Arithmetic Circuits 3

COE 202

Digital Logic Design

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

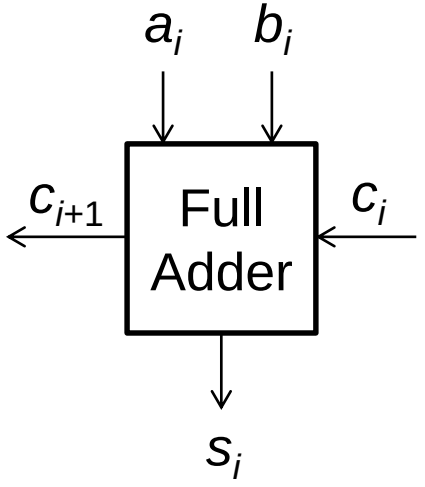
Presentation Outline

- Carry Lookahead Adder
- BCD Adder
- Binary Multiplier
- Carry-Save Adders in Multipliers

Carry Lookahead Adder

- Is it possible to eliminate carry propagation?
- Observation: $c_{i+1} = a_i\, b_i + (a_i \oplus b_i)\, c_i$
- If both inputs a_i and b_i are 1s then

  c_{i+1} will be 1 regardless of input c_i
- Therefore, define $g_i = a_i\, b_i$



- g_i is called **carry generate**: generates c_{i+1} regardless of c_i
- In addition, define $p_i = (a_i \oplus b_i)$ a_i or b_i is 1, not both
- p_i is called **carry propagate**: propagates value of c_i to c_{i+1}
- Equation of output carry becomes: $c_{i+1} = g_i + p_i\, c_i$
- If both inputs a_i and b_i are 0s then $g_i = p_i = 0$ and $c_{i+1} = 0$

Carry Bits

Carry bits are generated by a **Lookahead Carry Unit** as follows:

$c_0 =$ input carry

$c_1 = g_0 + p_0\, c_0$

$c_2 = g_1 + p_1\, c_1 = g_1 + p_1(g_0 + p_0 c_0) = g_1 + p_1\, g_0 + p_1\, p_0\, c_0$

$c_3 = g_2 + p_2\, c_2 = g_2 + p_2\, g_1 + p_2\, p_1\, g_0 + p_2\, p_1\, p_0\, c_0$

$c_4 = g_3 + p_3\, c_3 = g_3 + p_3\, g_2 + p_3\, p_2\, g_1 + p_3\, p_2\, p_1\, g_0 + p_3\, p_2\, p_1\, p_0\, c_0$

Define **Group Generate**: $GG = g_3 + p_3\, g_2 + p_3\, p_2\, g_1 + p_3\, p_2\, p_1\, g_0$

Define **Group Propagate**: $GP = p_3\, p_2\, p_1\, p_0$

$c_4 = GG + GP\, c_0$

Carry does not ripple anymore

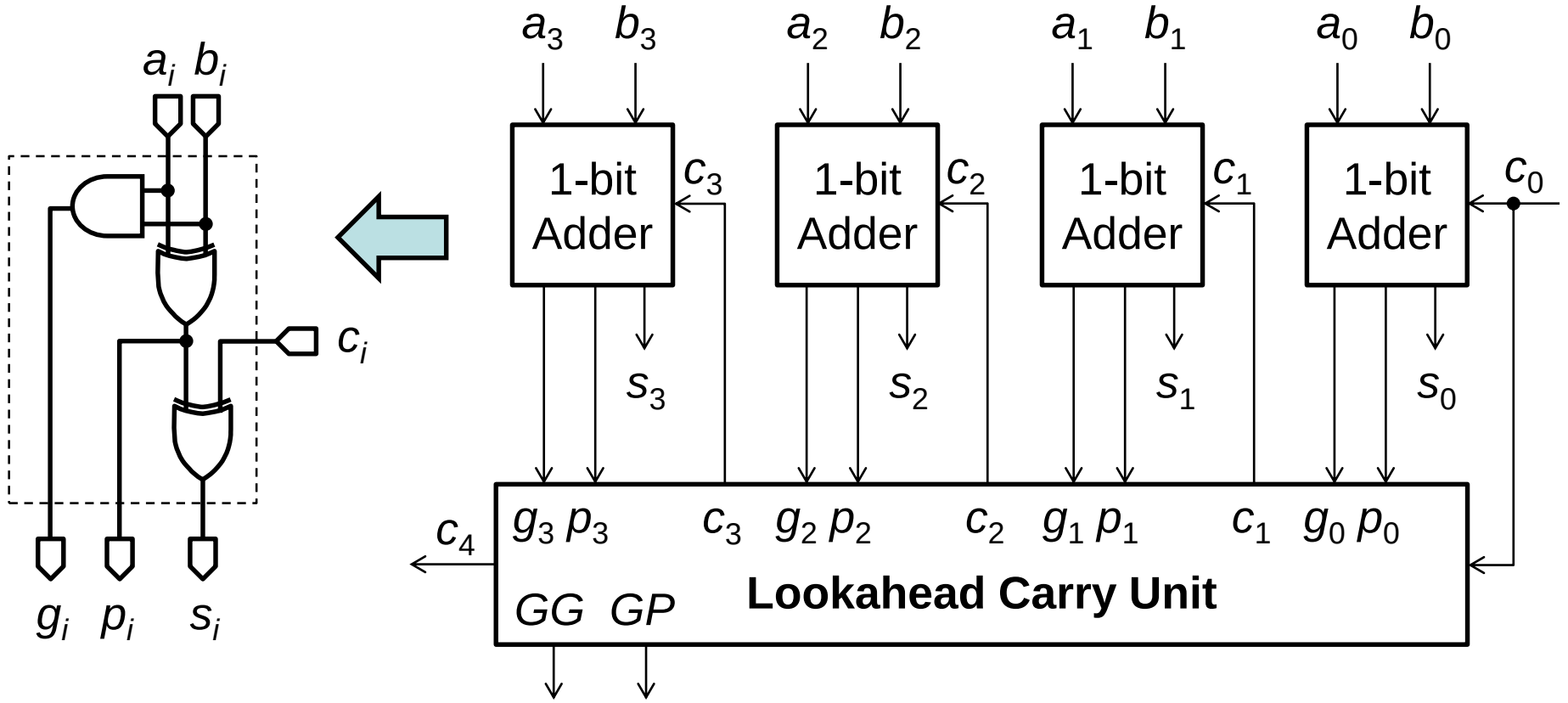
Reduced delay when generating c_1 to c_4 in parallel

