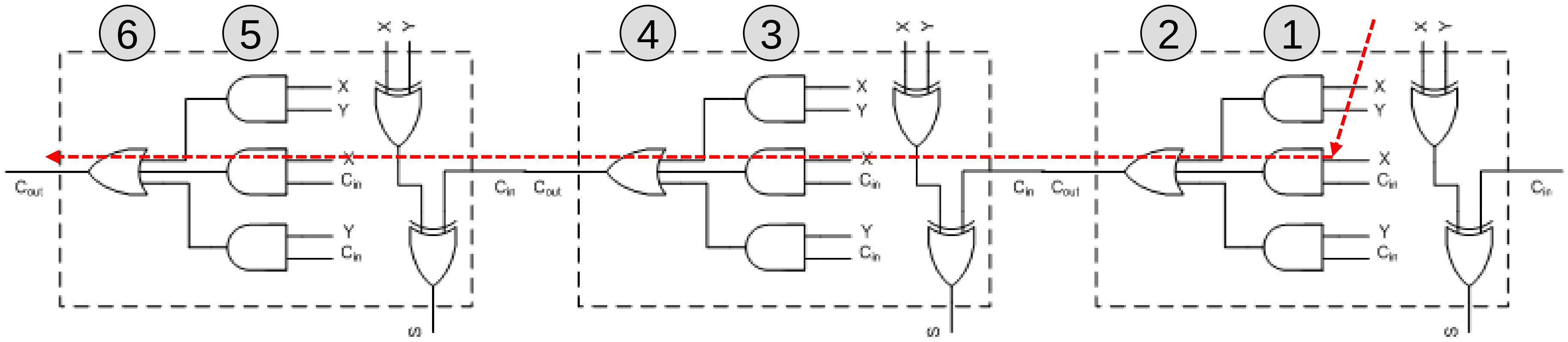
Assume t_{sum} = 5 ns,

t_{carry}= 4 ns



Ripple Carry Adders

- Ripple-carry adders (RCA) are slow due to carry propagation
  - At least 2 levels of logic per full adder
  - Total delay for n-bit adder = n * T_{fa}



Fast Adders

- Recall that any logic function can be implemented as a 2-level implementation
  - SOP (AND-OR / NAND-NAND) implementation
  - POS (OR-AND / NOR-NOR) implementation
- Rather than waiting for the previous carry, [$C_{i+1} = f(X_i,Y_i,C_i)$] can we compute the carry as a function of just the inputs
  - $C_{i+1} = f(X_i,X_{i-1},\ldots X_0,Y_i,Y_{i-1},\ldots Y_0)$
  - This requires gates with many inputs which is infeasible in modern technologies above 4 or 5 inputs
  - But, we can try to use this idea of generating multiple carries at once by looking at many inputs

