

Arithmetic Circuits 2

EE 200

Digital Logic Circuit Design

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Presentation Outline

- ❖ BCD Adder
- ❖ Magnitude Comparator
- ❖ Binary Multiplier
- ❖ Carry-Save Adders in Multipliers

BCD Addition

- ❖ We use binary arithmetic to add the BCD digits

$$\begin{array}{r} 1000 \\ + 0101 \\ \hline 1101 \end{array}$$

$$\begin{array}{r} 8 \\ + 5 \\ \hline 13 (>9) \end{array}$$

- ❖ 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

$$\begin{array}{r} 1000 \\ + 0101 \\ \hline 1101 \\ + 0110 \\ \hline 1\ 0011 \end{array}$$

← Final answer
in BCD

$$\begin{array}{r} 8 \\ + 5 \\ \hline 13 (>9) \\ + 6 \text{ (add 6)} \\ \hline 19 \text{ (carry + 3)} \end{array}$$

Multiple Digit BCD Addition

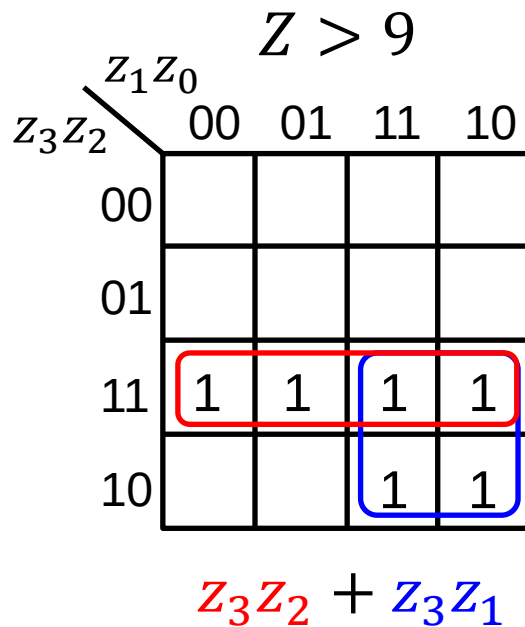
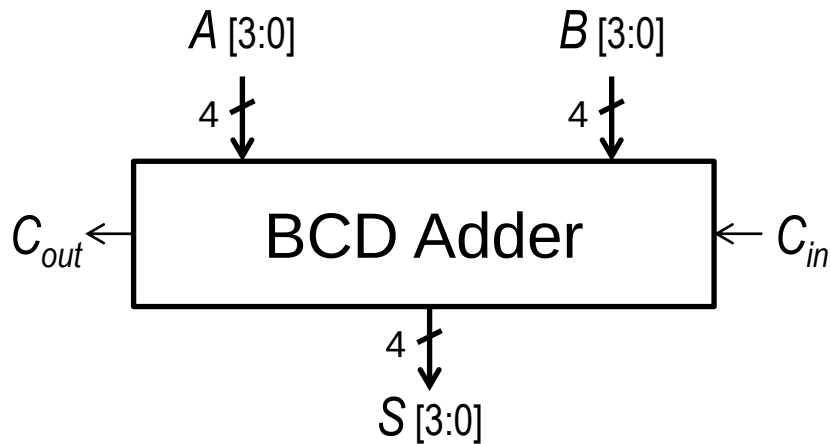
Add: 2905 + 1897 in BCD

