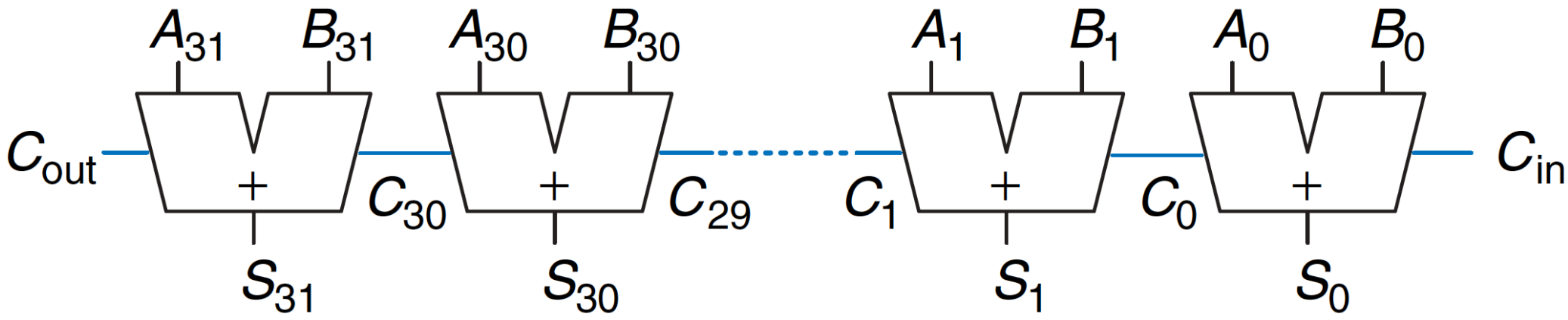
Carry-Lookahead Adder

Ripple Carry Adder



Carry-Lookahead Adder

Waiting for propagation of carry is expensive all the way through adder is expensive

Saw with carry-select adders that we can do work in parallel to decrease total time at cost of additional hardware

Carry-select adders duplicate everything (twice as many full adders as ripple-carry adder)

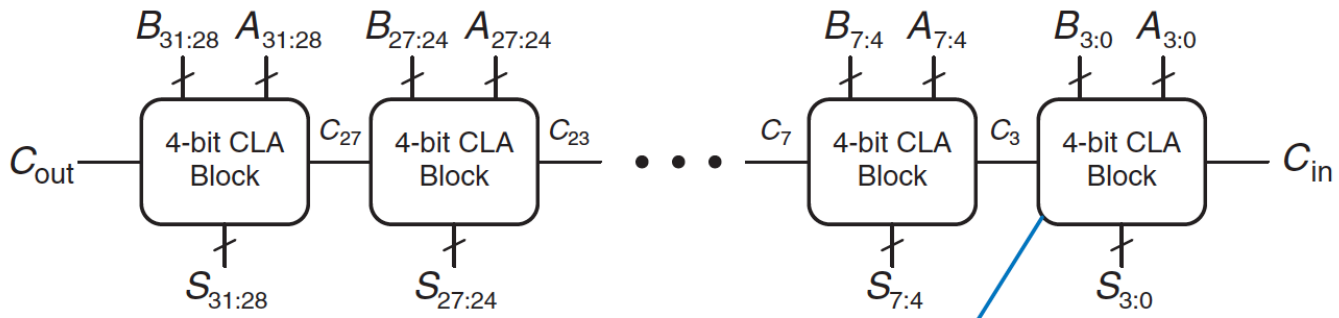
In reality, computing the carry is the slowdown, so we can save some hardware by focusing on precomputing just the carry