4-Bit Carry Lookahead Adder

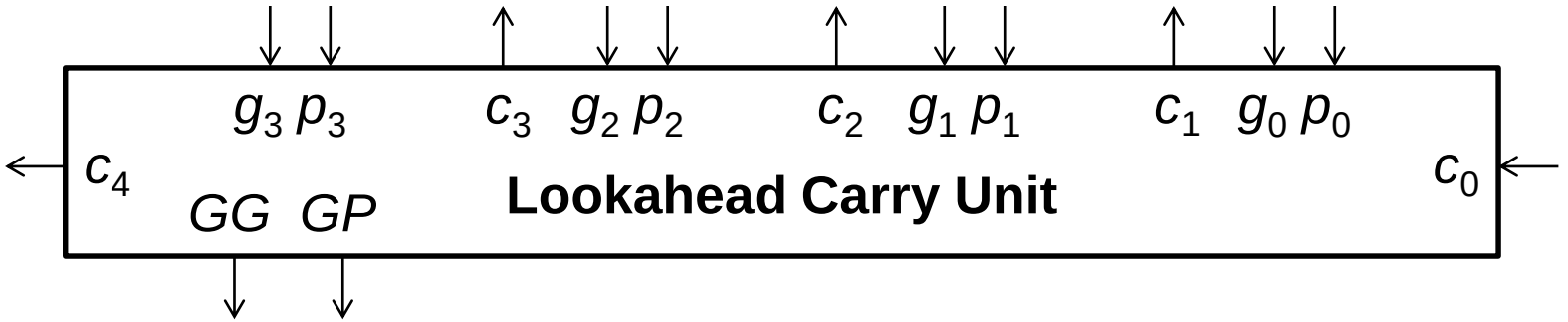
All **generate** and **propagate** signals (g_i, p_i) are generated in parallel

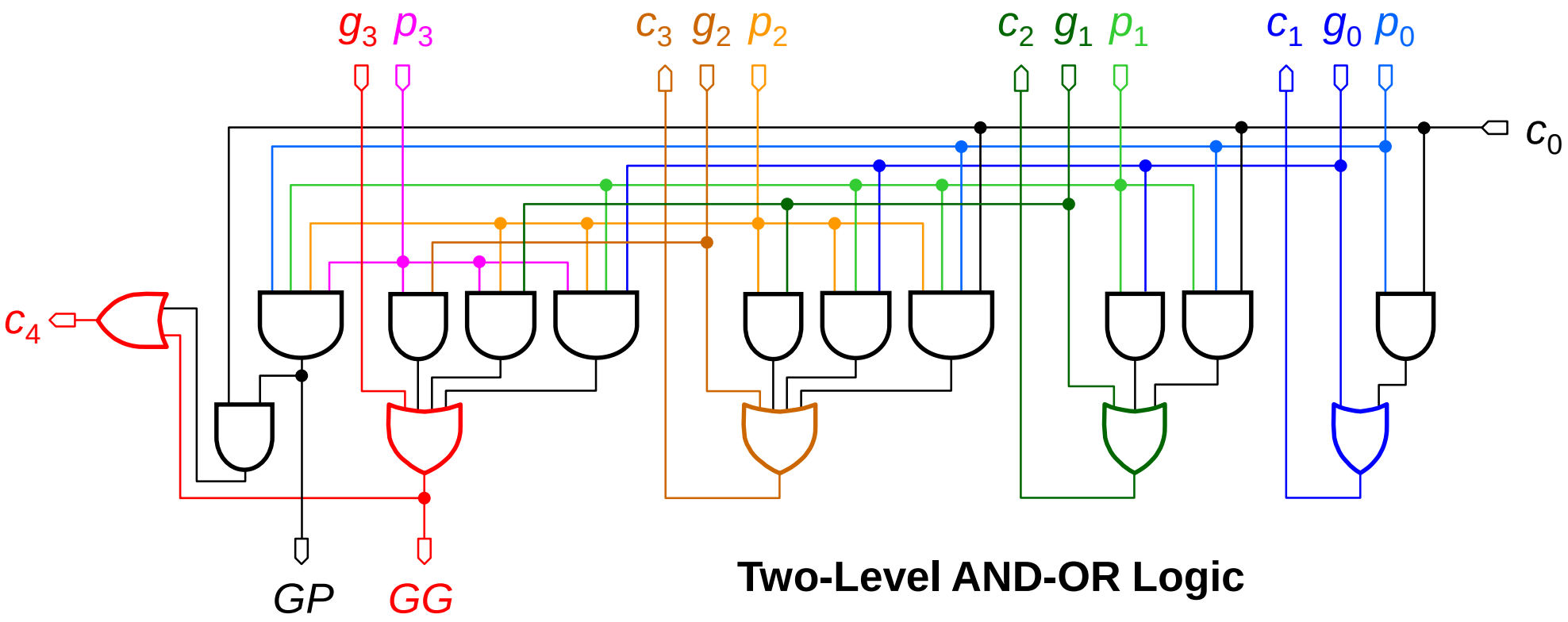
All carry bits (c_1 to c_4) are generated in parallel