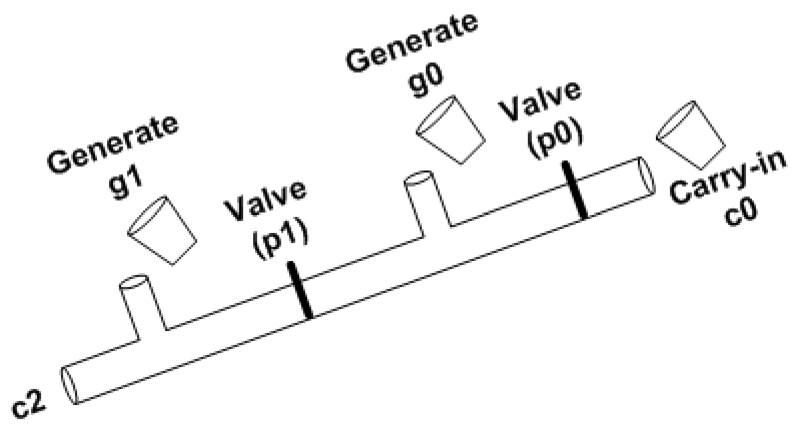
Fast Adders

- To produce multiple carries in parallel, let us define some new signals for each column of addition that indicate information about the carry-out regardless of carry-in:
  - g_i = Generate: This column will generate a carry-out whether or not the carry-in is ‘1’
    g_i is true when A_i and B_i is 1 => $g_i = A_i \bullet B_i$
  - p_i = Propagate: This column will propagate a carry-in (if there is one) to the carry-out.
    p_i is true when A_i or B_i is 1 => $p_i = A_i + B_i$
- Using these signals, we can define the carry-out (c_{i+1}) as:

$$c_{i+1} = g_i + p_i c_i$$

Carry Lookahead Analogy

- Consider the carry-chain like a long tube broken into segments. Each segment is controlled by a valve (propagate signal) and can insert a fluid into that segment (generate signal)
- The carry-out of the diagram below will be true if g1 is true or p1 is true and g0 is true, or p1, p0 and c1 is true



Carry Lookahead Logic

- Define each carry in terms of p_i, g_i and the initial carry-in (c_0) and not in terms of carry chain (intermediate carries: c1,c2,c3,…)
- $c1 = g_0 + p_0c_0$
- $c2 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0c_0$
- c3 = …
- c4 = …

4-Bit CLA

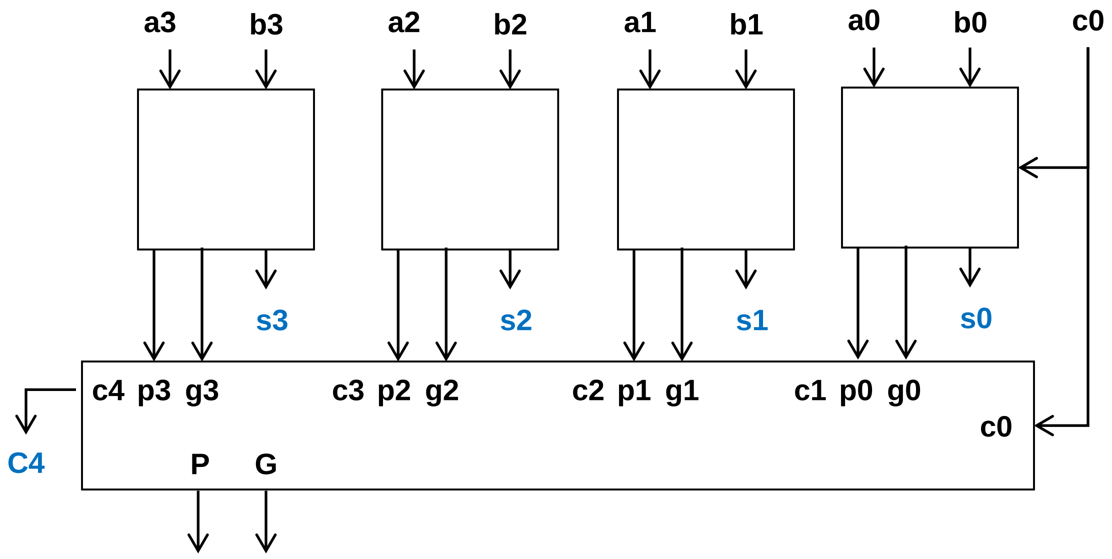
- At this point we should probably stop as we have a 5-input gate in our equation
- Let’s take our logic and build a 4-bit carry lookahead adder (CLA)

Delay to produce s2

- Delay for pi,gi = 1
- Delay to produce c2 = 2
- Delay to produce s2 = 2

= 5 gates

(Compare to 8 gate delays for RCA)