Carry-Lookahead Adder

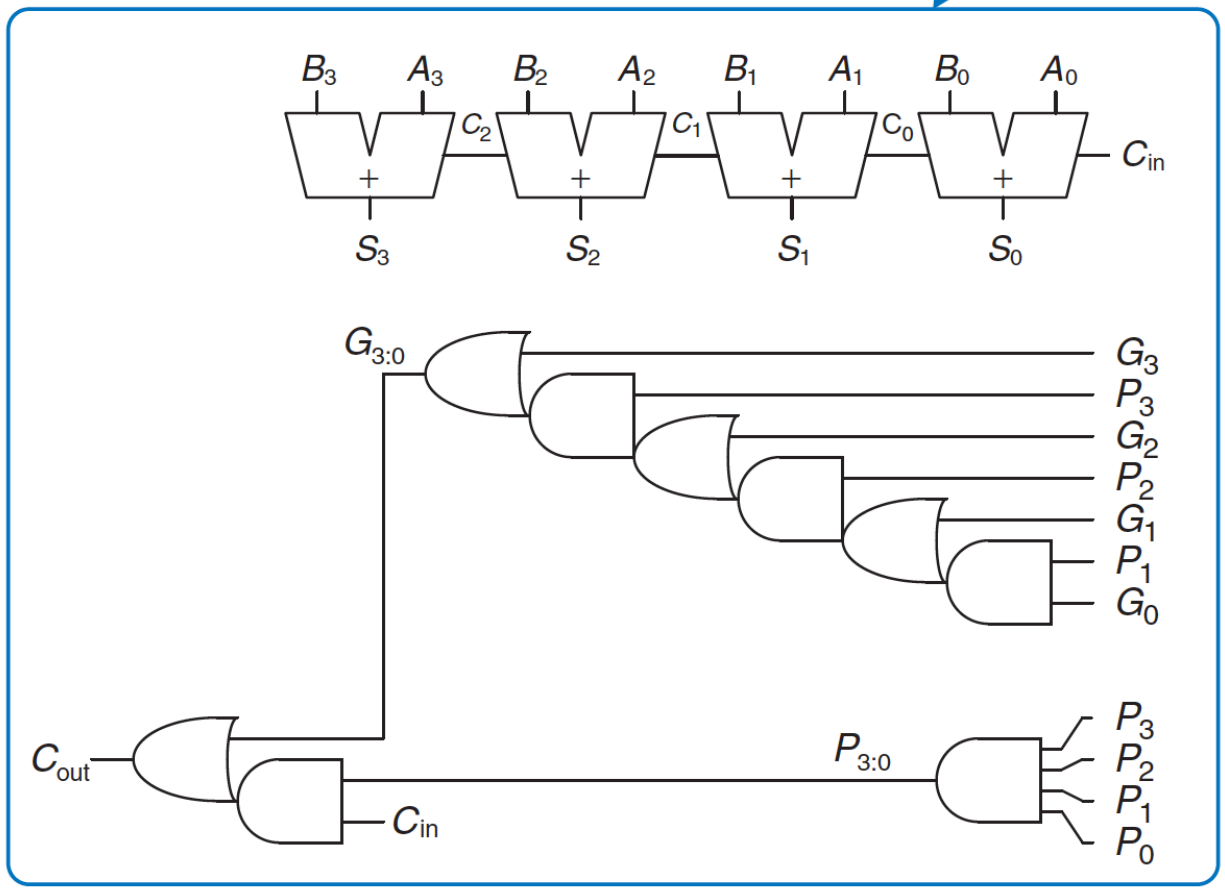
Use same general trick as carry-select – do most of the work in parallel, before carry-in arrives

Big picture:

- Divide adder into blocks
- Include separate circuitry to calculate the carry out from each block
- Most of work for carry-out is precomputed in all blocks at once


$B_{31:28}$ $A_{31:28}$
$B_{27:24}$ $A_{27:24}$
$B_{7:4}$ $A_{7:4}$
$B_{3:0}$ $A_{3:0}$
C_{out}
4-bit CLA Block
C_{27}
C_{23}
C_7
C_3
C_{in}
$S_{31:28}$
$S_{27:24}$
$S_{7:4}$
$S_{3:0}$

(a)


B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_2 C_1 C_0 C_{in}
S_3 S_2 S_1 S_0
$G_{3:0}$
G_3 P_3 G_2 P_2 G_1 P_1 G_0
C_{out} C_{in}
$P_{3:0}$
P_3 P_2 P_1 P_0

(b)

Computing Carry-out

Simple enough to determine whether single column of sum results in carry-out

Pay attention to “generate” vs “propagate” terminology

$$\begin{array}{r} 0101_2 \\ +\ 0111_2 \\ \hline \end{array}$$

Computing Carry-out

i_{th} column *generates* carry if it produces carry out independent of carry in

A and B are 1

$G_i = A_iB_i$

i_{th} column *propagates* carry if there is carry out whenever there is carry in

A or B are 1

$P_i = A_i + B_i$

Computing Carry-out

For individual column of sum:

$$C_i = A_iB_i + (A_i + B_i)C_{i-1} = G_i + P_iC_{i-1}$$

$$
\begin{array}{r}
0101\ 0100\ 1001_2 \\
+\ 0111\ 1101\ 0110_2 \\
\hline
\end{array}
$$

