PDS0101

Introduction to Digital Systems

Functions of Combinational Logic I

MMU
MULTIMEDIA UNIVERSITY

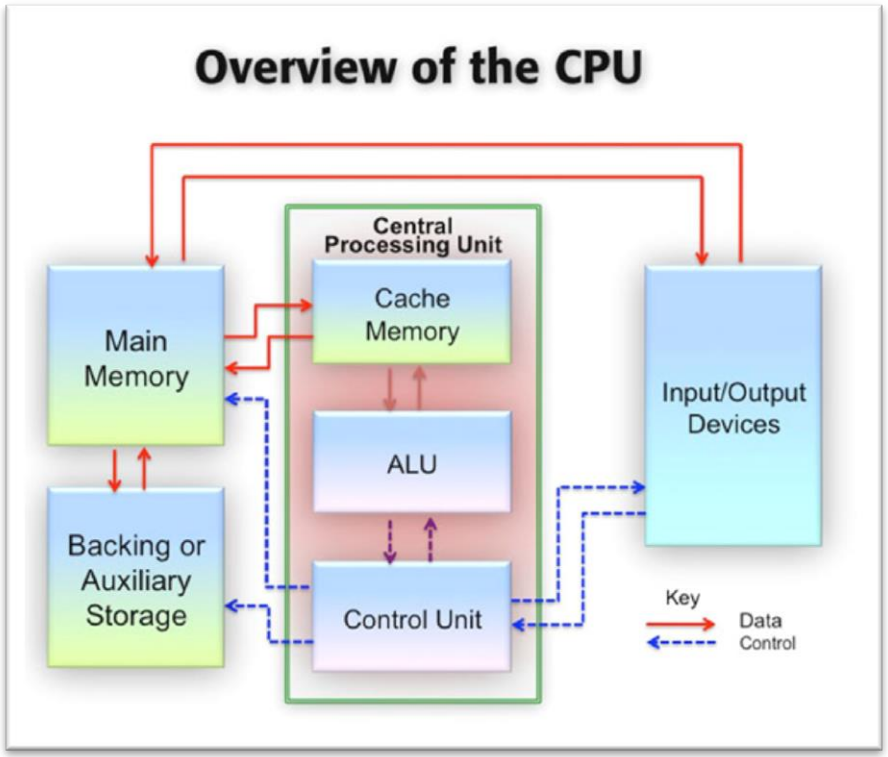
Lecture outcome

- By the end of today's lecture you should know
  - How to distinguish between half- and full-adders
  - How to use full-adders to implement multibit parallel binary adders
  - Differences between ripple carry and look-ahead carry adders
  - Combine parallel adders to create larger bit number adders
  - How to define a decoder
  - How to design a simple decoder circuit to decode any combination of bits

- **NOTE**: contents in this set of slides are intentionally incomplete and content will be shown in class as examples of how they are derived

Adders

- Adders (and rarely but sometimes called *summers)* are logical/electrical circuits that performs addition of digits
- Important in ALU (arithmetic logic unit) in the CPU to calculate memory addresses, indexes etc.
- Although possible to add any type of digit, majority of adders handle binary addition



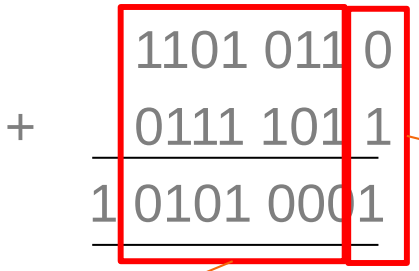
Recap - Binary Addition

- Rules

Augend	Addend	Sum	Carryout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Example: Perform the binary additions of 1101 0110 and 111 1011

  Solution:



LSB addition only takes into account one augend and one addend

Remaining addition steps requires one augend, one addend and one carry

Half-adder

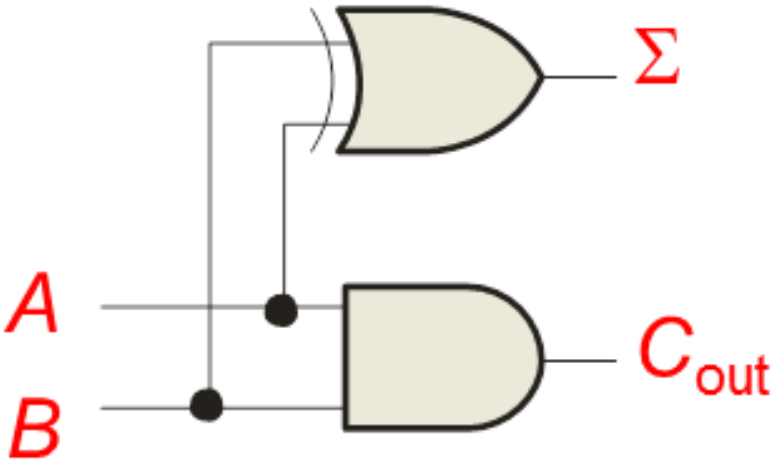
- The half-adder accepts two *single bit* binary digits on its inputs and produces two binary digits on its outputs – a sum bit and a carry bit
  - This circuit needs 2 binary inputs and 2 binary outputs
- The input variables designate the augend and addend bits: the output variables produce the sum and carry
- The carry signal represents an overflow into the next digit of a multi-digit addition
- Based on the binary addition truth table, the simplest half-adder design uses an XOR gate for S and an AND gate for C

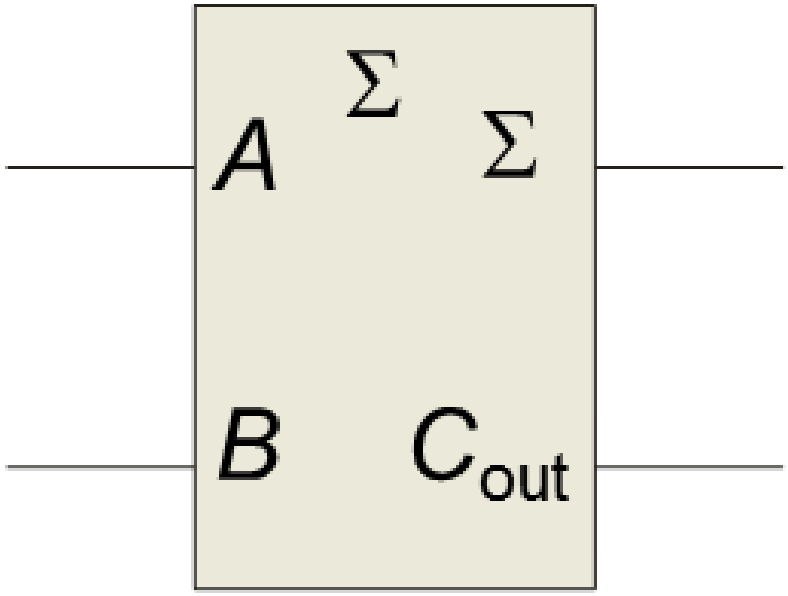
Inputs		Outputs	
A	*B*	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = AB' + A'B$

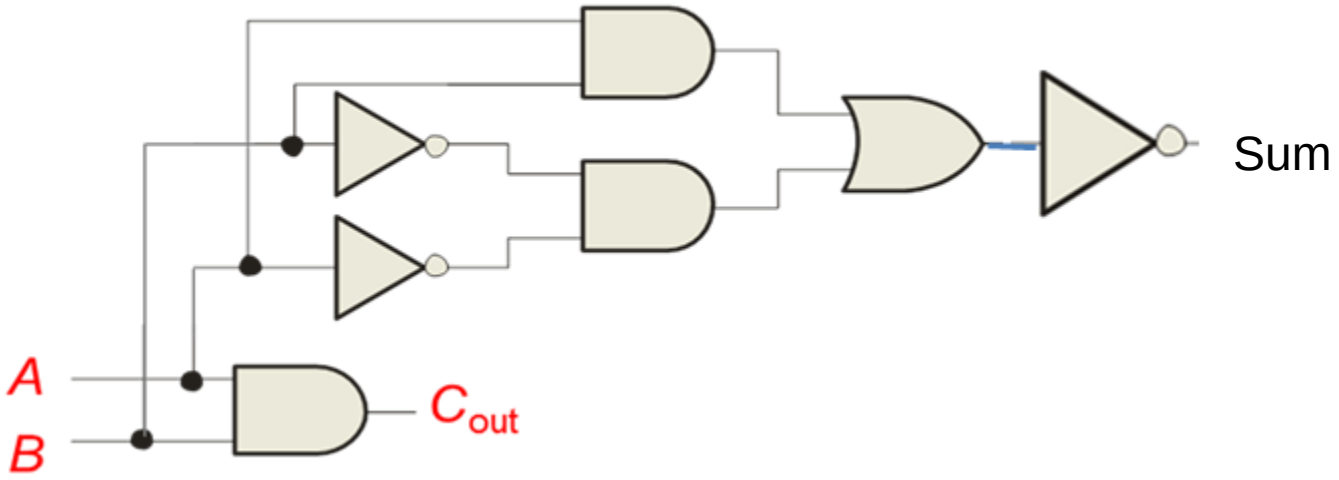
$C_{out} = AB$

- The logic symbol and equivalent circuit are:





- Implementing practices from combinational logic analysis, the adder can be implemented purely with AND, OR and NOT gates.



Sum = (A'B' + AB)'
or A'B + AB'

Full-adder

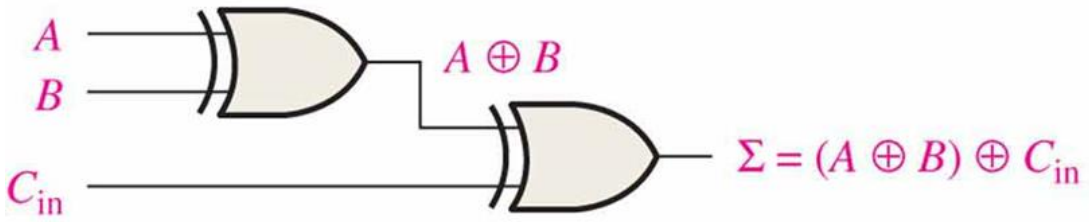
- By contrast, a full adder now has three binary inputs (A, B, and Carry in) and two binary outputs (Carry out and Sum)
- The truth table summarizes the operation:
  - The full-add must sum two input bits *and* the carry in
  - Starting from the half-adder, the sum of the input bits A and B is the XOR of both bits
  - To add C_{in} to this, it is XORed again thus yielding the expression for sum output as

$$\sum = (A \oplus B) \oplus C_{in}$$

  - This means two XOR gates are needed to implement the *sum* function in the full-adder

Inputs			Outputs	
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

NOTE: the 'full-adder' still only adds one (1) bit to another!!

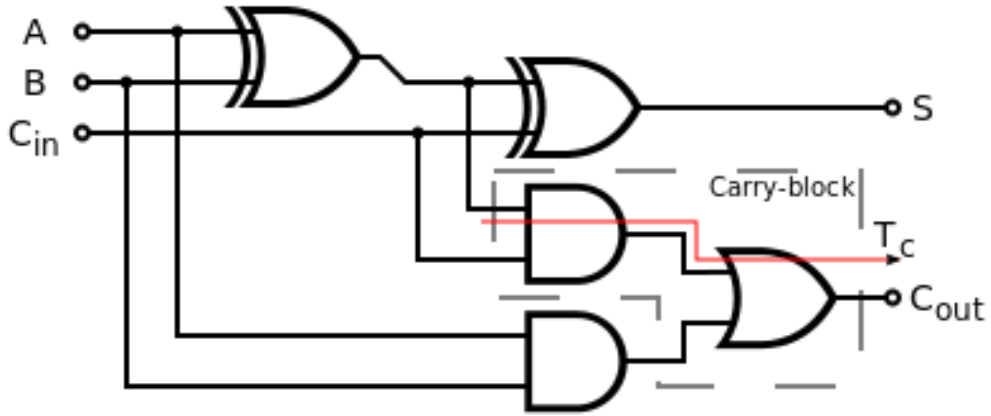


- Carry output is 1 when <u>both</u> inputs to the XOR gates are 1
- The output of the carry of the full-adder is therefore produced by first ANDing A and B as well as $A \oplus B$ with carry in.
- The resulting two terms are ORed to create the carry out
- Thus the boolean expression for carry out is