The sum bits are generated faster than ripple-carry adder



Lookahead Carry Unit

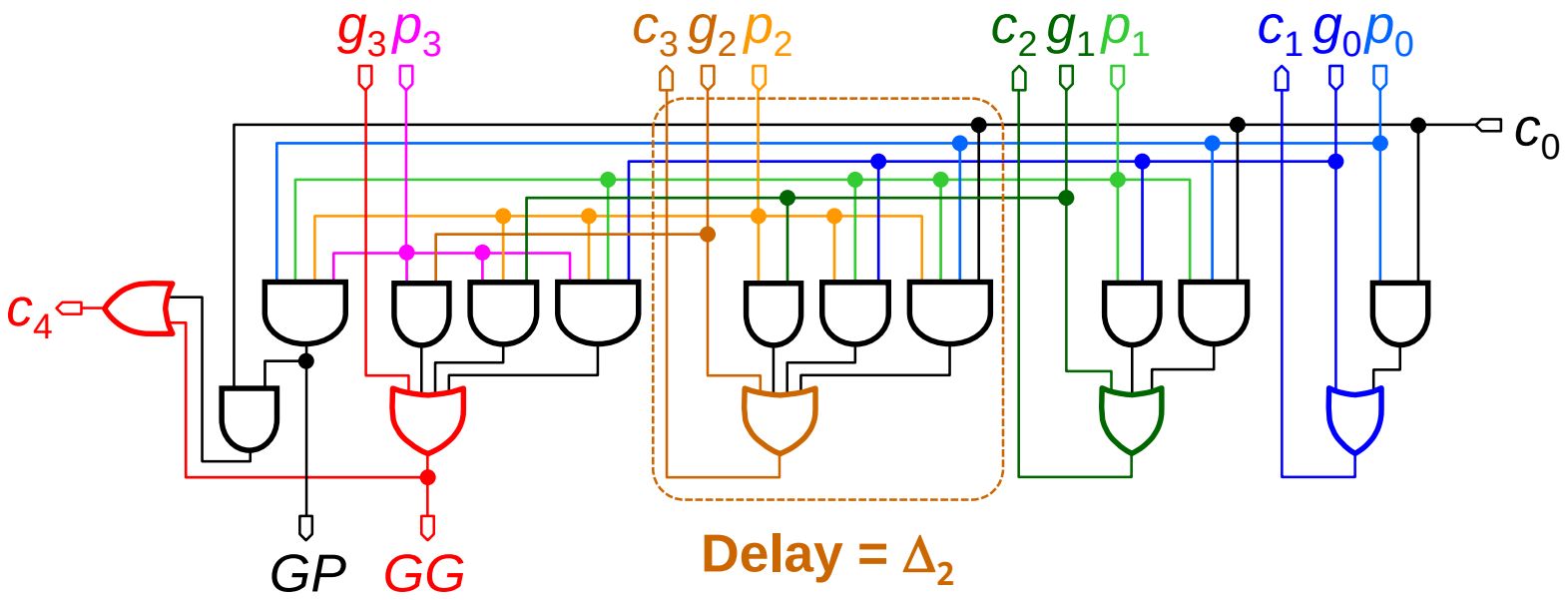


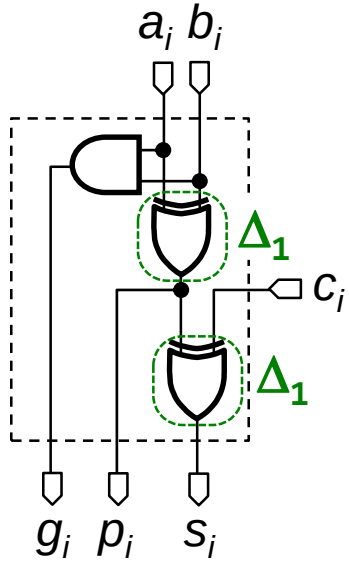


Two-Level AND-OR Logic

Longest Delay of the 4-bit CLA

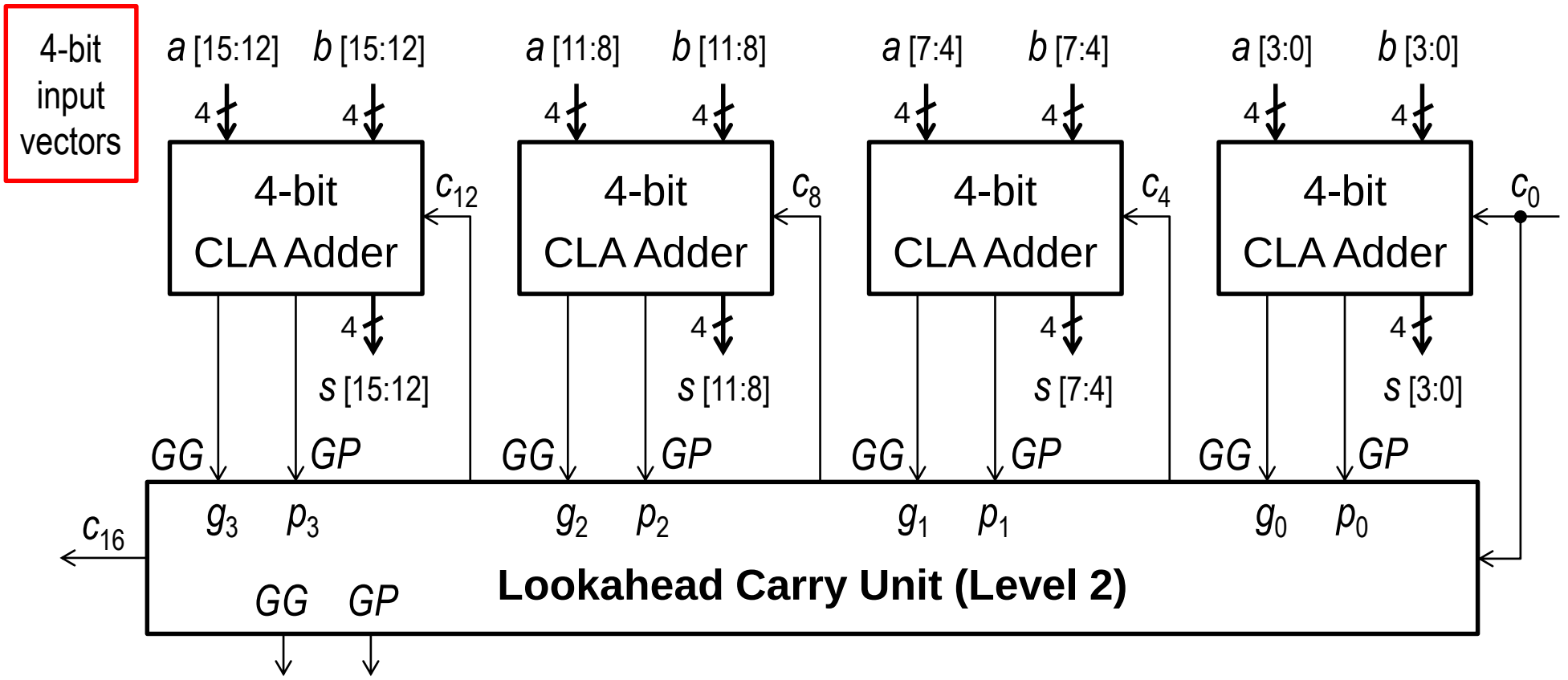
- All generate and propagate signals are produced in parallel
- Delay of all g_i and p_i = Δ_1 (Delay of XOR > Delay of AND)
- Carry bits c_1, c_2, and c_3 are generated in parallel (Delay = Δ_2)
  - Carry-out bit c_4 is not needed to compute the sum bits
- Longest Delay of the 4-bit CLA = $\Delta_1 + \Delta_2 + \Delta_1 = 2\,\Delta_1 + \Delta_2$