Extends to Blocks

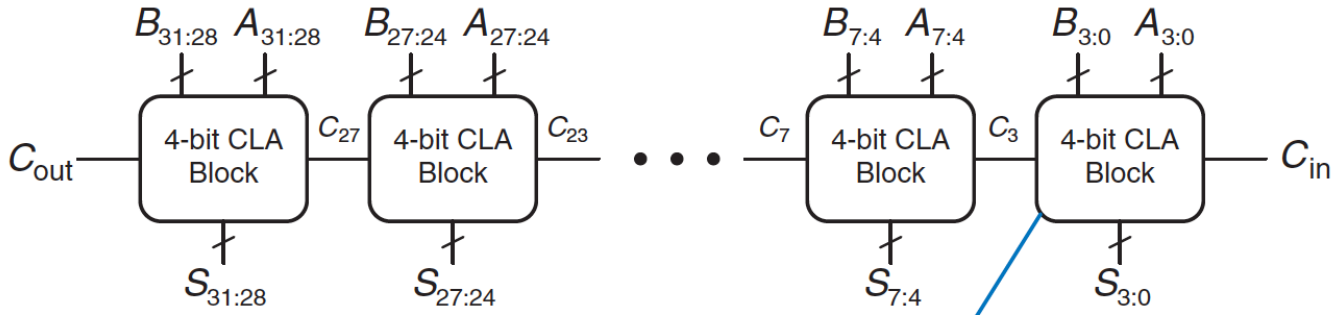
Block *generates* carry if it produces a carry out independent of carry in

Block *propagates* carry if it produces a carry out when there is a carry in

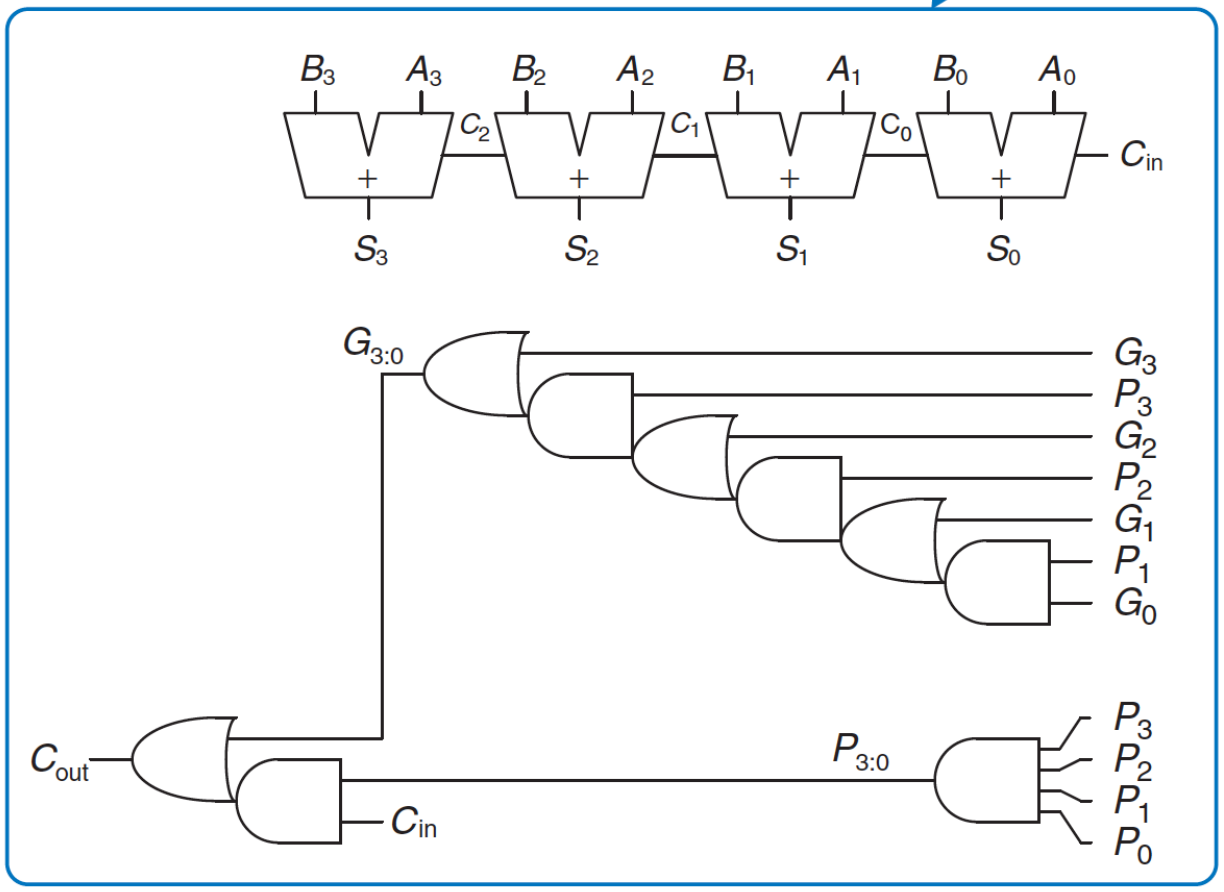
$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$$