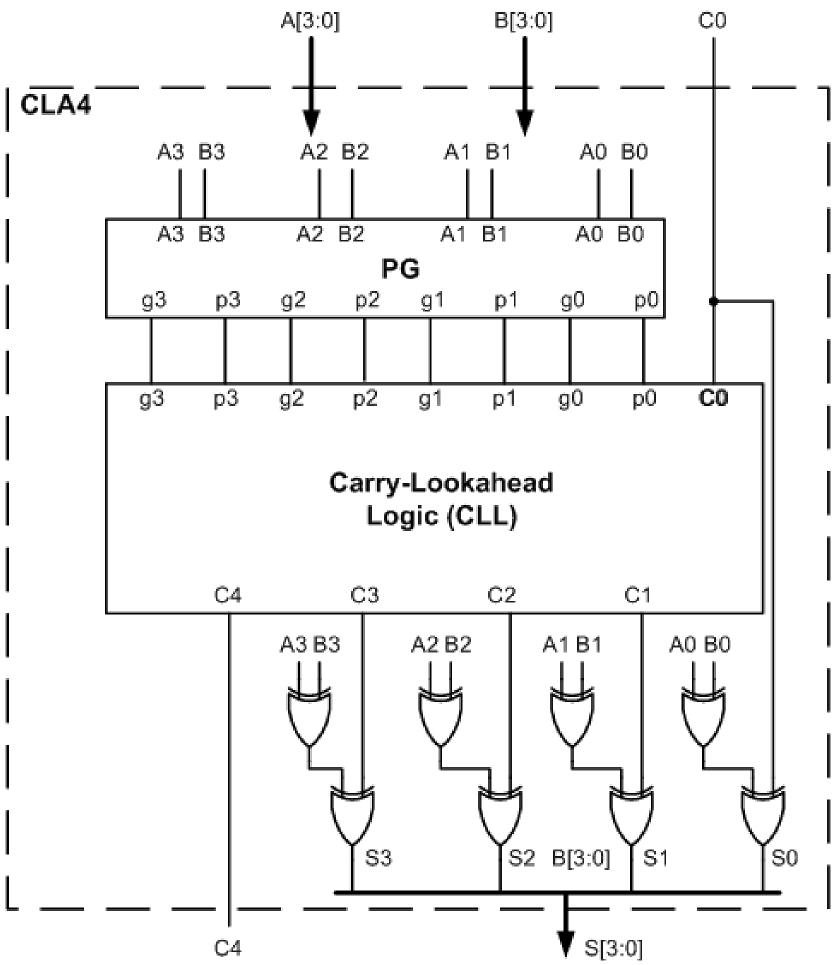


Is S3 produced later than S2?

Is C3 the last signal produced?