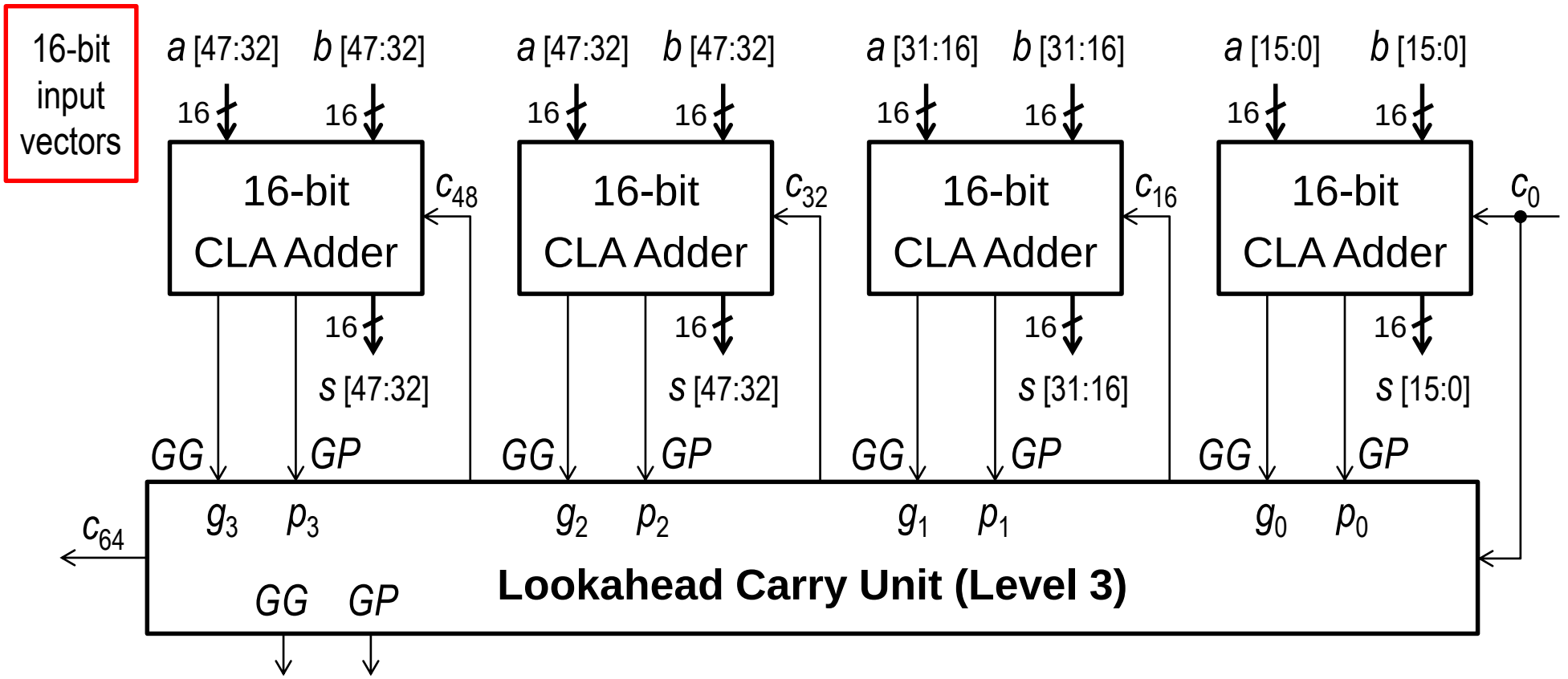
Hierarchical 16-Bit Carry Lookahead Adder

- Designed with Four 4-bit Carry Lookahead Adders (CLA)
- A **Second-Level Lookahead Carry Unit** is required
- Uses **Group Generate** (*GG*) and **Group Propagate** (*GP*) signals



Hierarchical 64-Bit Carry Lookahead Adder

- Designed with Four 16-bit Carry Lookahead Adders (CLA)
- A **Third-Level** **Lookahead Carry Unit** is required
- Uses **Group Generate** (*GG*) and **Group Propagate** (*GP*) signals



Next . . .

- Carry Lookahead Adder
- BCD Adder
- Binary Multiplier
- Carry-Save Adders in Multipliers

BCD Addition

- We use binary arithmetic to add the BCD digits

```
  1000          8
+ 0101        + 5
------        ----
  1101         13 (>9)
```

- If the result is more than 9, it must be corrected to use 2 digits
- To correct the digit, add 6 to the digit sum

```
  1000          8
+ 0101        + 5
------        ----
  1101         13 (>9)
+ 0110        + 6 (add 6)
------        -----------
1 0011 ←      19 (carry + 3)
```

Final answer in BCD

Multiple Digit BCD Addition

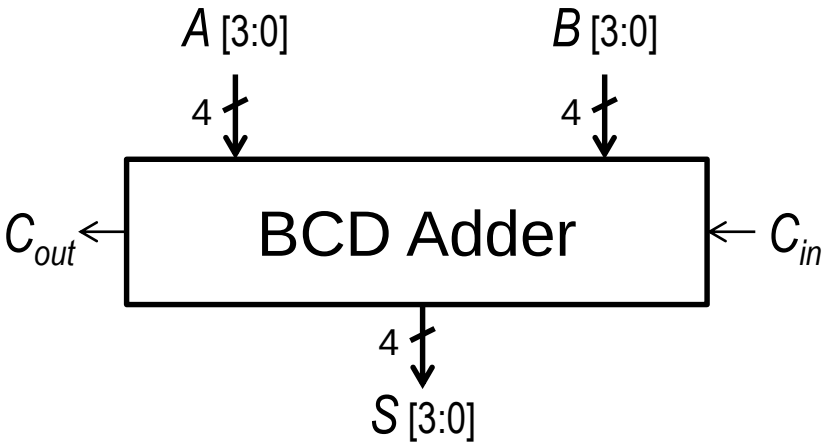
Add: 2905 + 1897 in BCD

Showing carries and digit corrections

carry	+1	+1	+1	
	0010	1001	0000	0101
+	0001	1000	1001	0111
	0100	10010	1010	1100
digit correction		0110	0110	0110
	0100	1000	0000	0010