$$P_{3:0} = P_3P_2P_1P_0$$

$$C_i = G_{i:j} + P_{i:j}C_j$$

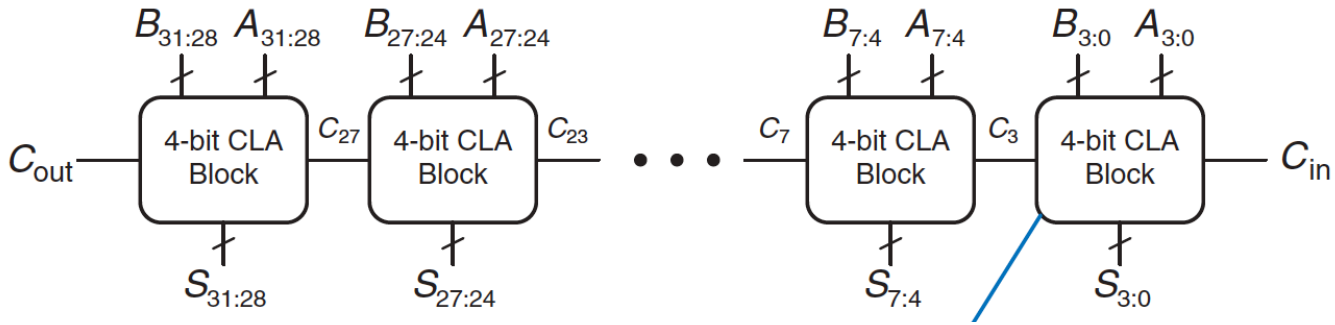


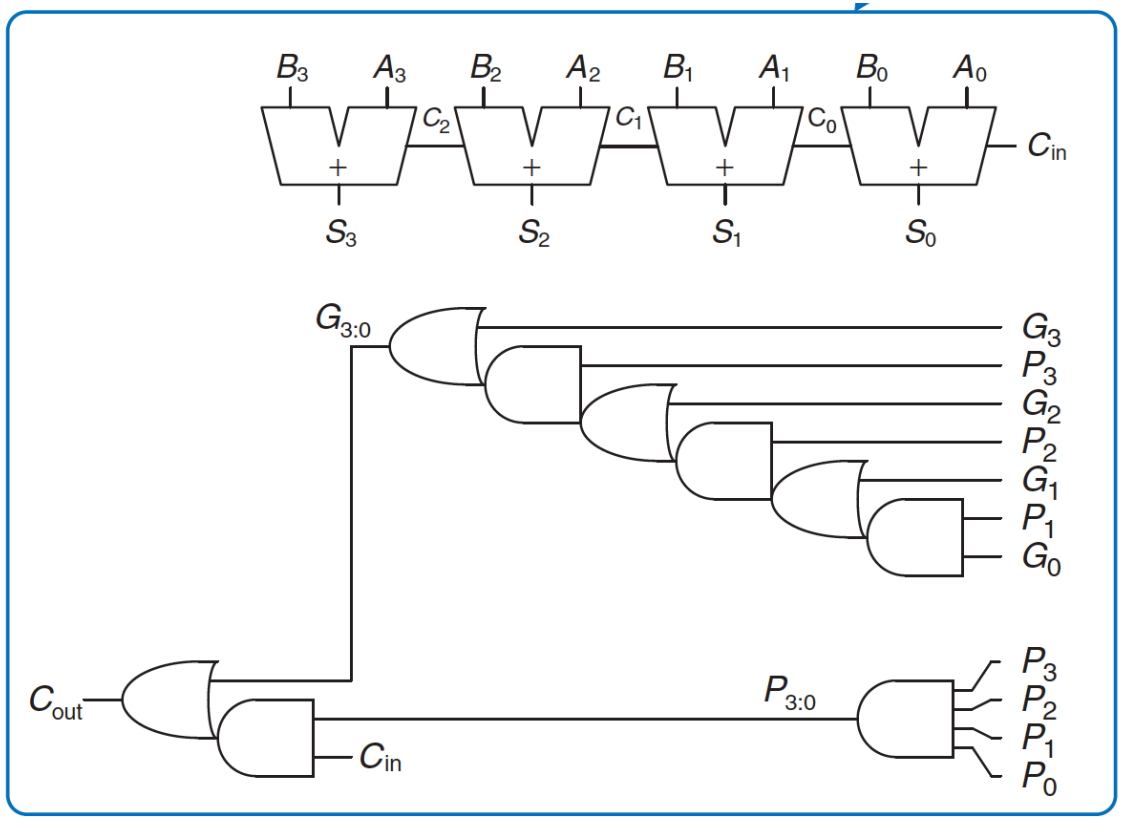
(a)