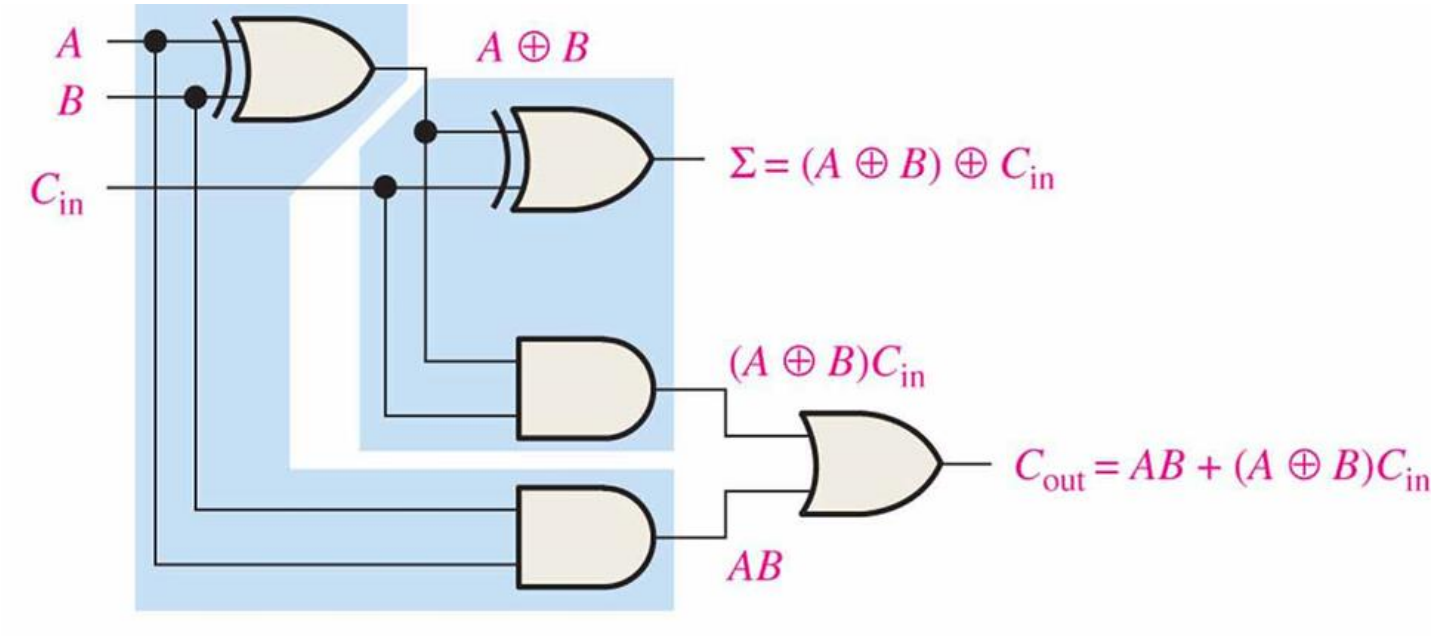
$$C_{out} = AB + (A \oplus B)C_{in}$$

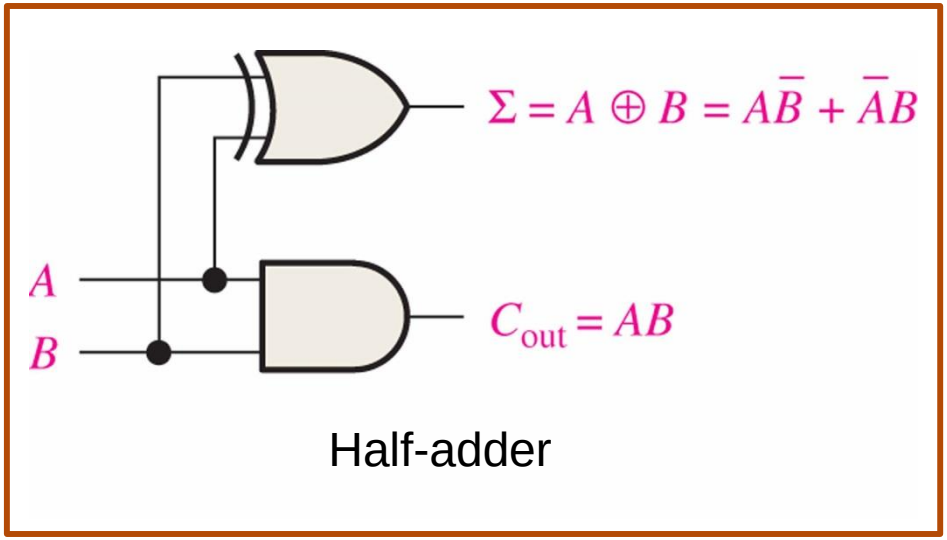
- And the resulting logic circuit is as shown below

Inputs				Outputs	
A	B	$A \oplus B$	C_{in}	C_{out}	Σ
0	0	0	0	0	0
0	0	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	1

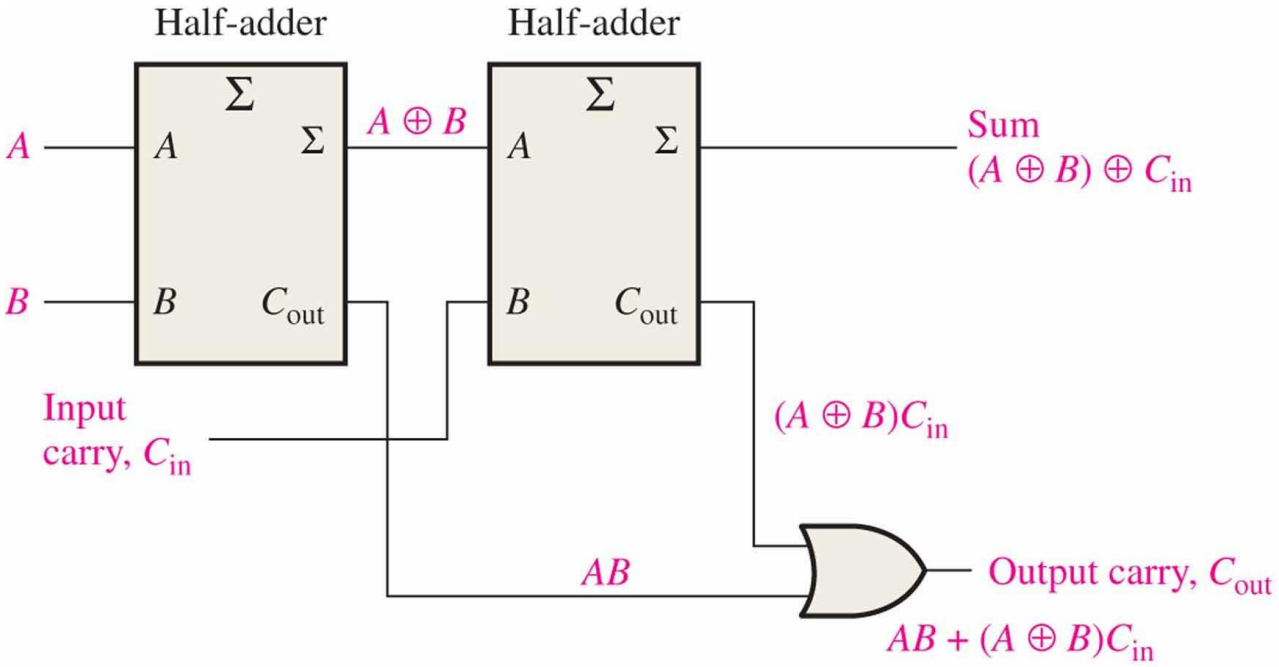


- Two half-adders are actually used to create a full adder

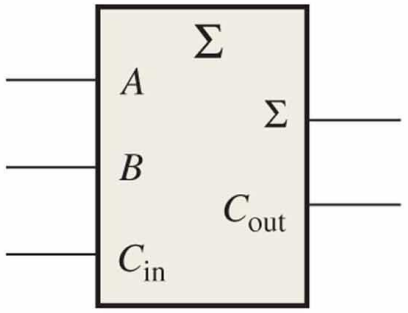




Using logic symbols

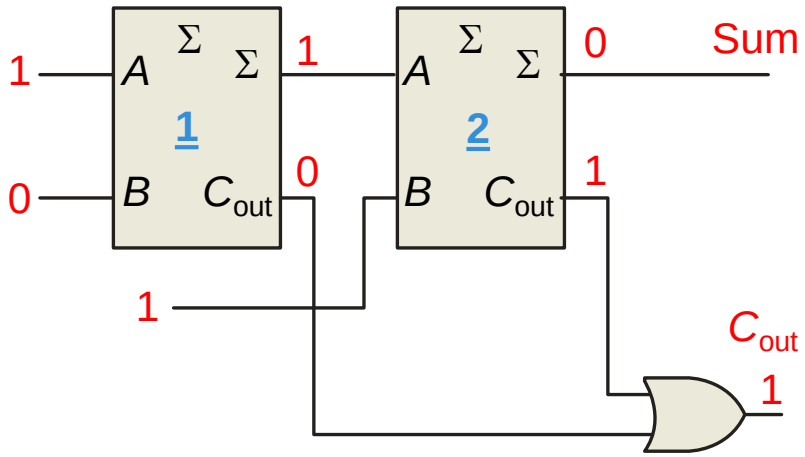


(a) Arrangement of two half-adders to form a full-adder



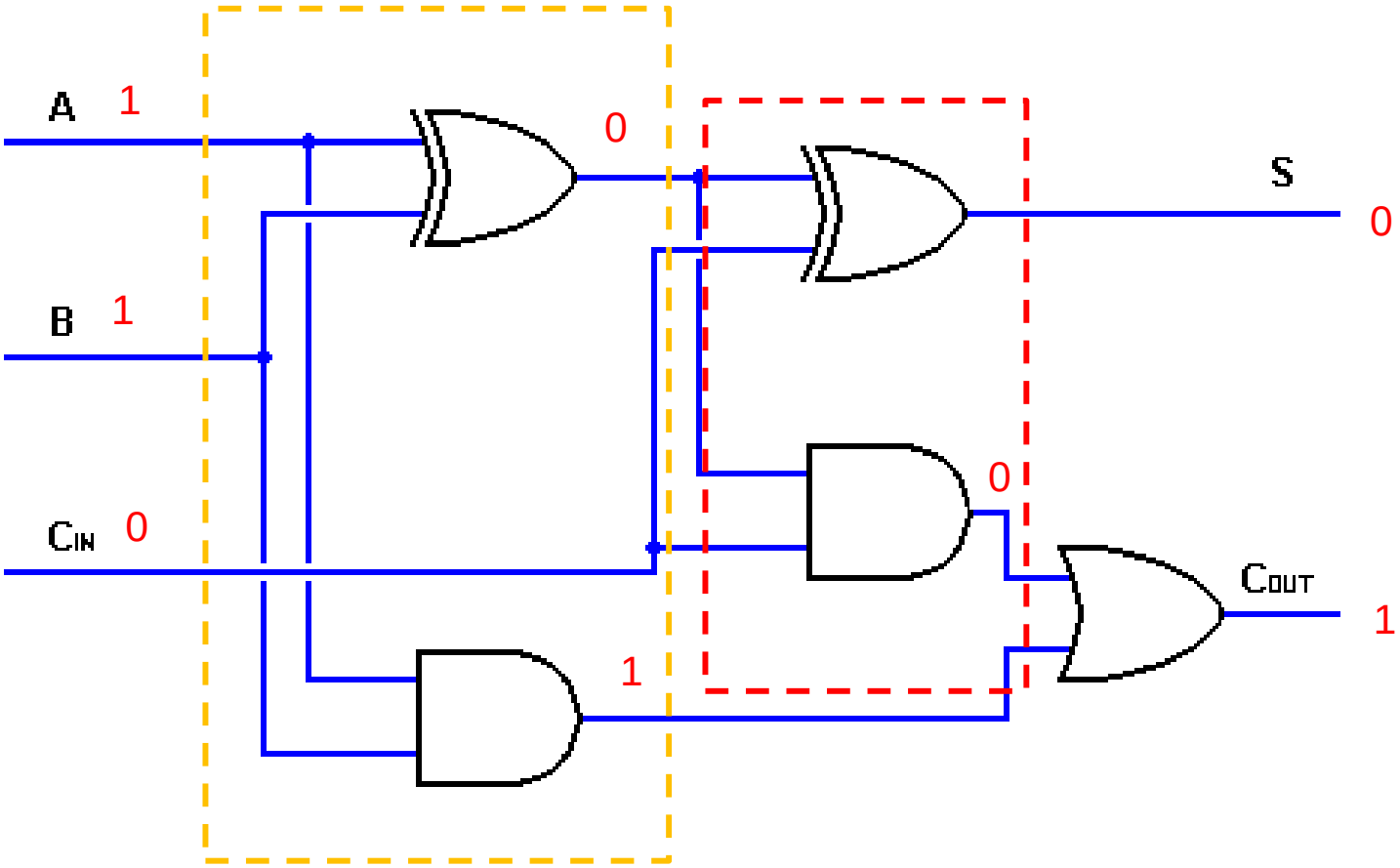
(b) Full-adder logic symbol

How the FA works with 1+1



- The first half-adder has inputs of 1 and 0; therefore the Sum =1 and the Carry out = 0
- The second half-adder has inputs of 1 and 1; therefore the Sum = 0 and the Carry out = 1
- The OR gate has inputs of 1 and 0, therefore the final carry out = 1