Final answer: 2905 + 1897 = 4802

BCD Adder



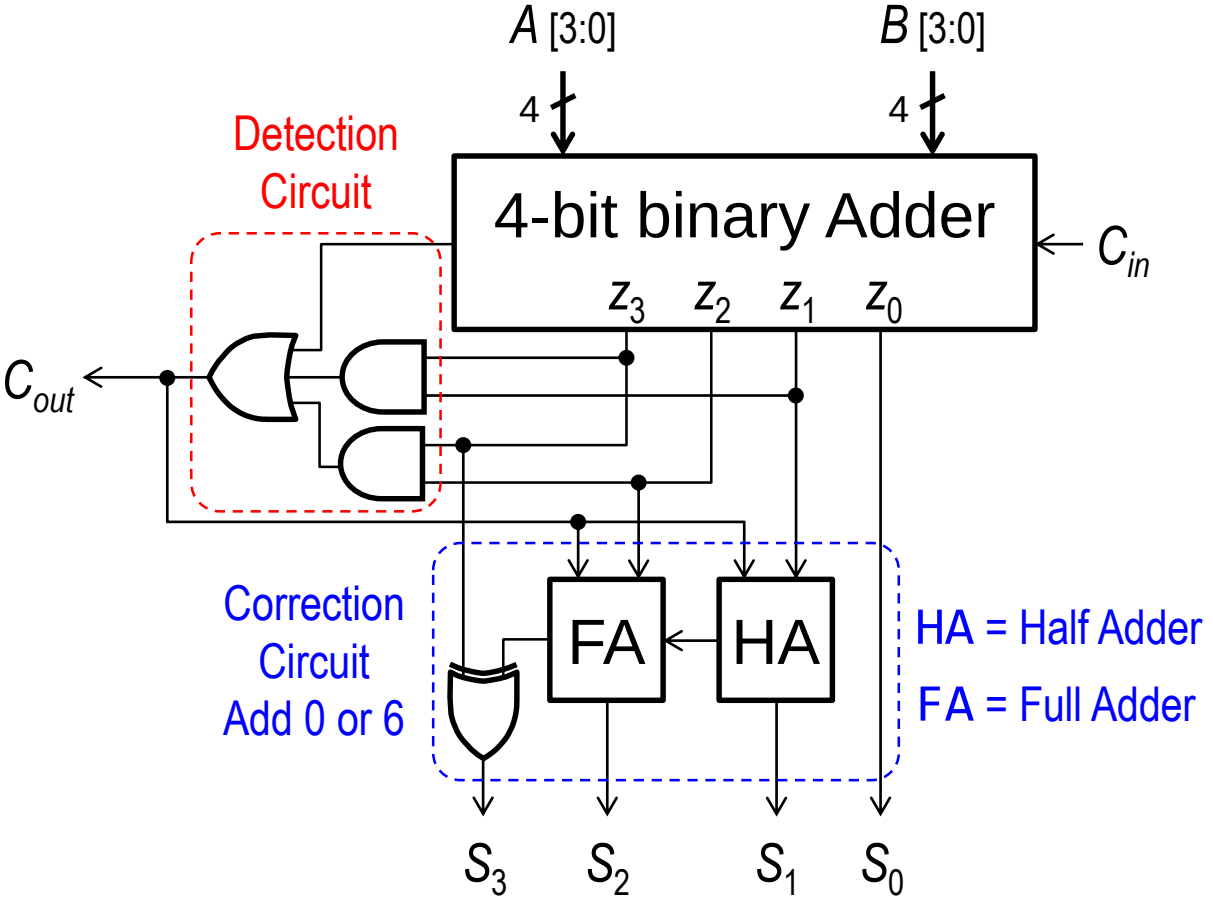
$Z > 9$

z_3z_2 \ z_1z_0	00	01	11	10
00				
01				
11	1	1	1	1
10			1	1

$z_3z_2 + z_3z_1$

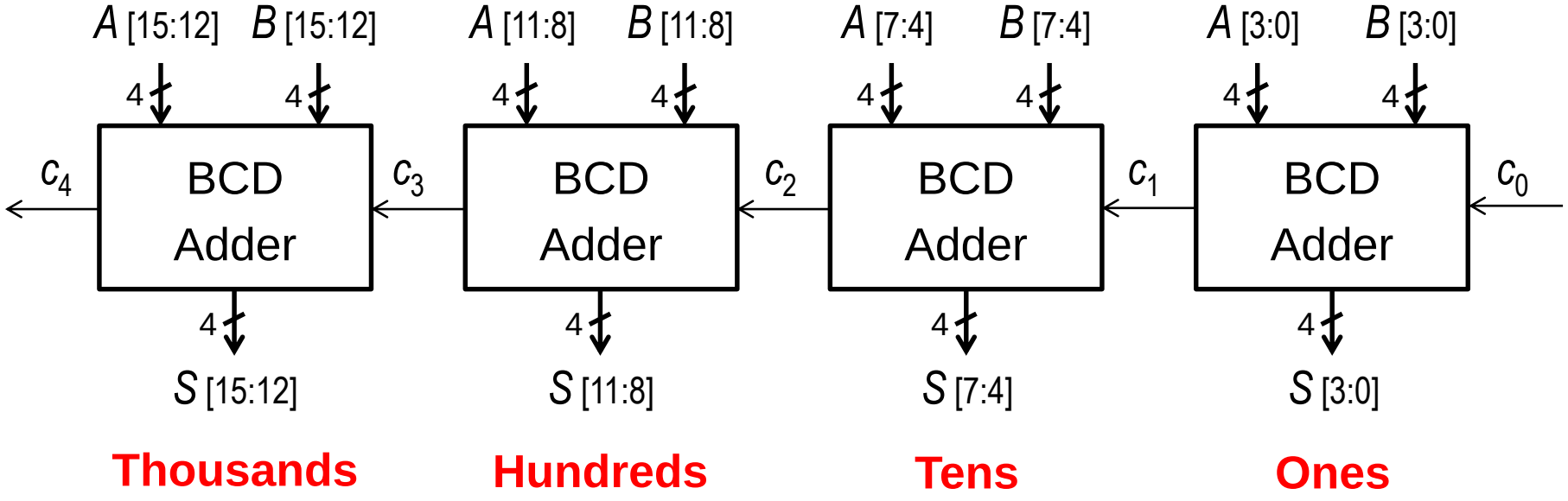
Correction is required if: A+B > 9

Correction circuit adds 0 or 6



Ripple-Carry BCD Adder

- Inputs are BCD digits: 0 to 9
- Sum are BCD digits: **ones**, **tens**, **hundreds**, **thousands**, etc.
- Can be extended to any number of BCD digits
- BCD adders are larger in size than binary adders



Next . . .