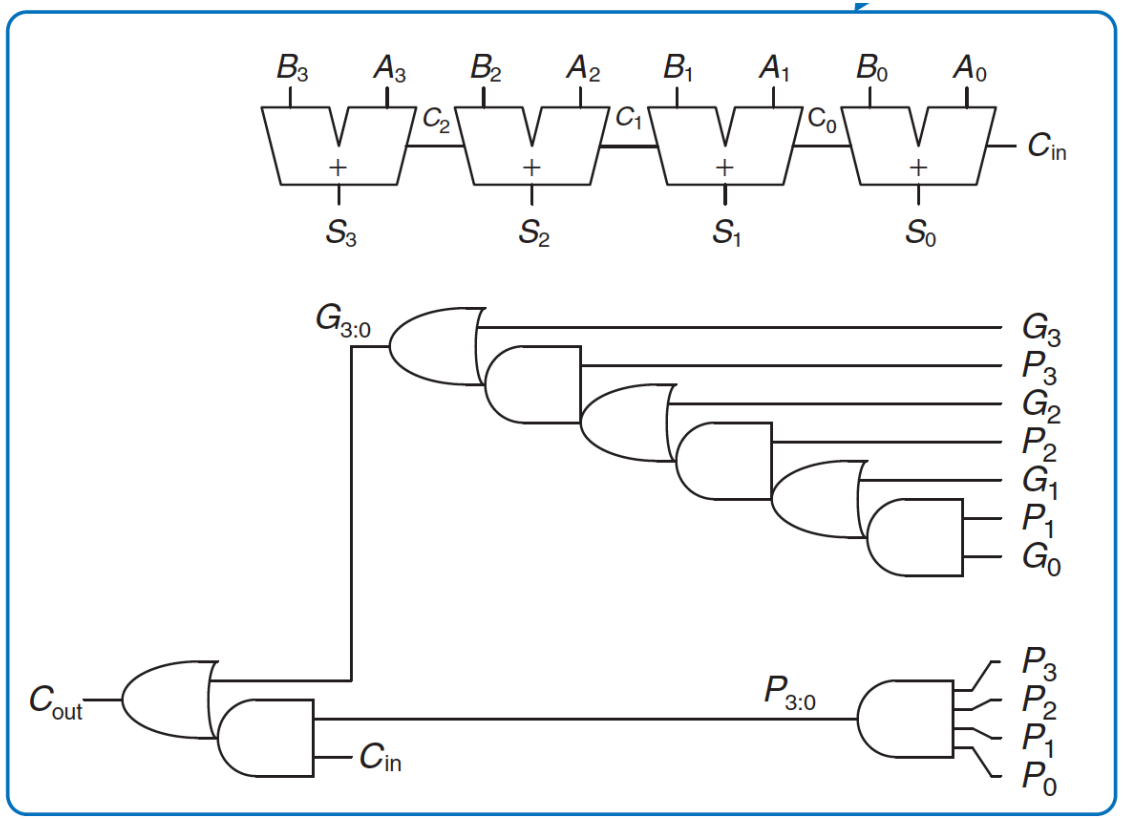
(b)