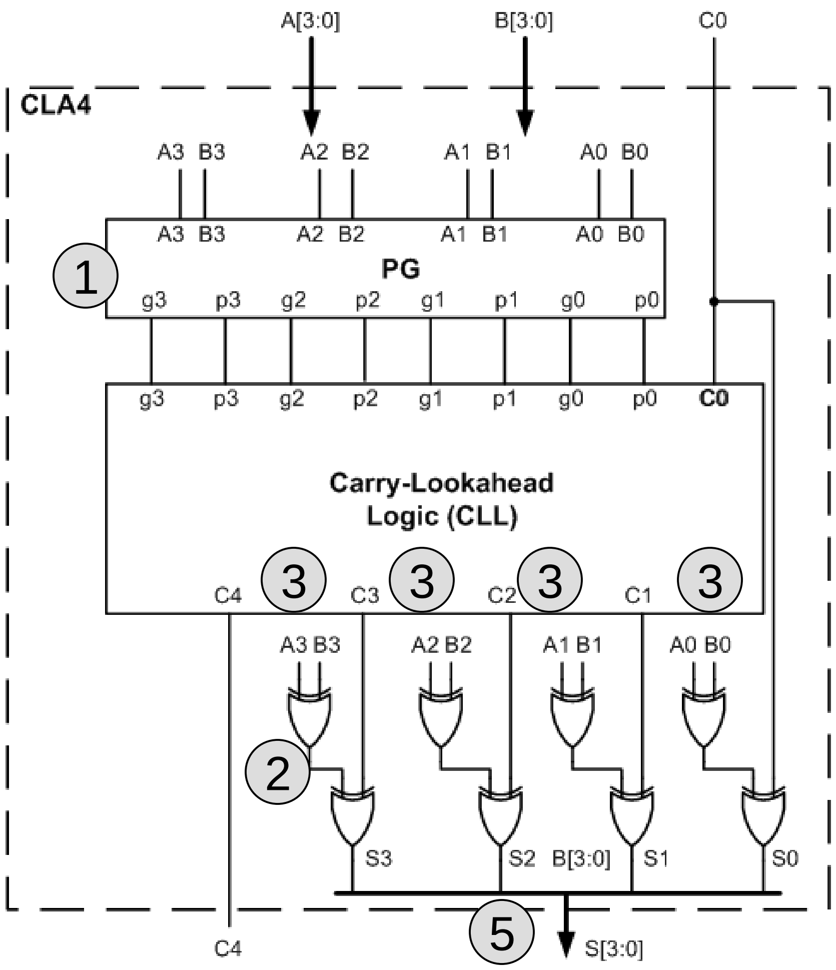
Carry Lookahead Adder

- Use carry-lookahead logic to generate all the carries in one shot and then create the sum
- Example 4-bit CLA shown below



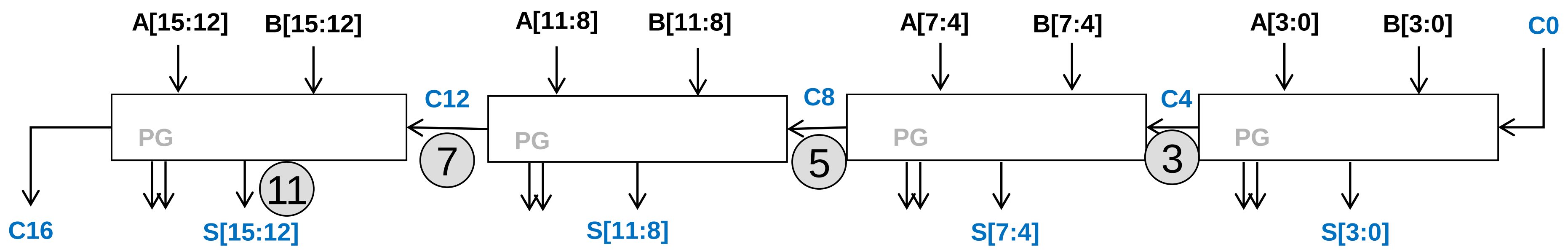
Carry Lookahead Adder

- Use carry-lookahead logic to generate all the carries in one shot and then create the sum
- Example 4-bit CLA shown below



16-Bit CLA

- At this point we should probably stop as we have a 5-input gate in our equation



16-bit RCA Delay = 16*2 = 32 gate delays

Delay of the above adder design = 3+2+2+4 = 11 gates

Let us improve by looking ahead at a higher level to produce C16, C12, C8, C4 in parallel

What's the difference between the equation for G here and C4 on the previous slides