- Carry Lookahead Adder
- BCD Adder
- Binary Multiplier
- Carry-Save Adders in Multipliers

Binary Multiplication

- Binary Multiplication is simple:

0×0=0, 0×1=0, 1×0=0, 1×1=1

Multiplicand $1100_2 = 12$

Multiplier × $1101_2 = 13$

```
      1100
     0000
    1100
   1100
--------
10011100
```

Product $10011100_2 = 156$

Binary multiplication

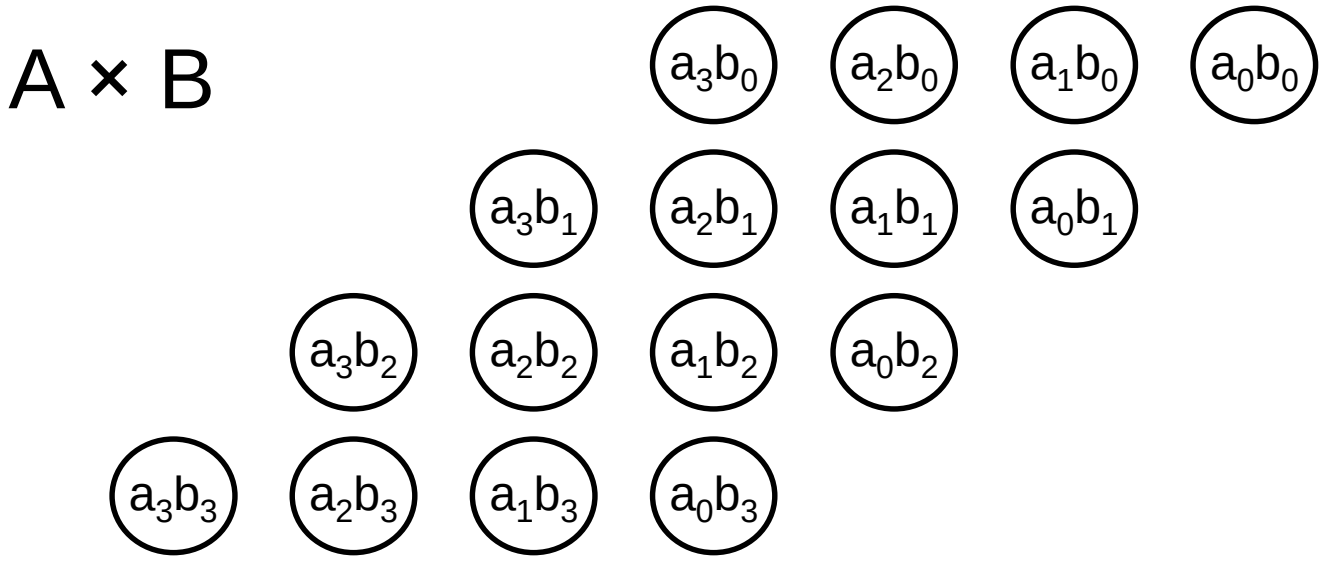
0 × multiplicand = 0

1 × multiplicand = multiplicand

- *n*-bit multiplicand × *n*-bit multiplier = 2*n*-bit product
- Accomplished via shifting and addition