- When compared with the logic circuit diagram of the full-adder



Using K-map to create full-adder

- Alternative to using truth table is the use K-map
  - Uses a three-variable K-map

For Carry (C_{out})

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C_{out} = AB + A\,C_{in} + B\,C_{in}$

For Sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$Sum = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$

$$\sum = \overline{AB}C_{in} + \overline{A}B\overline{C_{in}} + A\,\overline{BC_{in}} + ABC_{in}$$

$$= \overline{C_{in}}\left(\overline{A}B + A\overline{B}\right) + C_{in}\left(AB + \overline{AB}\right)$$

$$= \overline{C_{in}}\left(\overline{A}B + A\overline{B}\right) + C_{in}\overline{\left(\overline{A}B + A\overline{B}\right)}$$

If $X = \overline{A}B + A\overline{B} = A \oplus B$ then

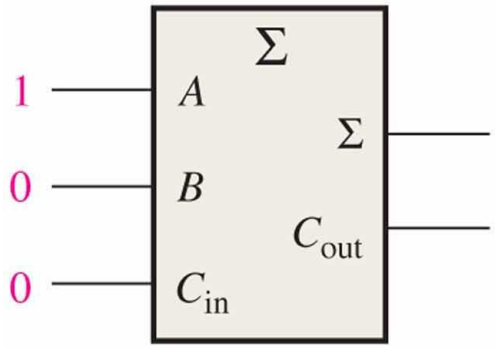
$$\sum = \overline{C_{in}}\left(\overline{A}B + A\overline{B}\right) + C_{in}\overline{\left(\overline{A}B + A\overline{B}\right)}$$

$$= \overline{C_{in}}X + C_{in}\overline{X}$$
$$= C_{in} \oplus X$$
$$= C_{in} \oplus (A \oplus B)$$

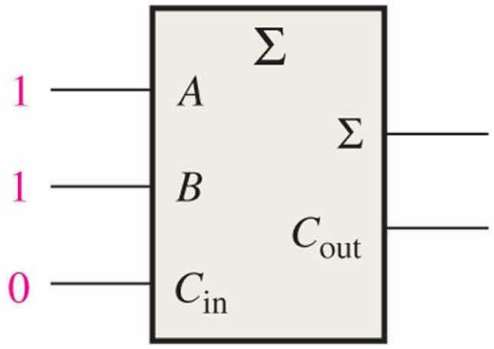
$$C_{out} = AB + AC_{in} + BC_{in}$$
$$= AB\left(C_{in} + \overline{C_{in}}\right) + AC_{in}\left(B + \overline{B}\right) + BC_{in}\left(\mathrm{A} + \overline{A}\right)$$
$$= ABC_{in} + AB\overline{C_{in}} + A\overline{B}C_{in} + \overline{A}BC_{in}$$
$$= \mathrm{AB} + C_{in}(A \oplus B)$$

Exercise

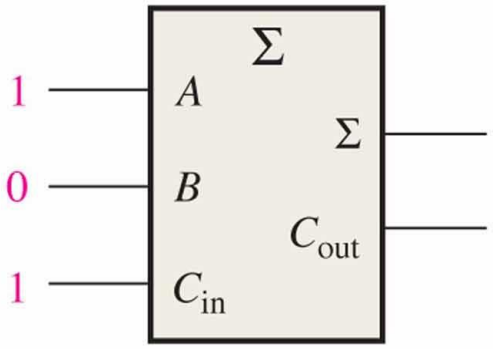
- For each of the full-adders below, determine the outputs for the inputs shown



(a)



(b)



(c)

(a) The input bits are $A = 1$, $B = 0$, and $C_{in} = 0$.

$1 + 0 + 0 = 1$ with no carry

Therefore, $\Sigma = 1$ and $C_{out} = 0$.

(b) The input bits are $A = 1$, $B = 1$, and $C_{in} = 0$.

$1 + 1 + 0 = 0$ with a carry of 1

Therefore, $\Sigma = 0$ and $C_{out} = 1$.

(c) The input bits are $A = 1$, $B = 0$, and $C_{in} = 1$.

$1 + 0 + 1 = 0$ with a carry of 1

Therefore, $\Sigma = 0$ and $C_{out} = 1$.

Parallel adders

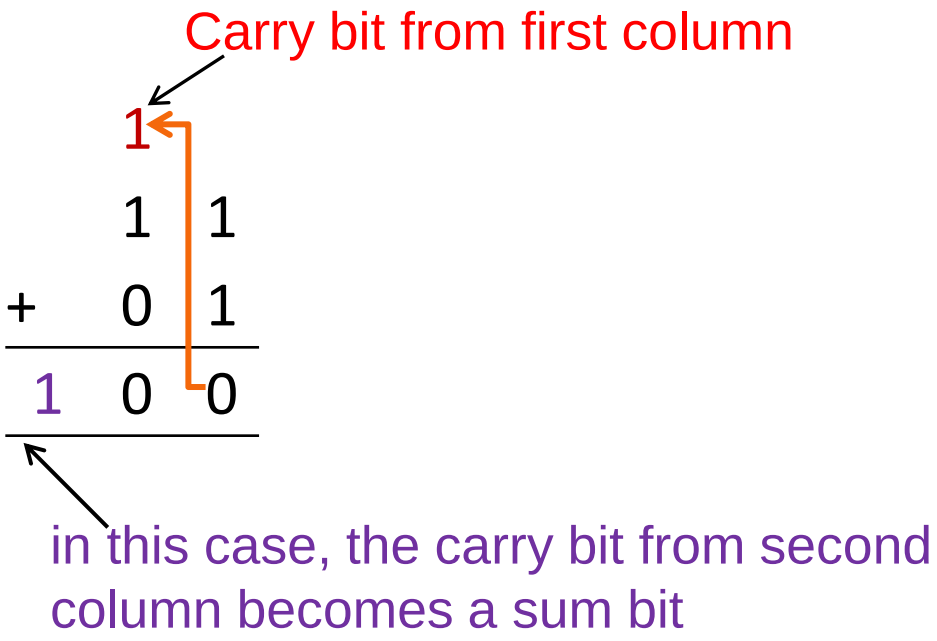
- A parallel adder is a digital circuit that produces the arithmetic sum of 2 binary numbers with 2 or more bits
- Constructed with full adders connected in cascade, with output carry from each full adder connected to the input carry of next full adder in the chain
- In the logic circuit, the augend bits of number A and the addend bits of number B are designated by subscript numbers from right to left, with subscript 1 denoting the least significant bit
  - E.g. If two binary numbers A and B are added together and each have 2 bits, the notation is

$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline C_O\Sigma_2\Sigma_1 \end{array}$$

- The carries are connected in a chain through the full adders

2-bit parallel adder

- In a 2-bit adder, LSB of two binary numbers are represented by A_1 and B_1
- The next higher bit are A_2 and B_2.
- The resulting three sum bits are $\sum_1$, $\sum_2$ and C_{out}, in which the C_{out} becomes MSB
- The carry output C_{out} of each adder is connected as the carry input of the next higher order

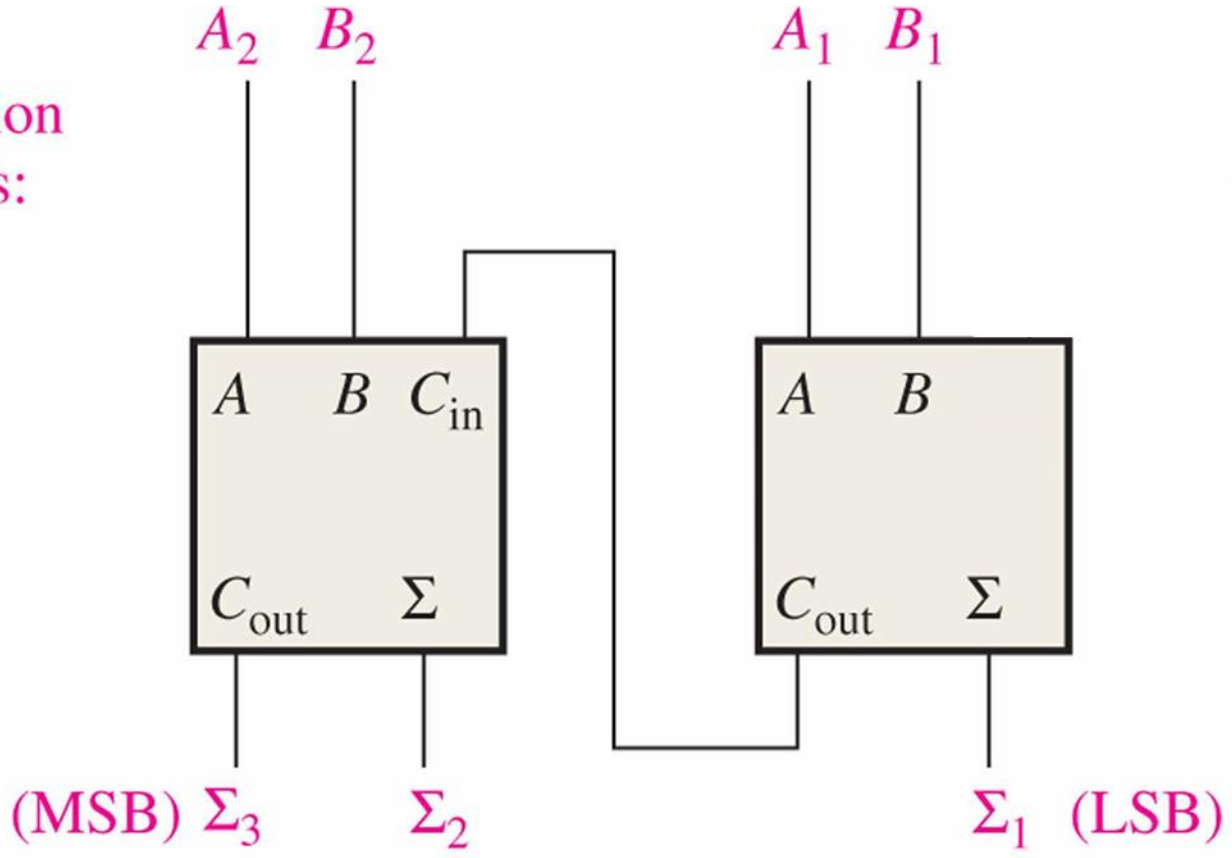


2-bit parallel adder (I)

- 2-bit parallel adder logic circuit using half- and full-adders

General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$

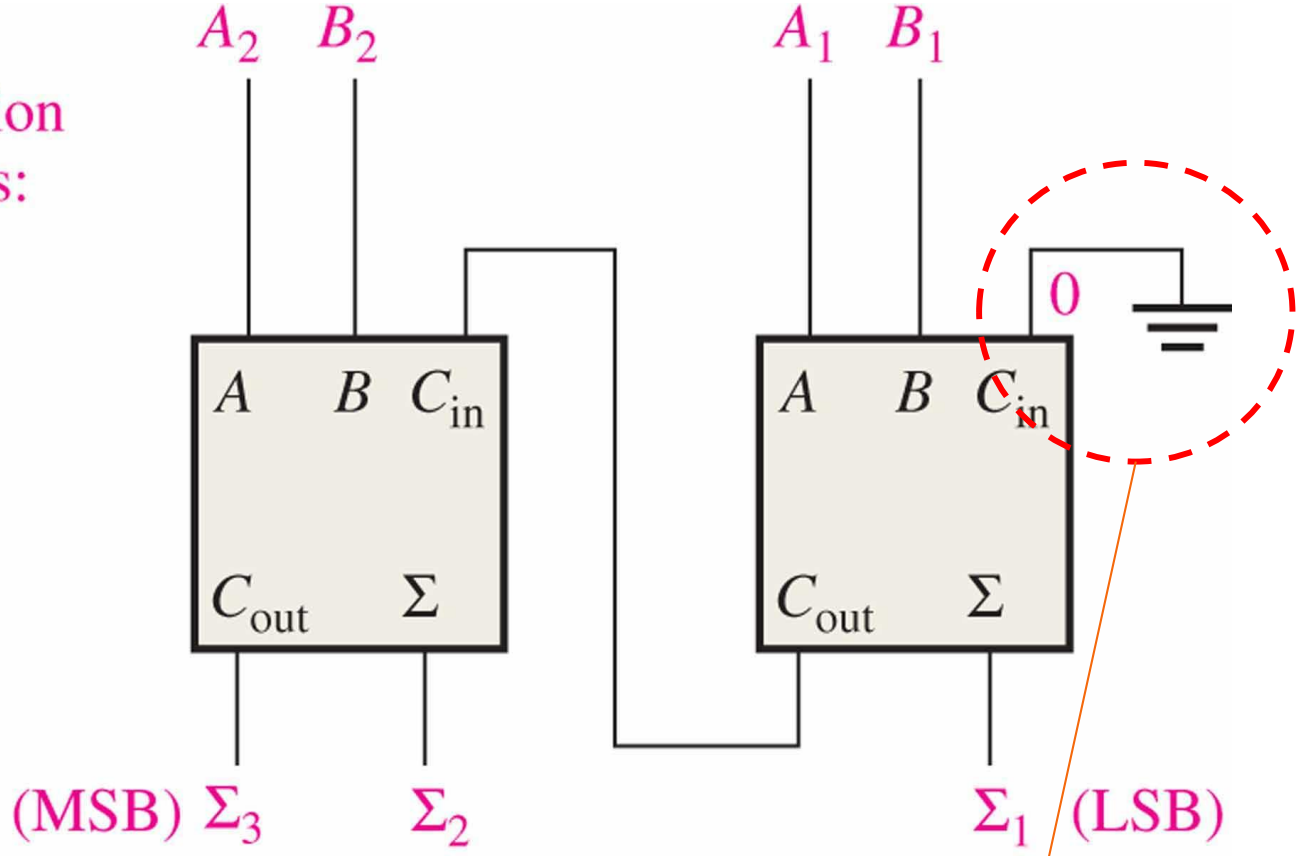


2-bit parallel adder (II)