Showing carries and digit corrections

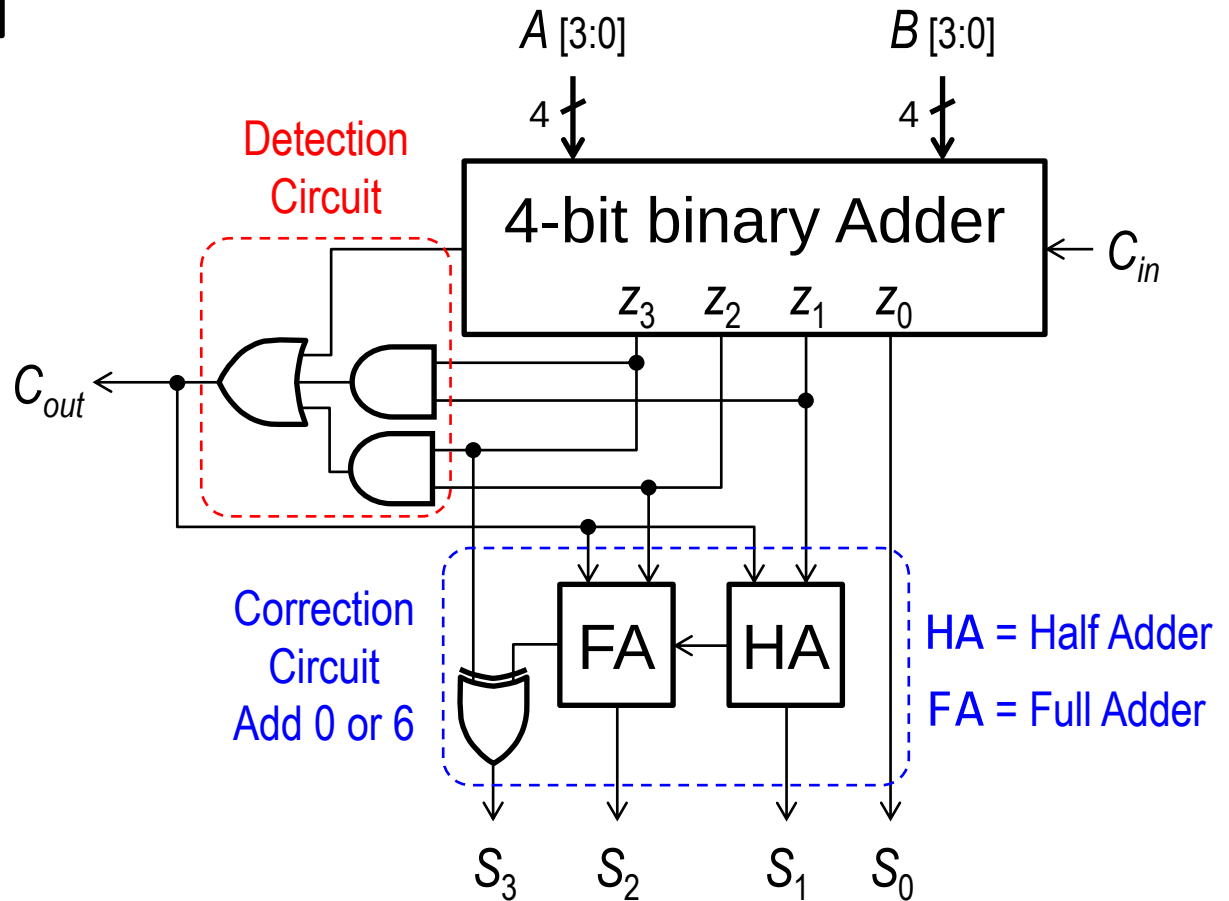
carry	+1	+1	+1	
+	0010	1001	0000	0101
	0001	1000	1001	0111
	0100	10010	1010	1100
digit correction		0110	0110	0110
	0100	1000	0000	0010