Define P and G as the overall Propagate and Generate signals for a set of 4 bits

$P = p3 \bullet p2 \bullet p1 \bullet p0$

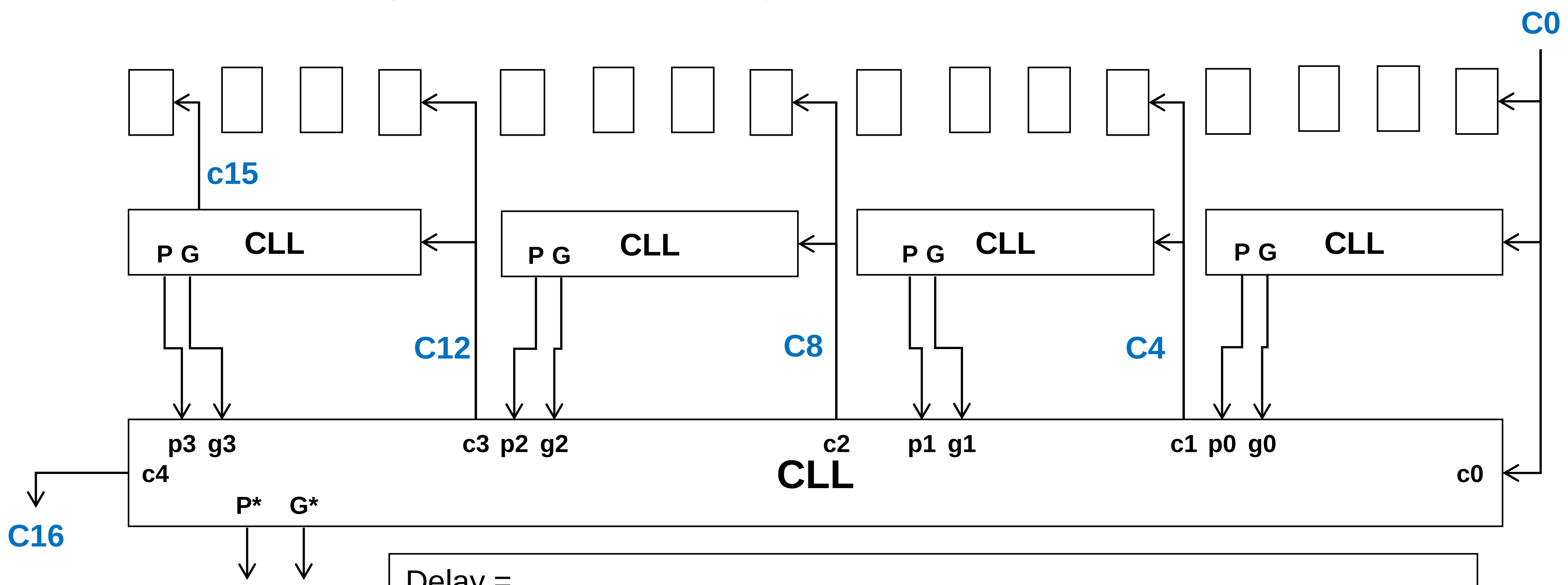
$G = g3 + p3 \bullet g2 + p3 \bullet p2 \bullet g1 + p3 \bullet p2 \bullet p1 \bullet g0$

16-bit CLA Closer Look

- Each 4-bit CLA only propagates its overall carry-in if each of the 4 columns propagates:
  - P0 = p3• p2 •p1 •p0
  - P1 = p7• p6 •p5 •p4
  - P2 = p11• p10 •p9 •p8
  - P3 = p15• p14 •p13 •p12
- Each 4-bit CLA generates a carry if any column generates and the more significant columns propagate
  - G0 = g3 + (p3 •g2) + (p3 •p2 •g1)+(p3 •p2 •p1 •g0)
  - …
  - G3 = g15 + (p15 •g14) + (p15 •p14 •g13)+(p15 •p14 •p13 •g12)
- The higher order CLL logic (producing C4,C8,C12,C16) then is realized as:
  - (C4) =>C1 = G0 + (P0 •c0)
  - …
  - (C16) => C4 = G3 + (P3 •G2) + (P3 •P2 •G1) +(P3 • P2 • P1 • G0)+ (P3 •P2 •P1 •P0 •c0)
- These equations are exactly the same CLL logic we derived earlier