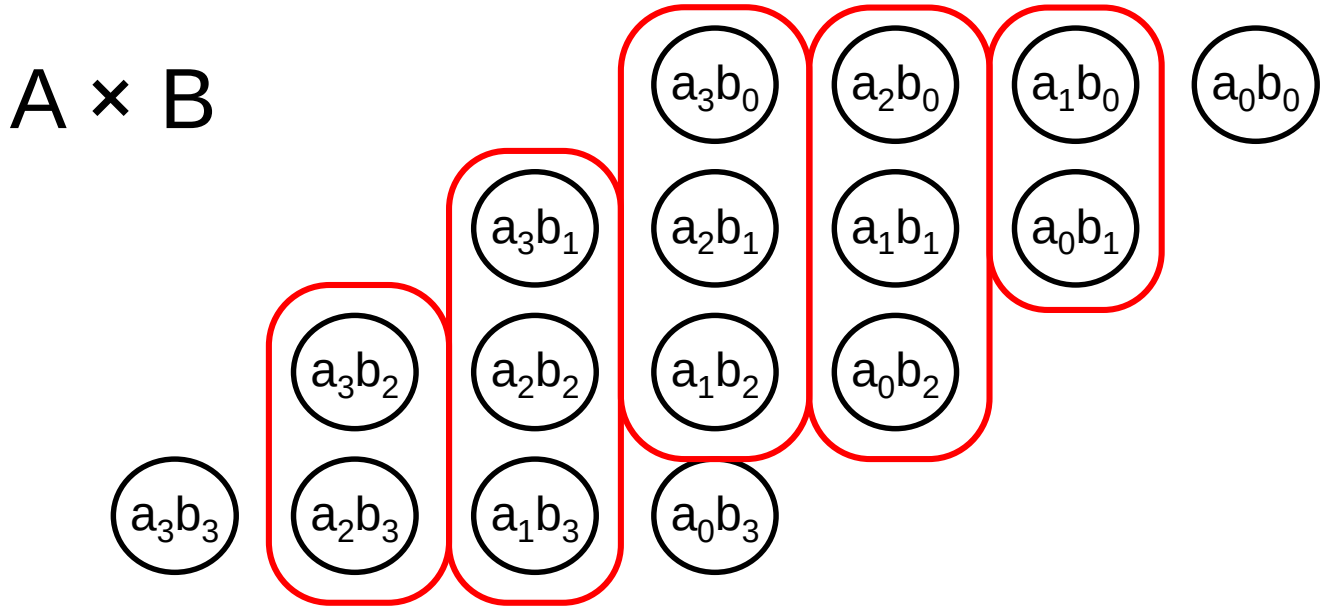
4-bit × 4-bit Binary Multiplier

- Suppose we want to multiply two numbers A and B
  - Example on 4-bit numbers: A = $a_3\ a_2\ a_1\ a_0$ and B = $b_3\ b_2\ b_1\ b_0$
- Step 1: AND (multiply) each bit of A with each bit of B
  - Requires n^2 AND gates and produces n^2 product bits
  - Position of a_ib_j = (i+j). For example, Position of a_2b_3 = 2+3 = 5



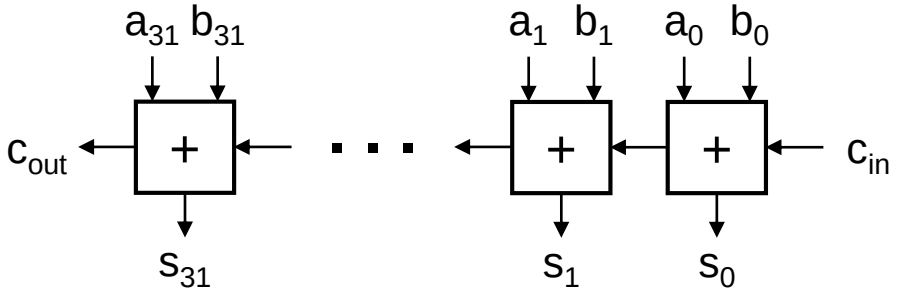
Adding the Bits Vertically

- ADD the product bits vertically using **Carry-Save adders**
  - Full Adder adds three vertical bits
  - Half Adder adds two vertical bits
  - Each adder produces a partial sum and a carry
- Use **Carry-propagate adder** for final addition

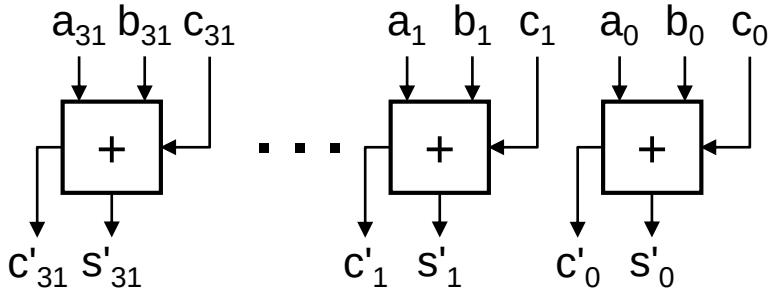


Carry Save Adder

- A n-bit carry-save adder produces two n-bit outputs
  - n-bit partial sum bits and n-bit carry bits
- All the n bits of a carry-save adder work in parallel
  - The carry does not propagate as in a carry-propagate adder
  - This is why a carry-save is faster than a carry-propagate adder
- Useful when adding multiple numbers (as in a multiplier)



Carry-Propagate Adder



Carry-Save Adder

Carry-Save Adders in a Multiplier

Step 1: Use **carry save adders** to add the partial products

- ✧ Reduce the partial products to just two numbers

Step 2: Use **carry-propagate adder** to add last two numbers