Final answer: $2905 + 1897 = 4802$

BCD Adder

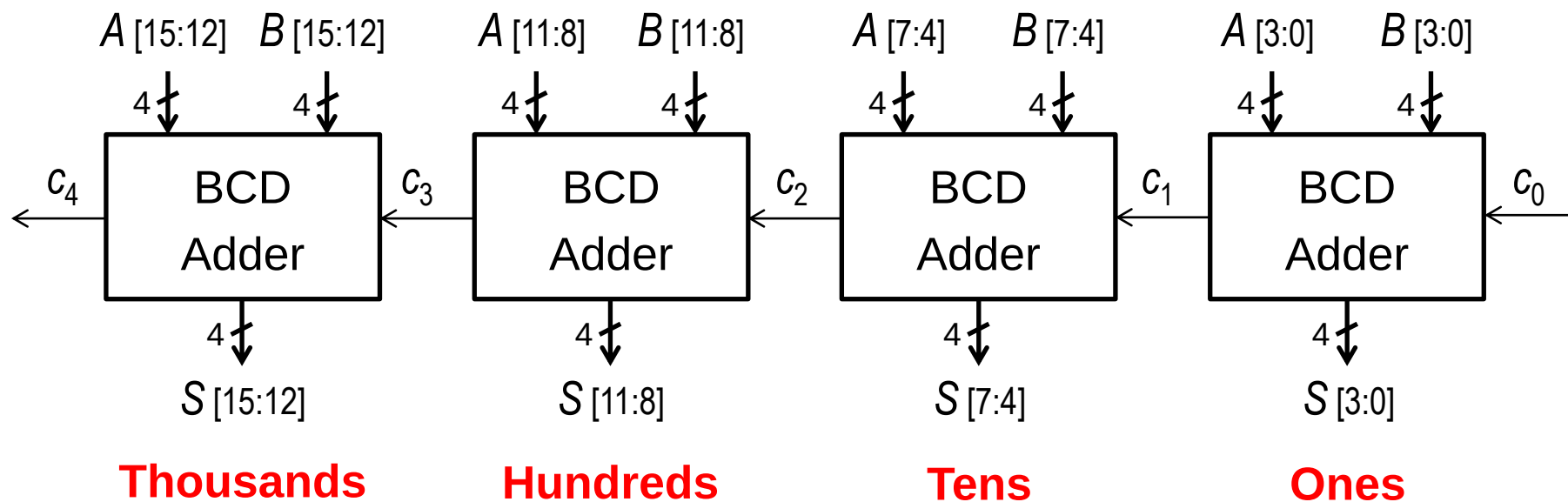


Correction is required if: $A+B > 9$
Correction circuit adds 0 or 6



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 . . .

- ❖ BCD Adder

- ❖ Magnitude Comparator

- ❖ Binary Multiplier

- ❖ Carry-Save Adders in Multipliers

Magnitude Comparator

❖ A combinational circuit that compares two unsigned integers

❖ Two Inputs:

✧ Unsigned integer A (m -bit number)

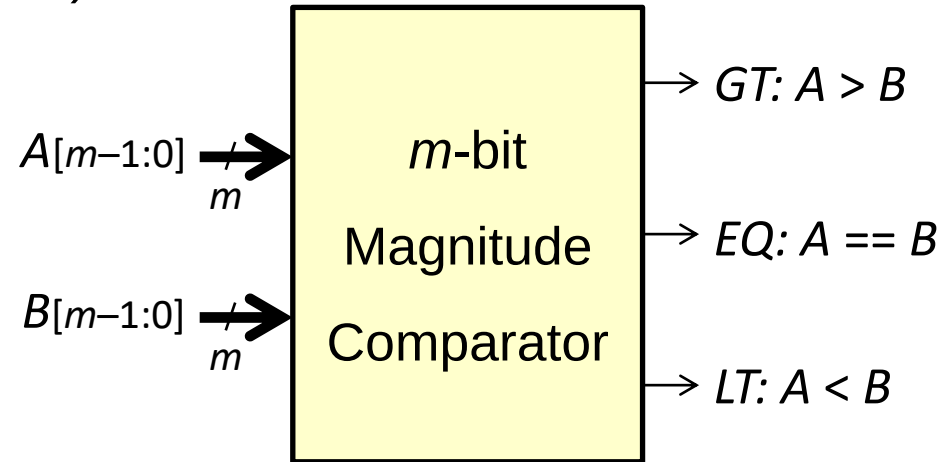
✧ Unsigned integer B (m -bit number)

❖ Three outputs:

✧ $A > B$ (GT output)

✧ $A == B$ (EQ output)

✧ $A < B$ (LT output)



❖ Exactly one of the three outputs must be equal to 1

❖ While the remaining two outputs must be equal to 0

Example: 4-bit Magnitude Comparator

❖ Inputs:

✧ $A = A_3A_2A_1A_0$

✧ $B = B_3B_2B_1B_0$

✧ 8 bits in total → 256 possible combinations

✧ Not simple to design using conventional K-map techniques

❖ The magnitude comparator can be designed at a higher level

❖ Let us implement first the EQ output (A is equal to B)

✧ $EQ = 1 \leftrightarrow A_3 == B_3, A_2 == B_2, A_1 == B_1, \text{ and } A_0 == B_0$

✧ Define: $E_i = (A_i == B_i) = A_iB_i + A'_iB'_i$

✧ Therefore, $EQ = (A == B) = E_3E_2E_1E_0$

The Greater Than Output

Given the 4-bit input numbers: A and B

1. If $A_3 > B_3$ then $GT = 1$, irrespective of the lower bits of A and B

Define: $G_3 = A_3 B'_3$ ($A_3 == 1$ and $B_3 == 0$)

2. If $A_3 == B_3$ ($E_3 == 1$), we compare A_2 with B_2

Define: $G_2 = A_2 B'_2$ ($A_2 == 1$ and $B_2 == 0$)