$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$$
$$P_{3:0} = P_3P_2P_1P_0$$
$$C_i = G_{i:j} + P_{i:j}C_j$$

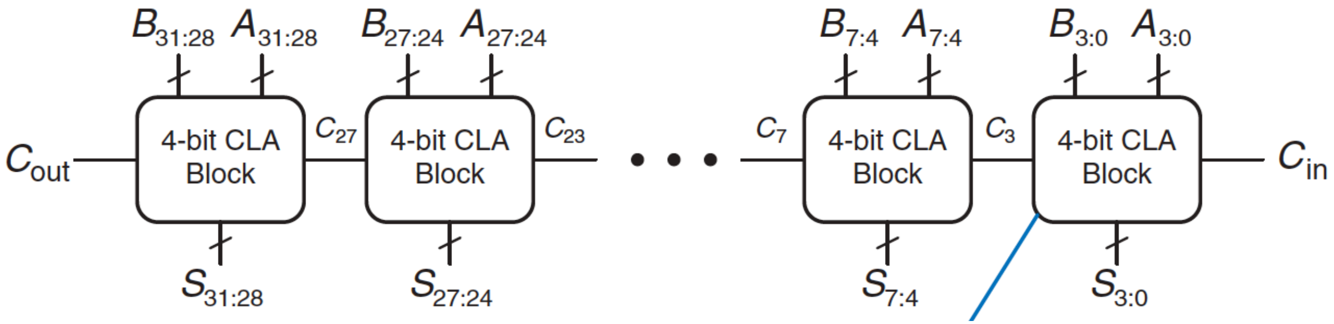

$B_{31:28}$ $A_{31:28}$ $B_{27:24}$ $A_{27:24}$
$B_{7:4}$ $A_{7:4}$ $B_{3:0}$ $A_{3:0}$
C_{out}
4-bit CLA Block
C_{27}
C_{23}
C_7
C_3
C_{in}
$S_{31:28}$ $S_{27:24}$ $S_{7:4}$ $S_{3:0}$


B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_2 C_1 C_0 C_{in}
S_3 S_2 S_1 S_0
$G_{3:0}$
G_3 P_3 G_2 P_2 G_1 P_1 G_0
C_{out} C_{in}
$P_{3:0}$
P_3 P_2 P_1 P_0

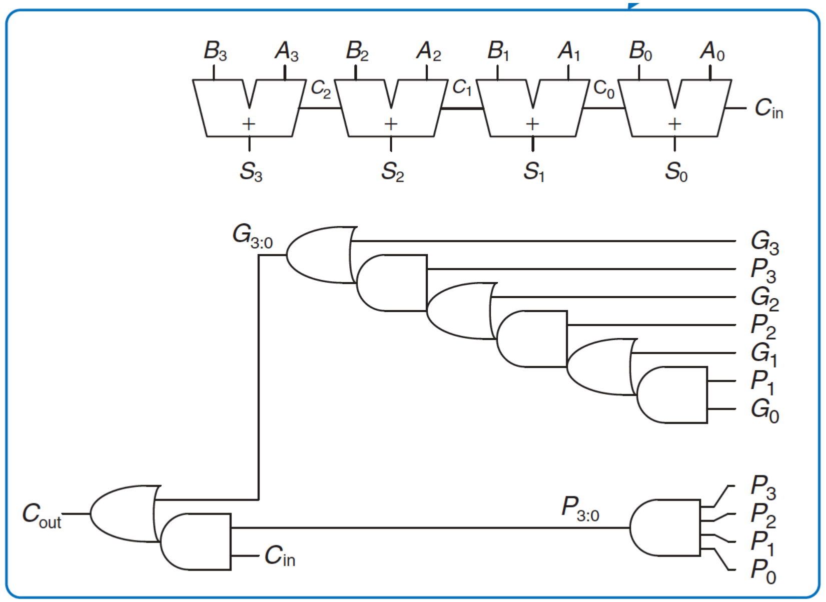
(b)


B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
C_2 C_1 C_0 C_{in}
S_3 S_2 S_1 S_0
$G_{3:0}$
G_3 P_3 G_2 P_2 G_1 P_1 G_0
C_{out} C_{in}
$P_{3:0}$
P_3 P_2 P_1 P_0

(b)