16-Bit CLA

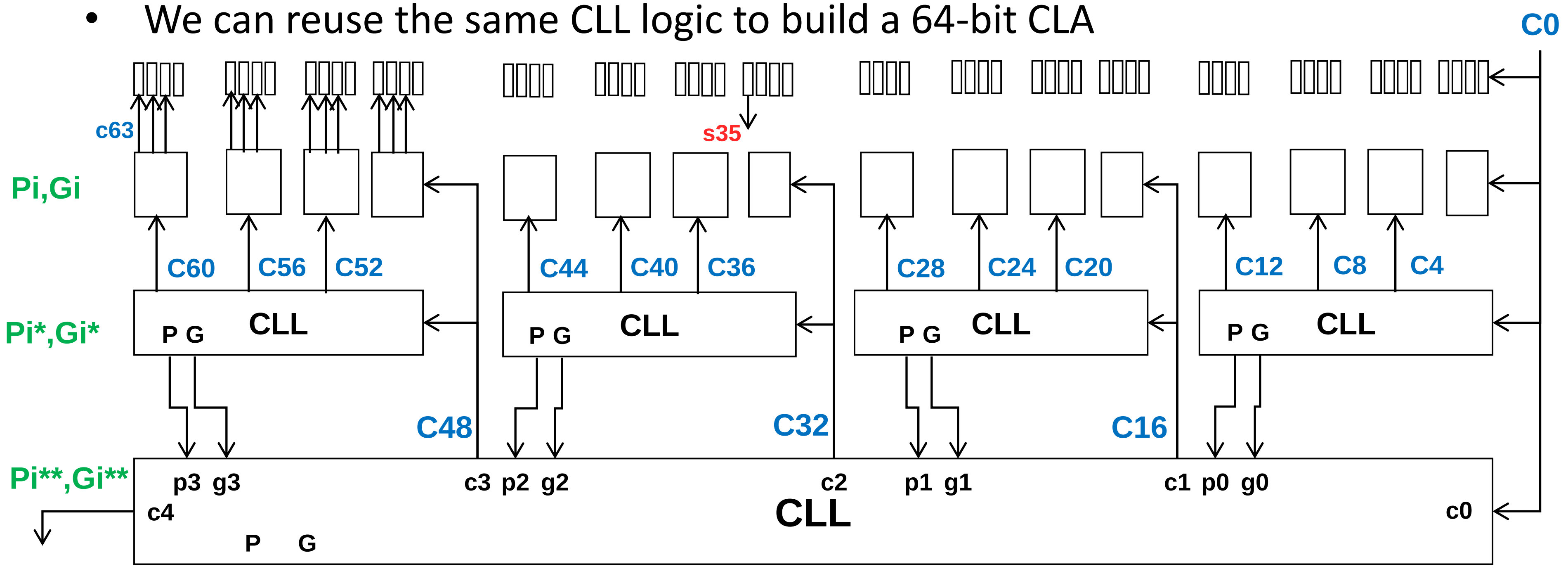
- Understanding 16-bit CLA hierarchy…



Delay =
= 3 = Delay in producing Pi,Gi
= 5 = Delay in producing Pi*,Gi*
= 5 = Delay in producing C4,C8,C12,C16
= 7 = Delay in producing c15
= 9 = Delay in producing S15

64-Bit CLA

- We can reuse the same CLL logic to build a 64-bit CLA



= 13 = Delay in producing S63

Is the delay in producing s63 the same as in s35?

= 5 = Delay in producing S2

= 4 = Delay in producing S0

= 3 = Delay in producing Pi,Gi

= 5 = Delay in producing Pj*,Gj*

= 7 = Delay in producing C48

= 9 = Delay in producing C60

= 11 = Delay in producing C63

= 13 = Delay in producing S63

= 13 Total Delay

Extrapolating CLA Logic Levels

- In the above designs we've assumed 5-input AND and OR gates are reasonable allowing us to group in blocks of 4
  - Define b = blocking factor = number of carries produced in parallel
- The greater the blocking factor the smaller the depth of logic (and vice-versa)
- This leads us to reason that the delay of a CLA is $O(\log_b n)$
- If we could only use 3-input gates we'd need a blocking factor of 2