- 2-bit parallel adder logic circuit using ***only full-adders***

General format, addition of two 2-bit numbers:

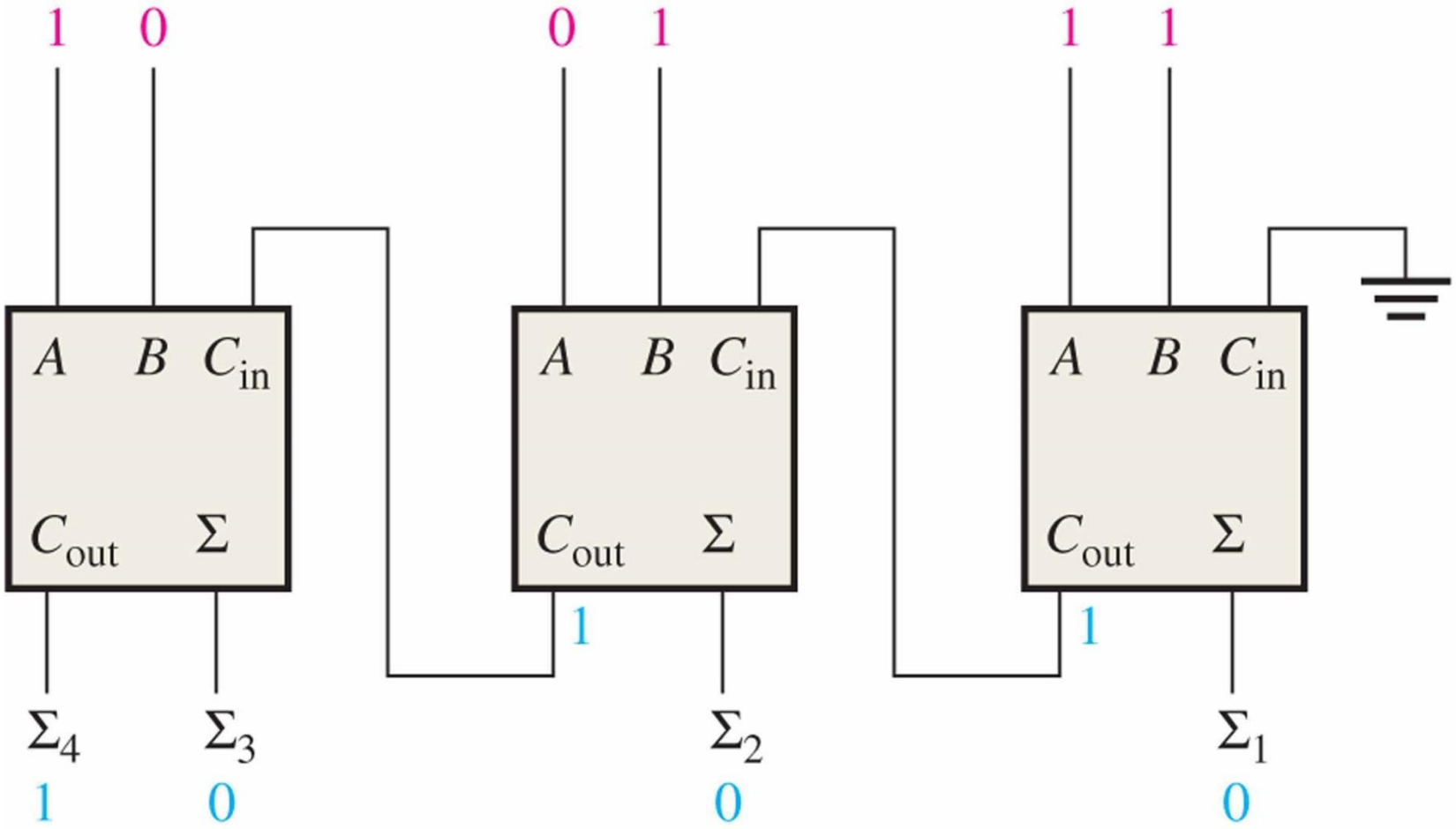
$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$



Note how the first FA carry in is grounded since LSB addition has no use for the carry in

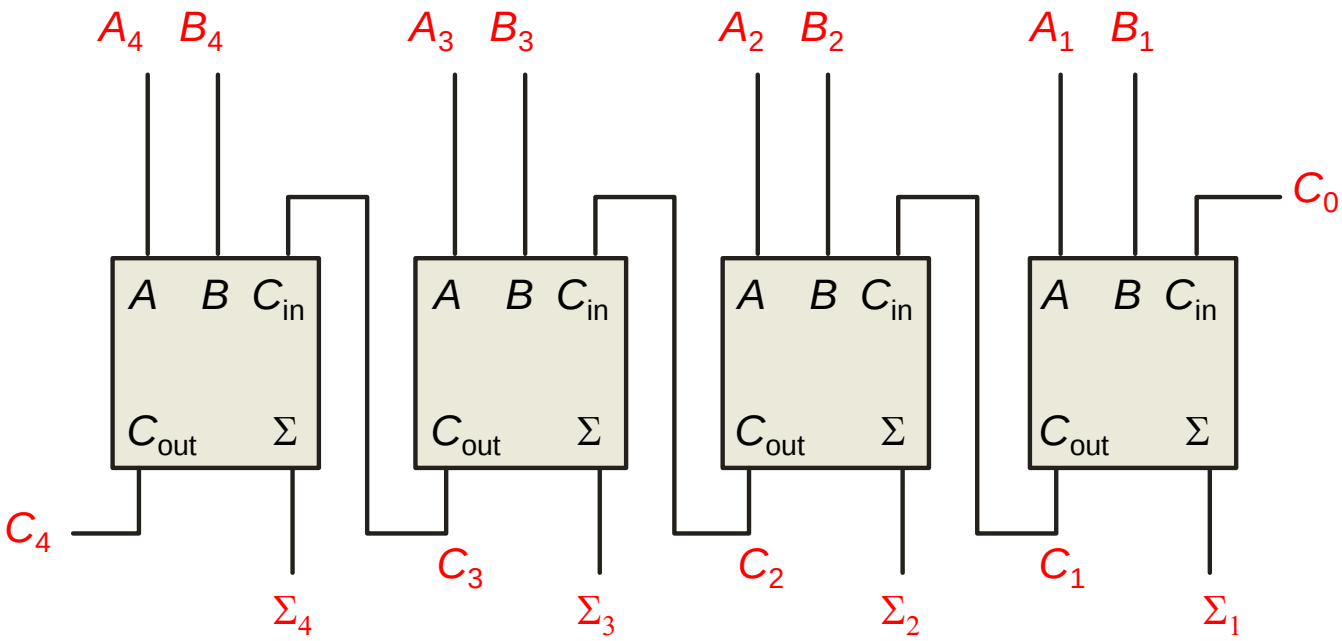
Exercise

- Determine the sum generated by the 3-bit adder below and show the intermediate carries
- What is the result when 101 and 110 are added together?



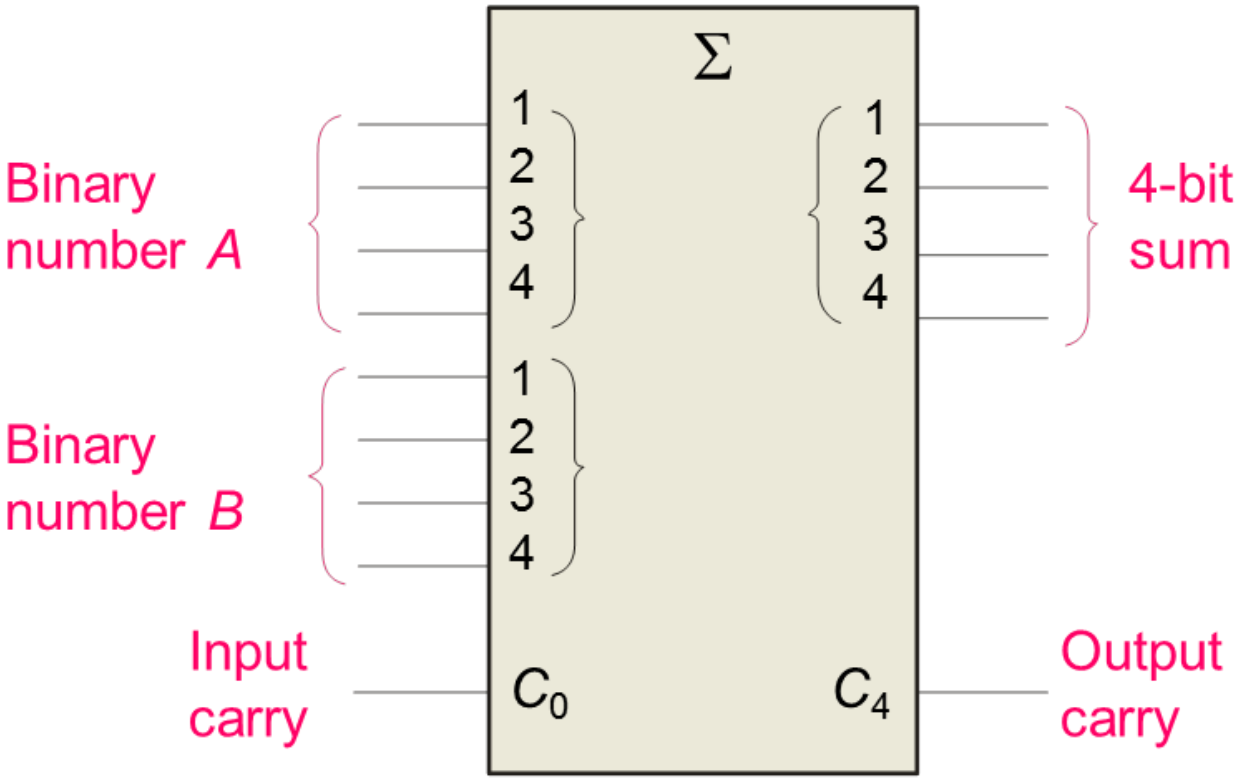
Parallel binary adders

- Multiple full adders can be combined into parallel adders that can add binary numbers with multiple bits.
- A 4-bit nibble adder is shown below, making use of 4 full-adders



- LSB FA adds the 1s from bits A_1 and B_1, followed by the 2s, 4s and 8s column bits being added by the following FAs
- The carry output of each adder is connected to the carry input of next adder called as *internal carries*

- The logic symbol for a 4-bit parallel adder is shown.
- This 4-bit adder includes a carry in (labeled (C_0) and a Carry out (labeled C_4).