3. If $A_3 == B_3$ and $A_2 == B_2$, we compare A_1 with B_1

Define: $G_1 = A_1 B'_1$ ($A_1 == 1$ and $B_1 == 0$)

4. If $A_3 == B_3$ and $A_2 == B_2$ and $A_1 == B_1$, we compare A_0 with B_0

Define: $G_0 = A_0 B'_0$ ($A_0 == 1$ and $B_0 == 0$)

Therefore, $GT = G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0$

The Less Than Output

We can derive the expression for the LT output, similar to GT

Given the 4-bit input numbers: A and B

1. If $A_3 < B_3$ then $LT = 1$, irrespective of the lower bits of A and B

Define: $L_3 = A'_3 B_3$ ($A_3 == 0$ and $B_3 == 1$)

2. If $A_3 = B_3$ ($E_3 == 1$), we compare A_2 with B_2

Define: $L_2 = A'_2 B_2$ ($A_2 == 0$ and $B_2 == 1$)

3. Define: $L_1 = A'_1 B_1$ ($A_1 == 0$ and $B_1 == 1$)

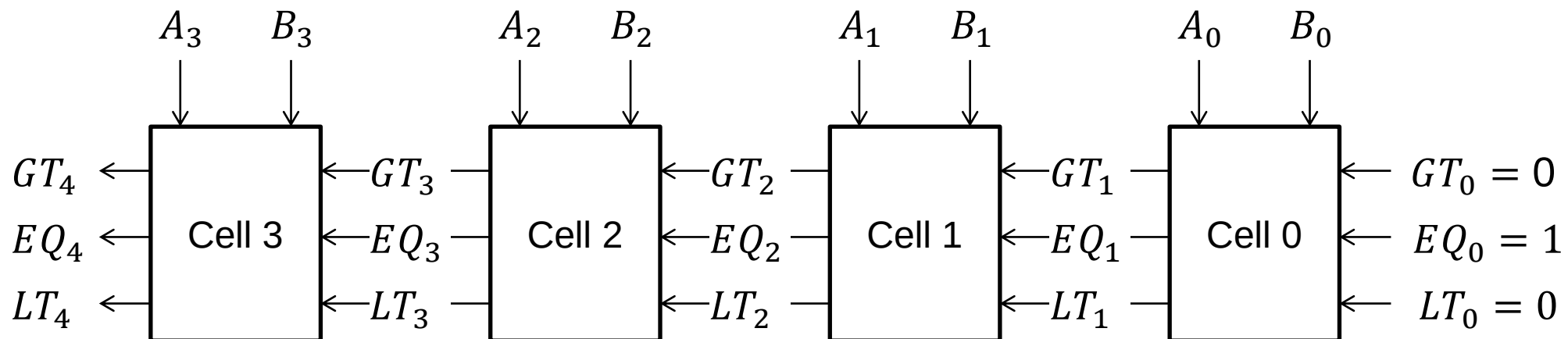
4. Define: $L_0 = A'_0 B_0$ ($A_0 == 0$ and $B_0 == 1$)

Therefore, $LT = L_3 + E_3 L_2 + E_3 E_2 L_1 + E_3 E_2 E_1 L_0$

Knowing GT and EQ , we can also derive $LT = (GT + EQ)'$

Iterative Magnitude Comparator Design

- ❖ The Magnitude comparator can also be designed iteratively
 - 4-bit magnitude comparator is implemented using 4 identical cells
 - Design can be extended to any number of cells
- ❖ Comparison starts at least-significant bit
- ❖ Final comparator output: $GT = GT_4$, $EQ = EQ_4$, $LT = LT_4$



Cell Implementation

❖ Each Cell i receives as inputs:

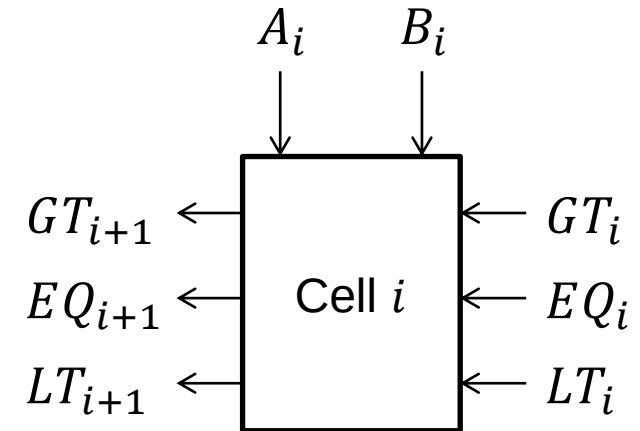
Bit i of inputs A and B : A_i and B_i

GT_i , EQ_i , and LT_i from cell $(i - 1)$

❖ Each Cell i produces three outputs:

GT_{i+1} , EQ_{i+1} , and LT_{i+1}

Outputs of cell i are inputs to cell $(i + 1)$



❖ **Output Expressions of Cell i**

$$EQ_{i+1} = E_i EQ_i$$

$$E_i = A_i' B_i' + A_i B_i \text{ (} A_i \text{ equals } B_i \text{)}$$

$$GT_{i+1} = A_i B_i' + E_i GT_i$$

$$A_i B_i' \text{ (} A_i > B_i \text{)}$$

$$LT_{i+1} = A_i' B_i + E_i LT_i$$

$$A_i' B_i \text{ (} A_i < B_i \text{)}$$

Third output can be produced for first two: $LT = (EQ + GT)'$

Next . . .

❖ BCD Adder

❖ Magnitude Comparator

❖ Binary Multiplier

❖ Carry-Save Adders in Multipliers

Binary Multiplication

❖ Binary Multiplication is simple:

$$0 \times 0 = 0, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0, \quad 1 \times 1 = 1$$

Multiplicand		$1100_2 = 12$
Multiplier	$\times$	$1101_2 = 13$
		1100
		0000
		1100
		1100
Product		$10011100_2 = 156$

Binary multiplication

$0 \times \text{multiplicand} = 0$

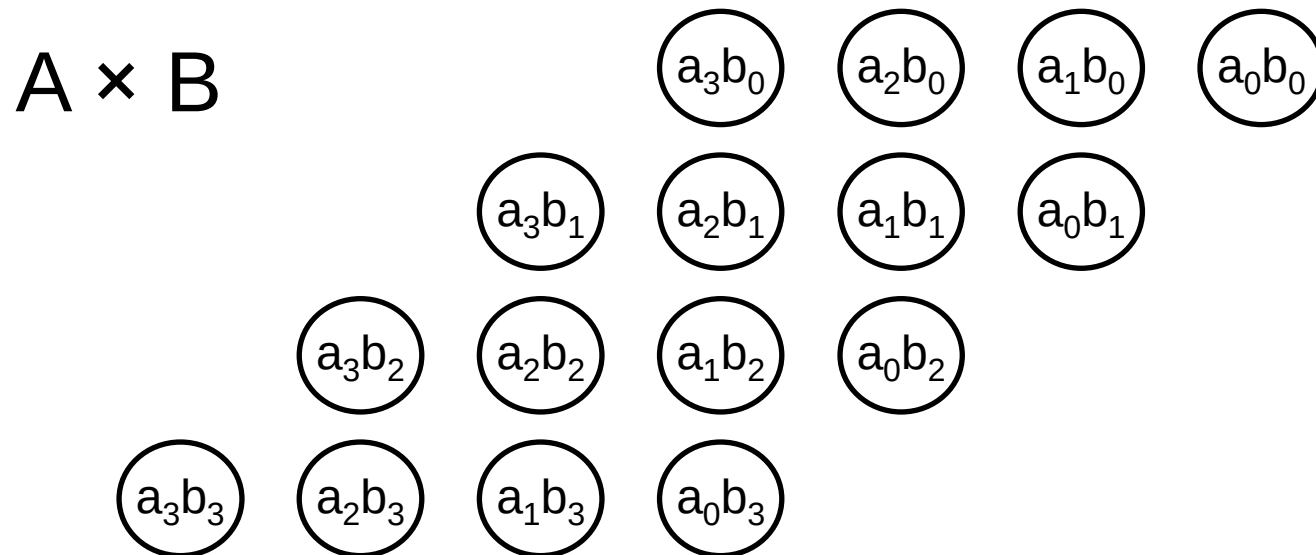
$1 \times \text{multiplicand} = \text{multiplicand}$