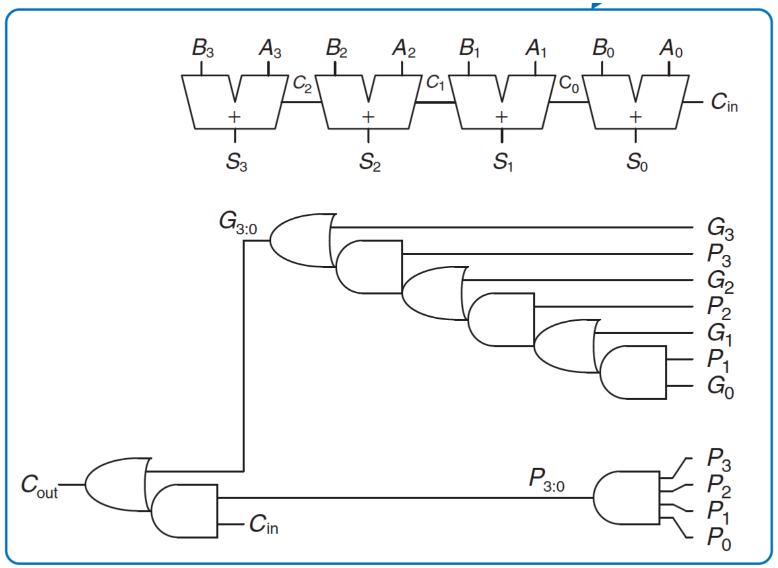


B31:28 A31:28
B27:24 A27:24
B7:4 A7:4
B3:0 A3:0
Cout
4-bit CLA Block
C27
4-bit CLA Block
C23
C7
4-bit CLA Block
C3
4-bit CLA Block
Cin
S31:28
S27:24
S7:4
S3:0



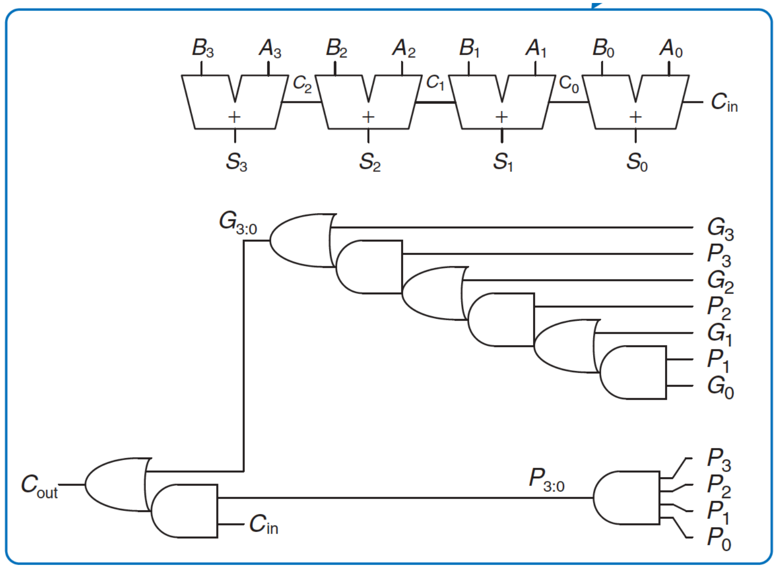
B3 A3 B2 A2 B1 A1 B0 A0
C2
C1
C0
Cin
S3 S2 S1 S0
G3:0
G3 P3 G2 P2 G1 P1 G0
Cout
Cin
P3:0
P3 P2 P1 P0

(b)



B3 A3 B2 A2 B1 A1 B0 A0
C2
C1
C0
Cin
S3 S2 S1 S0
G3:0
G3 P3 G2 P2 G1 P1 G0
Cout
Cin
P3:0
P3 P2 P1 P0

(b)



B3 A3 B2 A2 B1 A1 B0 A0
C2
C1
C0
Cin
S3 S2 S1 S0
G3:0
G3 P3 G2 P2 G1 P1 G0
Cout
Cin
P3:0
P3 P2 P1 P0

(b)

Steps to CLA

- Step 1:
  - First block starts computing S0 through S3 using ripple-carry adder
  - **All blocks** compute Gi:j and Pi:j
- Step 2:
  - First block finishes computing C3 using just two gates and precomputed G3:0 and P3:0
- Step 3:
  - Second block receives C3
  - Second block computes C7 using just two gates and precomputed G7:4 and P7:4
  - Second block starts computing S4 through S7 using ripple-carry adder
  - While S4 through S7 are being computed, C3 is already passed to next block
  - … and so on