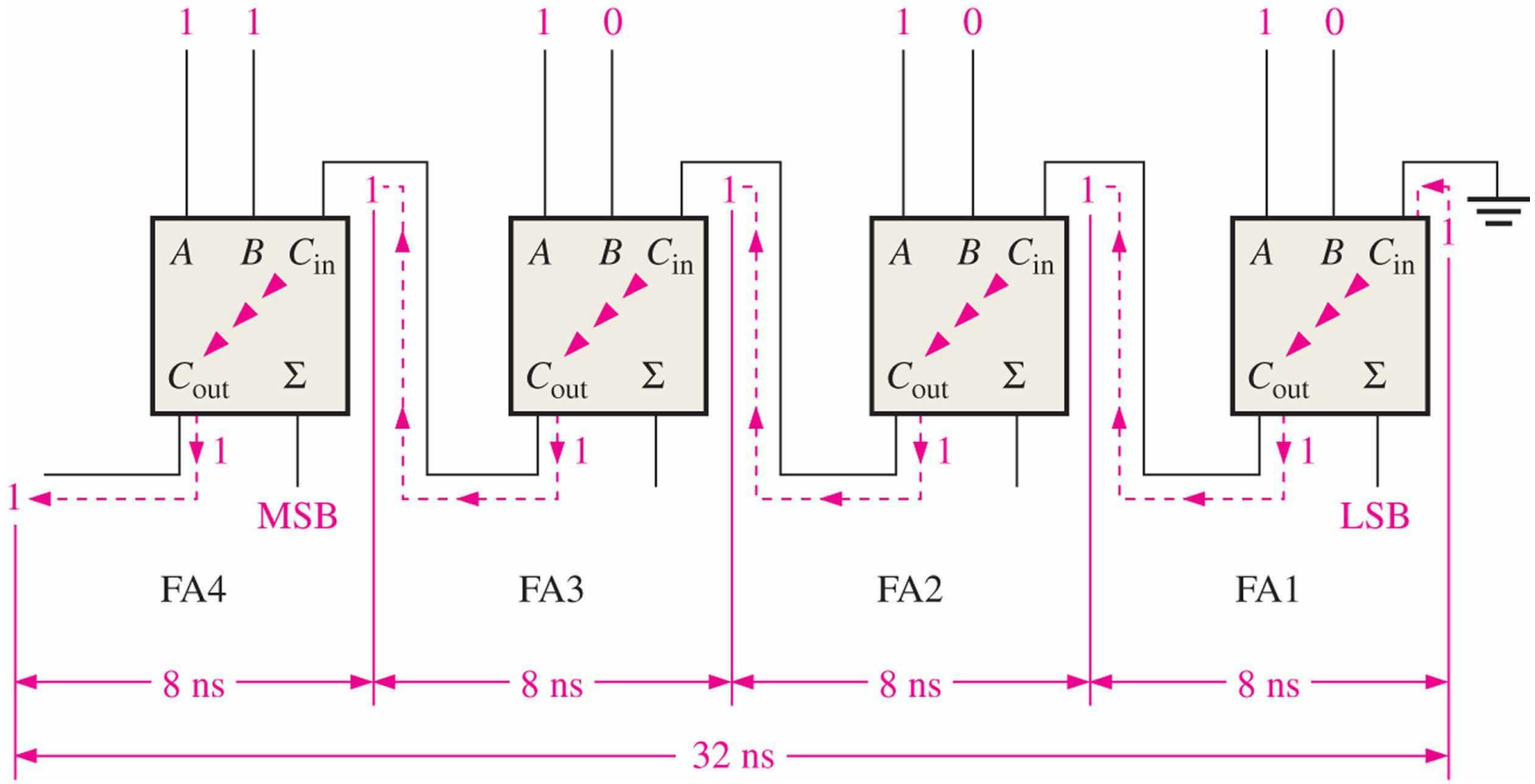
Carry propagation delay

- The addition of 2 binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same (i.e. all bits in each column perform addition simultaneously)
- However, in reality the signal (carry) must propagate through the gates before the correct output sum is available in the output terminals.
- The total propagation time is equal to the propagation delay of a typical gates times the number of gate levels in the circuit
- As the number of bits increase, the adder performance starts to suffer → The output carry at the MSB is not ready until it propagates through all of the full adders before
- This is known as the carry propagation delay → caused by the type of internal carry method used in the parallel adder

Ripple carry adder

- A ripple carry adder is one which the carry out of each FA is connected to the carry in of the next higher-order stage (a stage is one FA)
- The sum and output carry in any stage cannot be determined until the input carry is available, causing a delay in the addition process
- The carry propagation delay for each FA is the time from the application of the input carry until the output carry occurs (assuming other inputs A and B are already present) → this is multiplied by the number of FAs in a parallel adder for the total delay

- Total delay can vary with ripple carry depending on the carry bit
  - If both numbers added have no carry operations, time for addition is simple propagation time through a single adder (since all running simultaneously)
  - If carry bits occur, then worst case is the cumulative delay of each FA


1 1
1 0
1 0
1 0
A B C_{in}
C_{out} Σ
1
1
1
1
1
1
1
1
MSB
LSB
FA4
FA3
FA2
FA1
8 ns
8 ns
8 ns
8 ns
32 ns

Look-ahead carry adder

- A method to speed up the adder is by eliminating the ripple carry delay → implemented as look-ahead carry adder
- The look-ahead carry adder anticipates the output carry at each stage based on the inputs and produces the output carry either by carry generation OR carry propagation
  - Carry generation – occurs when output carry is produced/generated internally by FA. Only generated when both inputs are 1s and is expressed as an AND function of the inputs

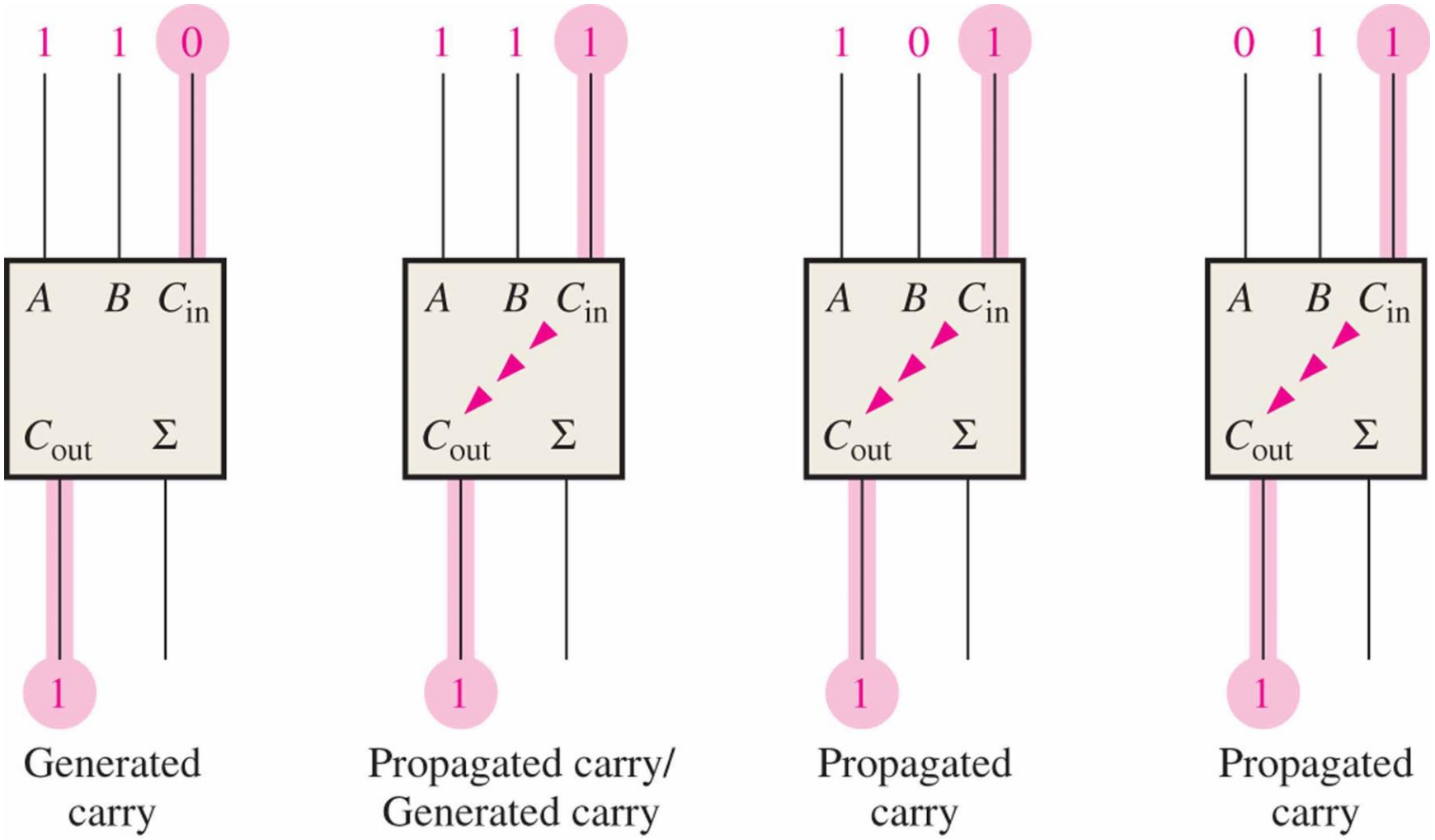
$$C_g = AB$$

  - Carry propagation – occurs when input carry is rippled to the output carry. Propagation occurs in FA when either/both inputs are 1s and is expressed as an OR function of the inputs

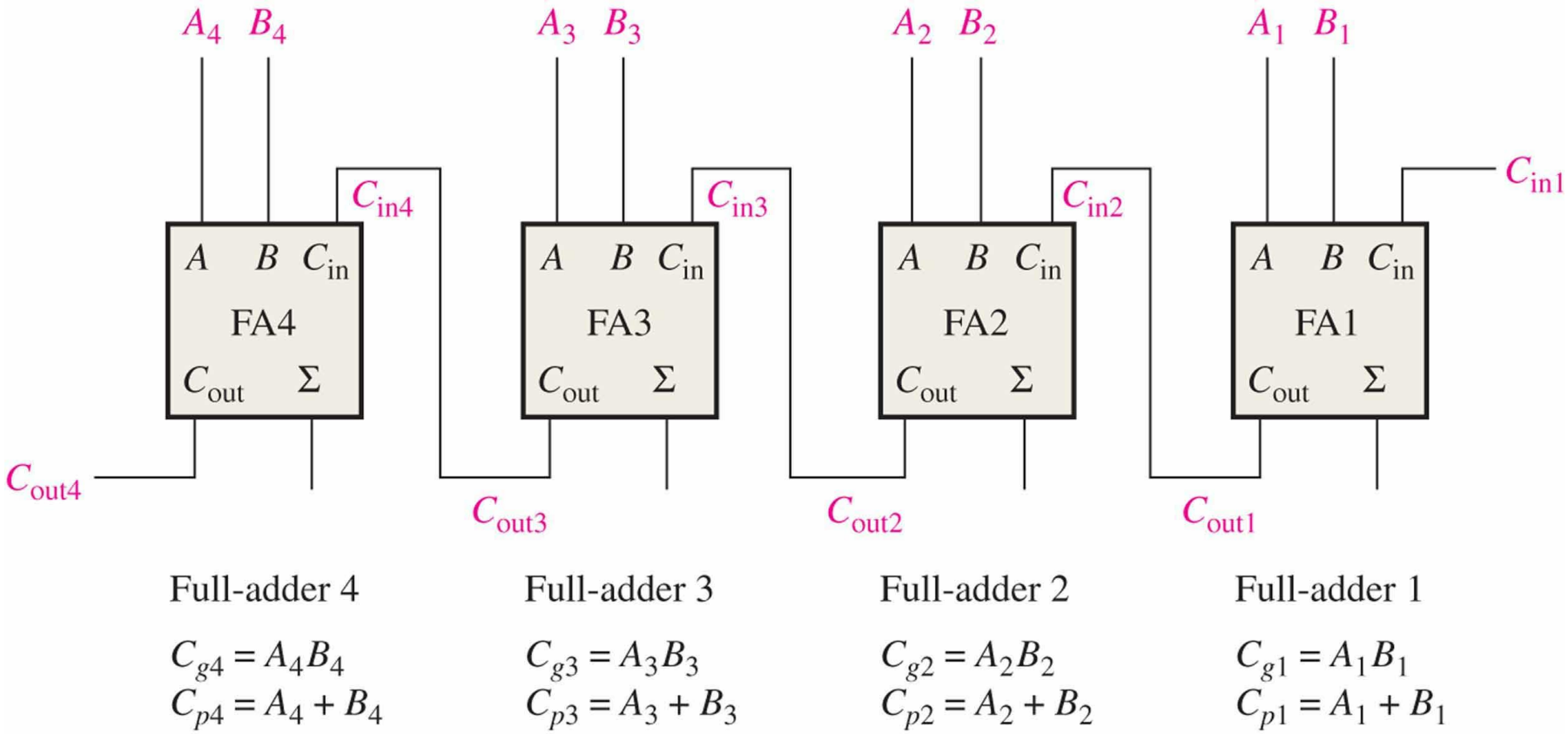
$$C_p = A + B$$

  - The output carry of a FA can thus be expressed in terms of the generated and propagated carry

$$C_{out} = C_g + C_p C_{in}$$


1 1 0
A B C_{in}
C_{out} Σ
1
Generated carry
1 1 1
A B C_{in}
C_{out} Σ
1
Propagated carry/ Generated carry
1 0 1
A B C_{in}
C_{out} Σ
1
Propagated carry
0 1 1
A B C_{in}
C_{out} Σ
1
Propagated carry