❖ n -bit multiplicand $\times$ n -bit multiplier = $2n$ -bit product

❖ Accomplished via **shifting** and **addition**

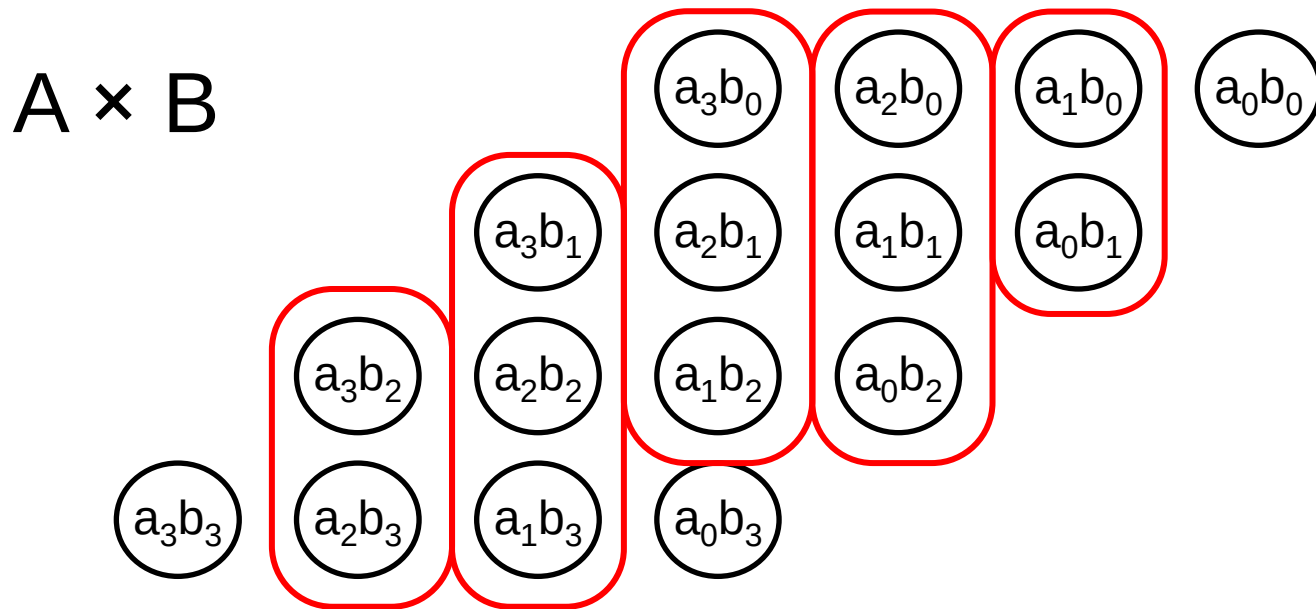
4-bit × 4-bit Binary Multiplier

- ❖ 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 $a_i b_j = (i+j)$. For example, Position of $a_2 b_3 = 2+3 = 5$



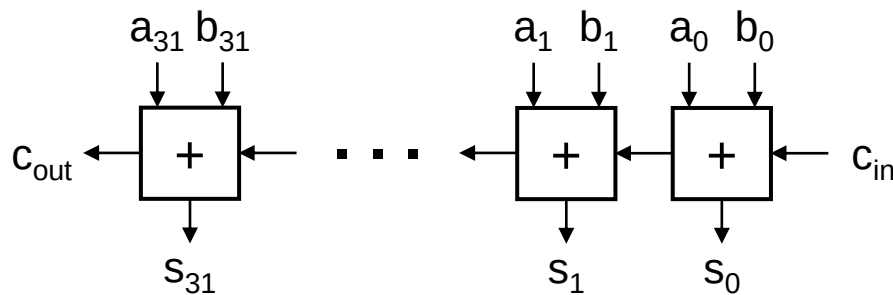
Adding the Bits Vertically

- ❖ 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

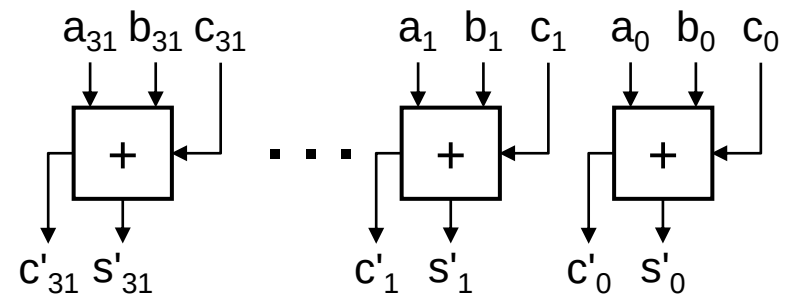


Carry Save Adder

- ❖ 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