Blocking factor of 2

- Each A box generates
  - $p_i = a_i + b_i$
  - $g_i = a_i \bullet b_i$
  - $s_i = a_i \oplus b_i$
- Each B box generates
  - $P_i = p_i \bullet p_{i-1}$
  - $G_i = g_i + p_i \bullet g_{i-1}$
  - $c_{i+1} = G_i + (P_i \bullet c_i)$

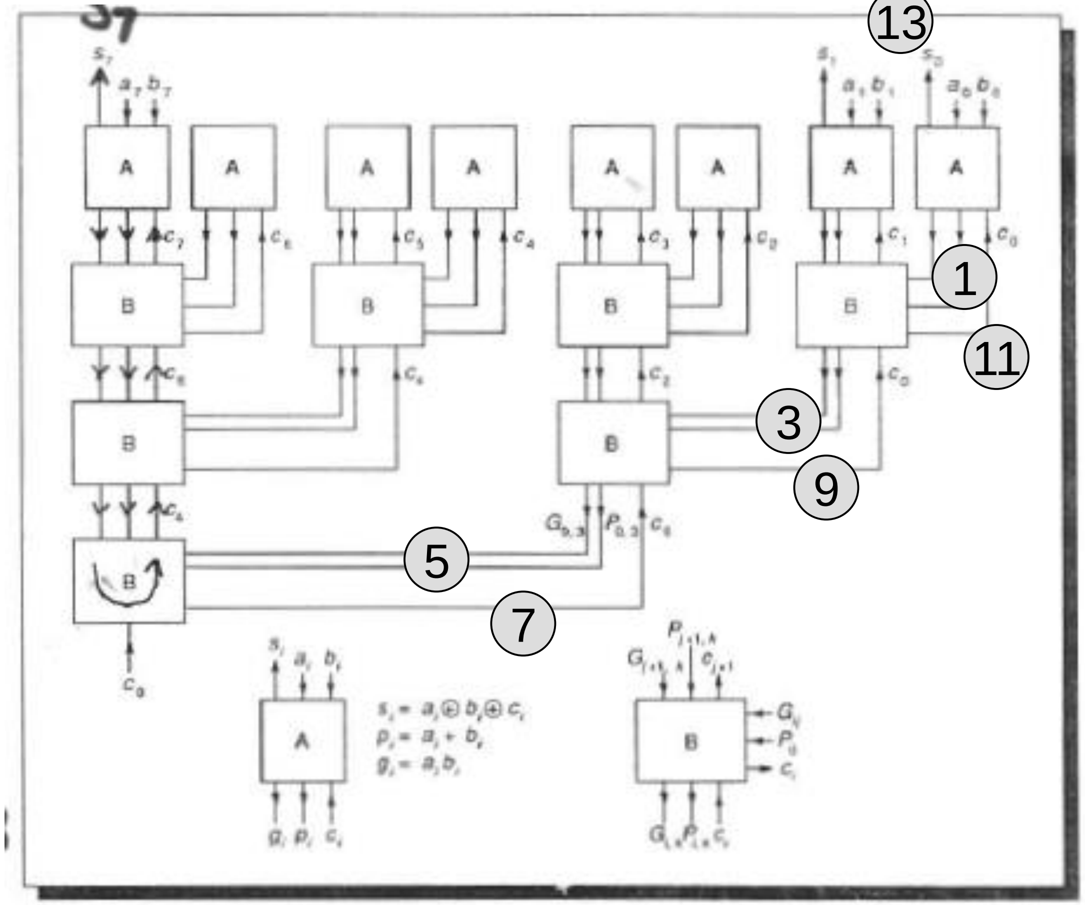


FIGURE A.13 Complete carry-lookahead tree adder. This is the combination of Figures A.11 and A.12. The numbers to be added enter at the top, flow to the bottom to combine with c_0, and then flow back up to compute the sum bits.

Credits

- These slides were derived from Gandhi Puvvada's EE 457 Class Notes