- Applying the generated and propagated carry to a normal FA



- Each FA stage immediately has the value of the generated and propagated queries since it relies solely on respective inputs A and B to the FA
- The only remaining thing is the C_{in} to each FA

- Further developing the each FA stage using Boolean analysis
  - FA1

$$C_{out1} = C_{g1} + C_{p1}C_{in1}$$

  - FA2

$$C_{out2} = C_{g2} + C_{p2}C_{in2} = C_{g2} + C_{p2}C_{out1}$$
$$= C_{g2} + C_{p2}\left(C_{g1} + C_{p1}C_{in1}\right) = C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}$$

  - FA3

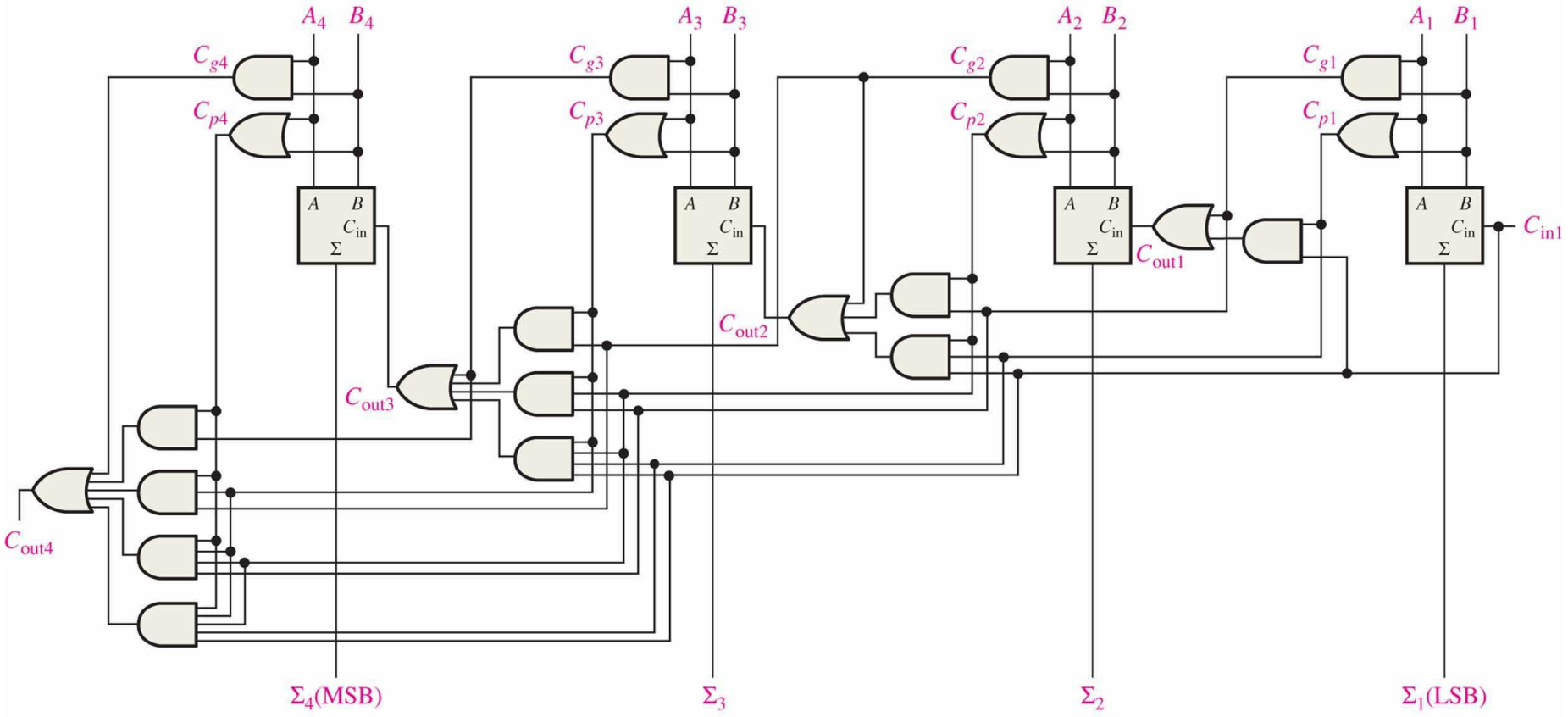
$$C_{out3} = C_{g3} + C_{p3}C_{in3} = C_{g3} + C_{p3}C_{out2}$$
$$= C_{g3} + C_{p3}\left(C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}\right)$$
$$= C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}$$

  - FA4

$$C_{out4} = C_{g4} + C_{p4}C_{in4} = C_{g4} + C_{p4}C_{out3}$$
$$= C_{g4} + C_{p4}\left(C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}\right)$$
$$= C_{g4} + C_{p4}C_{g3} + C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1} + C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$$

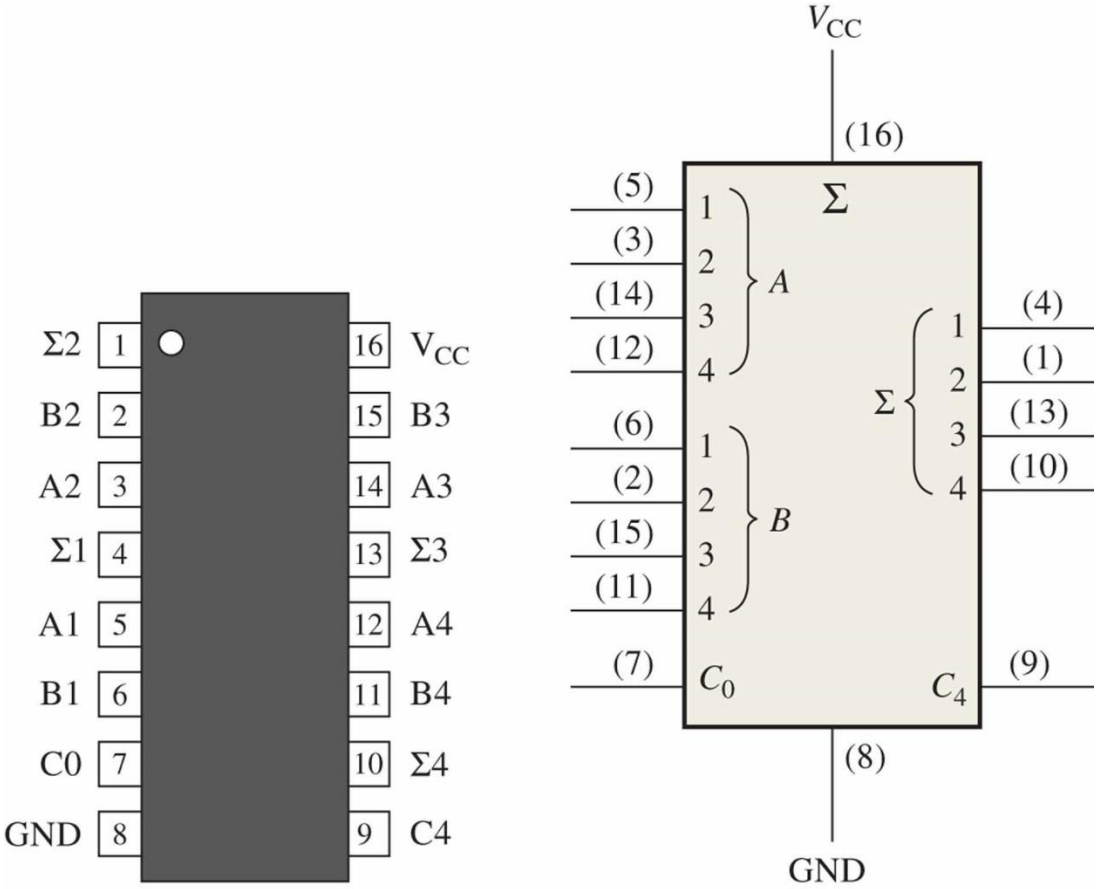
- In each FA stage, notice that the carry output is dependent only on C_{in1}, C_g and C_p of the stage and C_g and C_p of preceding stages → all of which can be expressed as inputs A and B to each stage thus there is no need to wait for ripple carry

- A fully implemented look-ahead parallel nibble adder



Nibble adder as FLD

- The 74LS283 is an example of FLD which features look-ahead carry → adds logic to minimize the output carry delay.



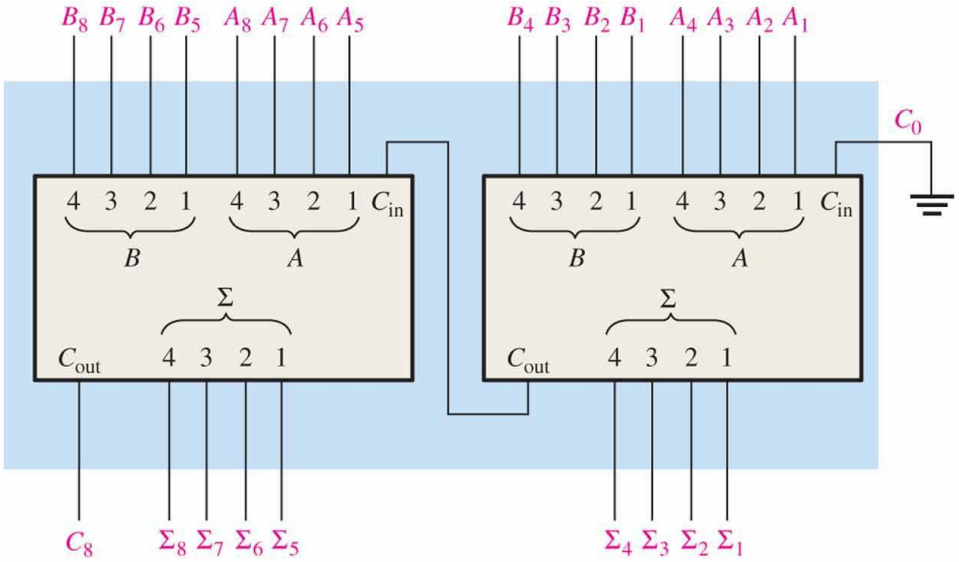
(a) Pin diagram of 74LS283

(b) 74LS283 logic symbol

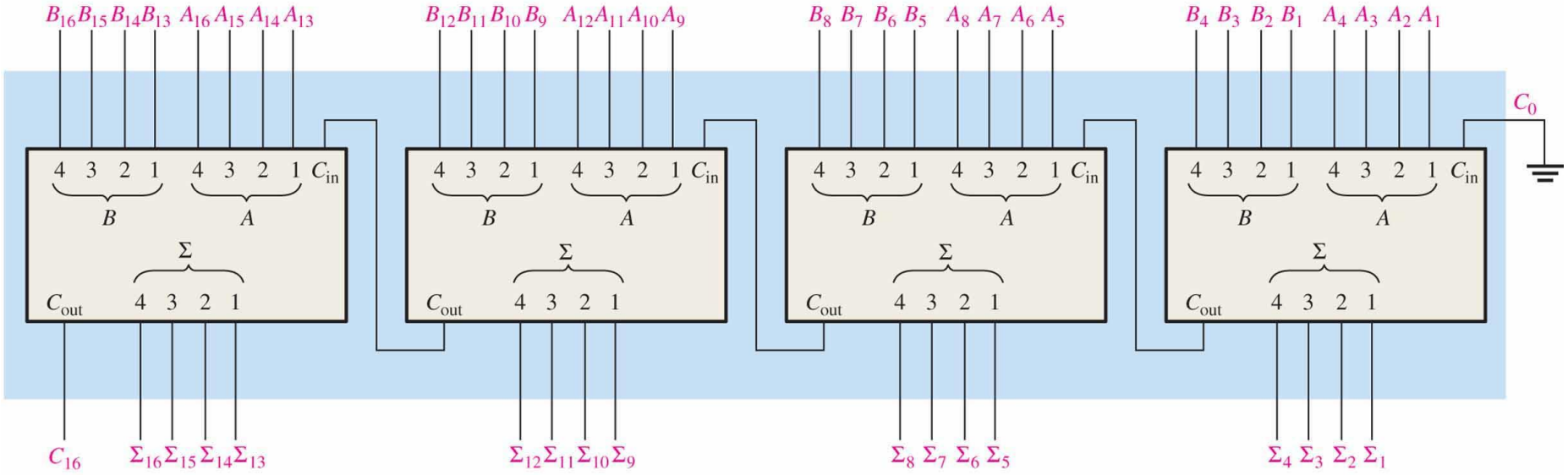
Adder expansion

- Nibble adders can be expanded to handle addition of two 8-bit numbers simply by using two adders
- Carry input of lower-order adder is grounded whilst the higher-order adder has its carry in connected to the carry out of the low one → this is called cascading
- Process can be repeated to create 12-, 16-bit etc adders

- Note that combining multiple parallel adders (e.g. four 74LS283 ICs to create a 16-bit parallel adder) will result in an adder which is a combination look-ahead and ripple carry
  - Each parallel adder will be using look-ahead
  - Carries between parallel adders require ripple method instead



(a) Cascading of two 4-bit adders to form an 8-bit adder



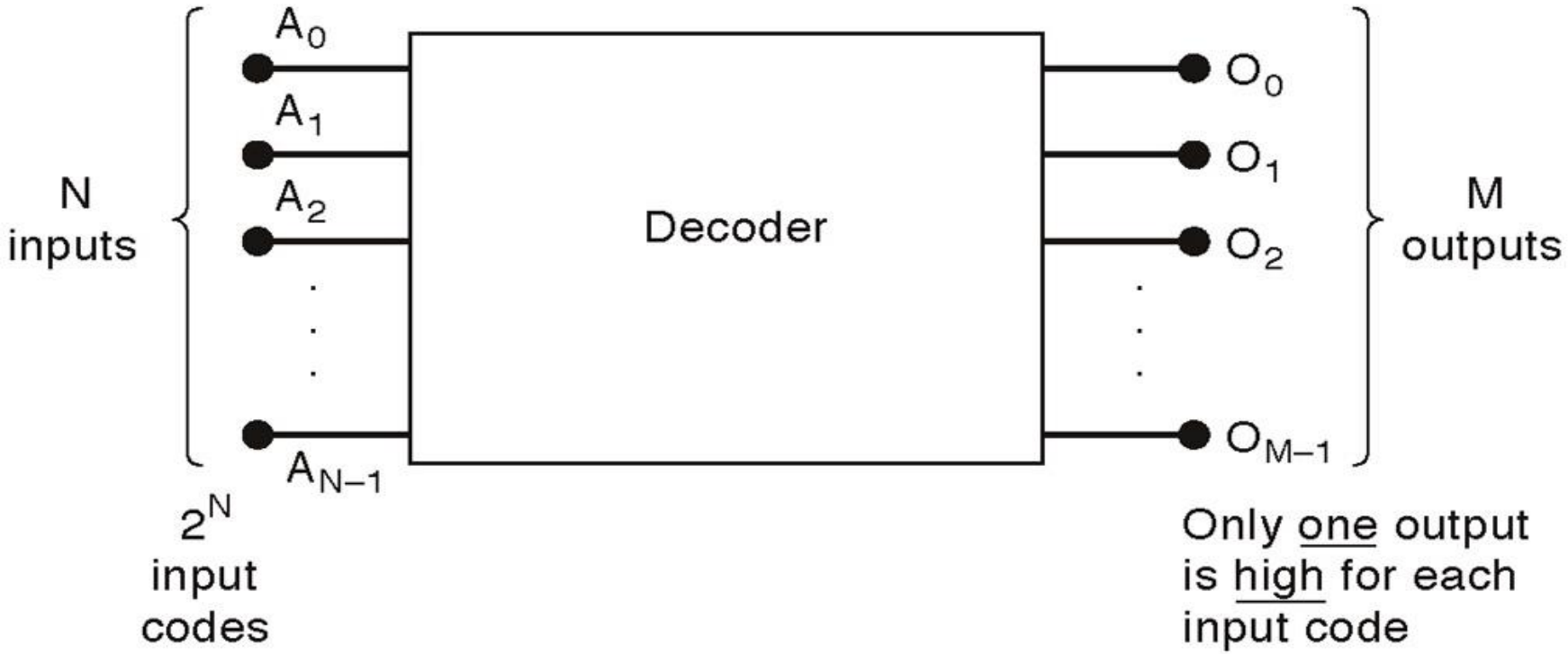
(b) Cascading of four 4-bit adders to form a 16-bit adder

Decoder

- Digital circuit that detects presence of specific combination of bits (code) on its inputs by indicating on a specified output level
- A decoder has *n* input lines to handle *n* bits and one to 2^n output lines to indicate presence of one or more *n*-bit combinations
- There are many types of decoders but in principle all work very similar to each other → detect bit combination and generate corresponding output

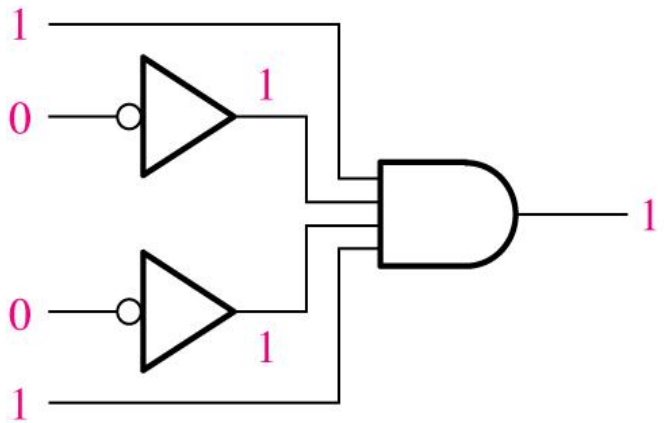
General decoder diagram

- There are 2^N possible input combinations, from A_0 to A_{N-1}.
- For each of these input combinations only one of the M outputs will be active HIGH (1), all the other outputs are LOW (0).

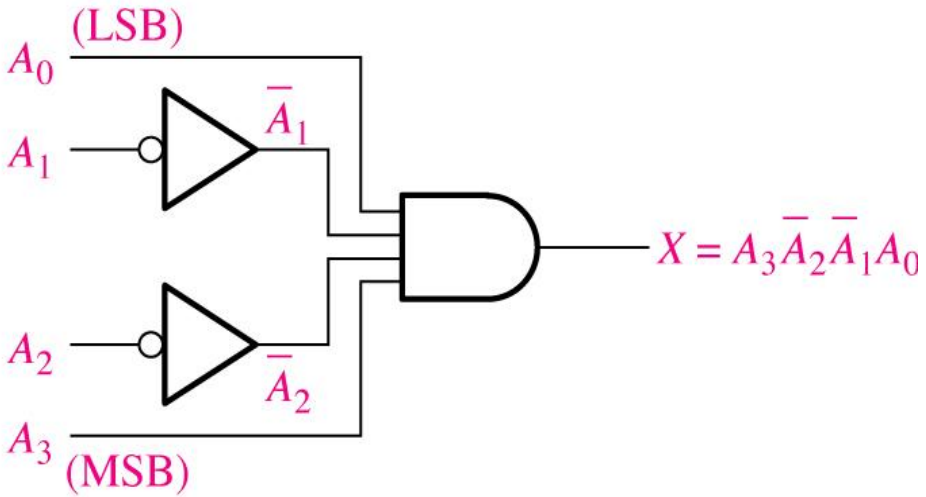


Basic binary decoder

- A single logic gate can be used to create a basic binary decoder. E.g.
  - Suppose you need to determine when a binary 1001 occurs on the input lines of a digital circuit
  - Easiest to use 4-input AND gate → outputs HIGH when all inputs are HIGH
  - To make sure all the inputs to the AND gate are high when the binary 1001 occurs, an inverted can be used to invert the middle bits (so they always remain inverted to the LSB and MSB bits)



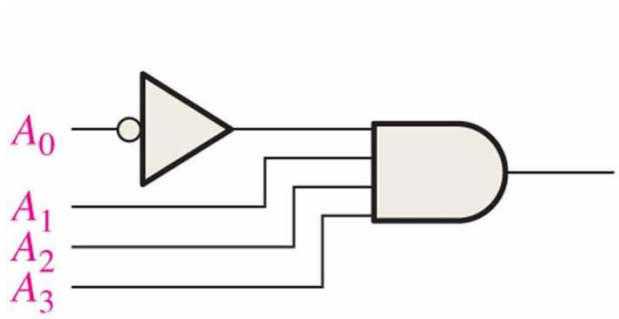
(a)



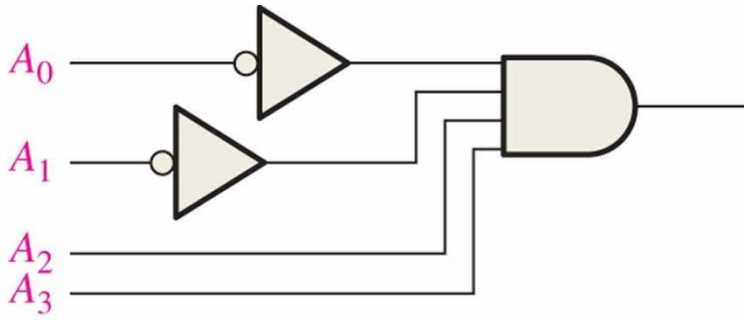
(b)

Exercise

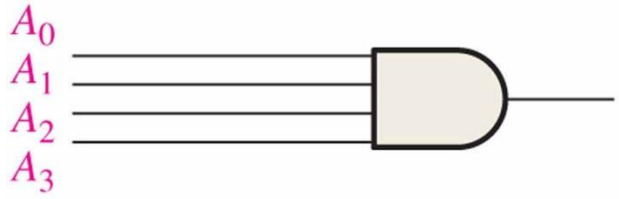
- Describe the following basic decoders in terms of when they are active (HIGH output) and the binary input that triggers this condition assuming that A_3 is MSB



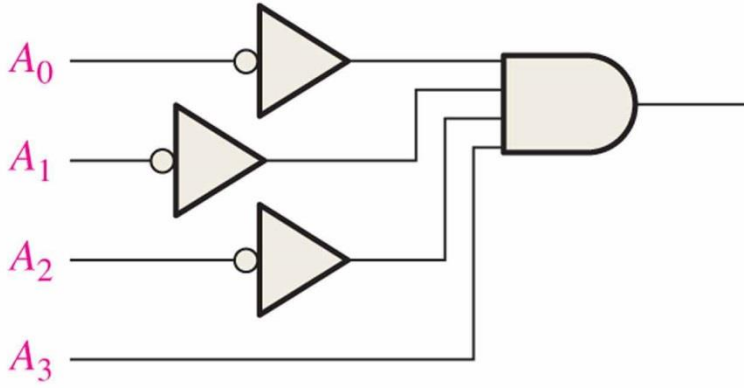
(a)



(b)



(c)

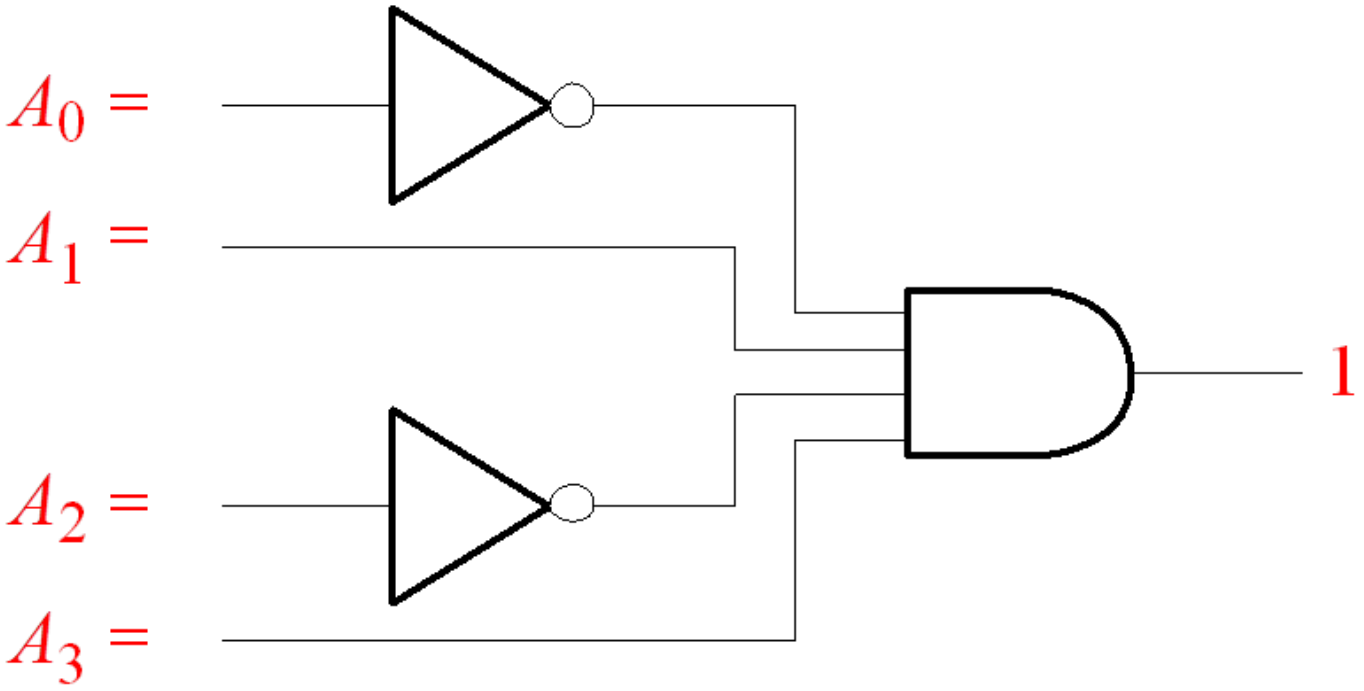


(a) $A_3A_2A_1A_0 = \mathbf{1110}$
(c) $A_3A_2A_1A_0 = \mathbf{1111}$

(b) $A_3A_2A_1A_0 = \mathbf{1100}$
(d) $A_3A_2A_1A_0 = \mathbf{1000}$

Exercise

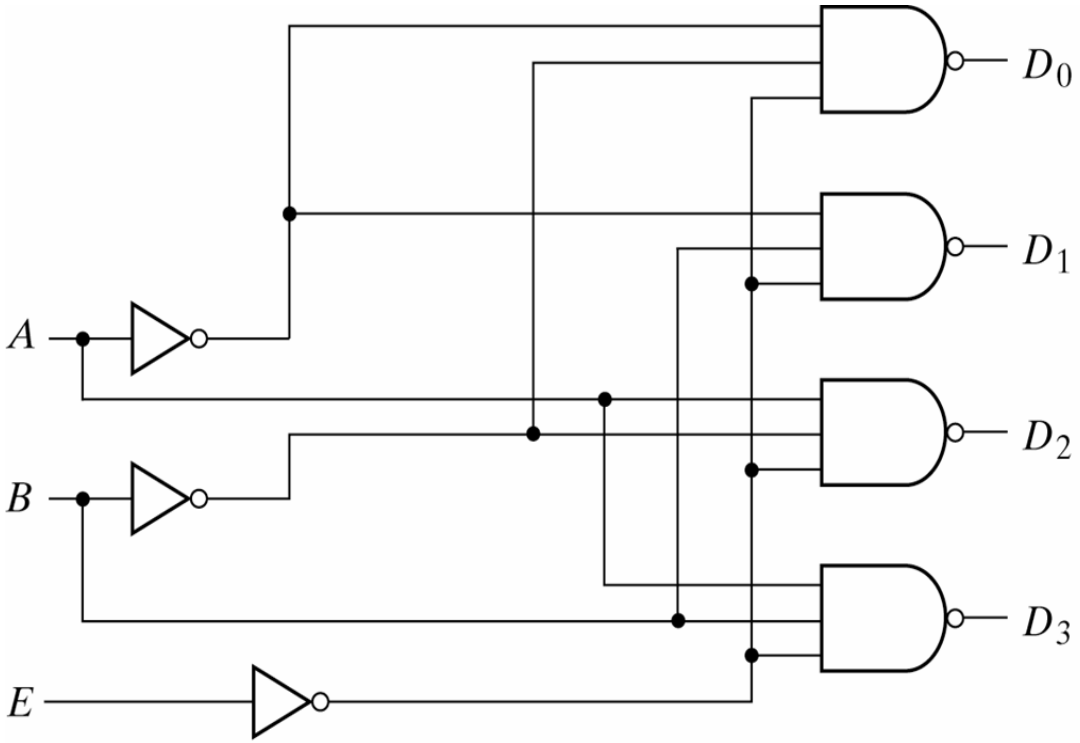
- Assuming the output of the decoder shown is a logic 1 (HIGH), what are the inputs to the decoder?



$A_3A_2A_1A_0=1010$

2-bit decoder

- Create a decoder which selects one of four possible outputs based on two input values.
- Example decoder here implements active LOW output.
- Possible decoder:
  - The circuit operates with complemented outputs and a complement enable input.
  - The decoder is enabled when E is equal to 0.
  - Only one output can be equal to 0 at any given time, all other outputs are equal to 1.
  - The output whose value is equal to 0 represents the minterm selected by inputs A and B
  - The circuit is disable when E is equal to 1.

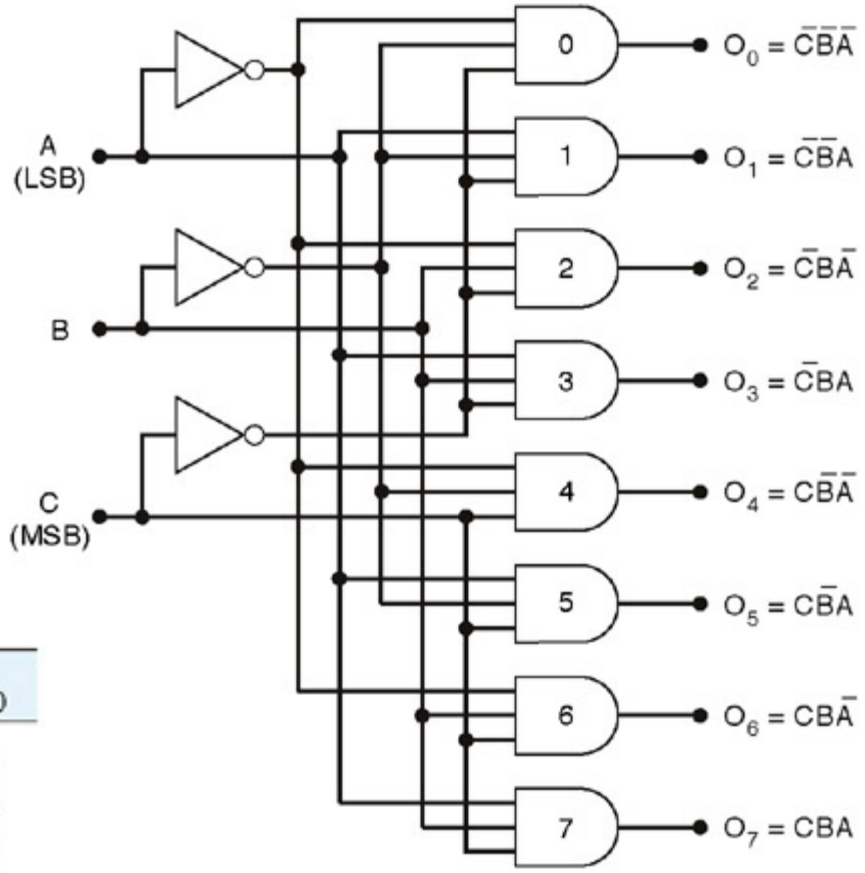


(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

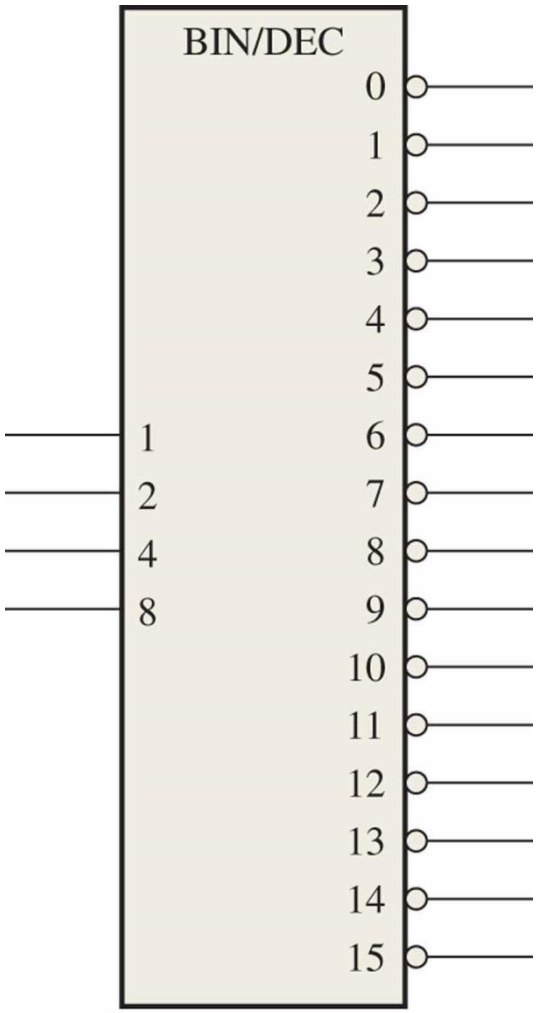
3-bit decoder (active HIGH)



C	B	A	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

4-bit decoder

- 4-bit decoder is called 4-line-to-16-line decoder or 1-of-16 decoder because there four inputs and sixteen outputs.
- For any given code on the inputs, one of the sixteen outputs is activated



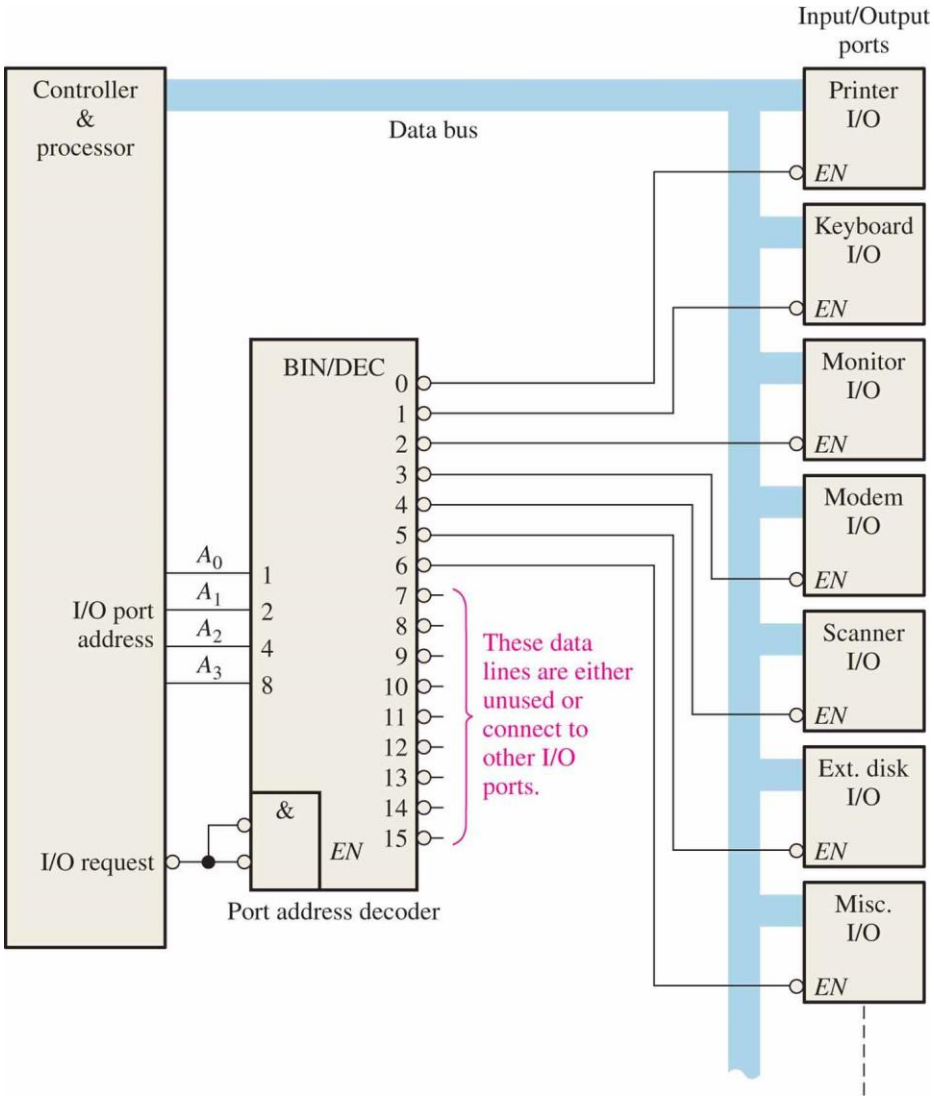
4-bit decoder truth table

- Devise the truth table for the 4-bit decoder if given that the decoder is active HIGH

DECIMAL DIGIT	BINARY INPUTS A_3	A_2	A_1	A_0	DECODING FUNCTION	OUTPUTS 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\bar{A}_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\bar{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

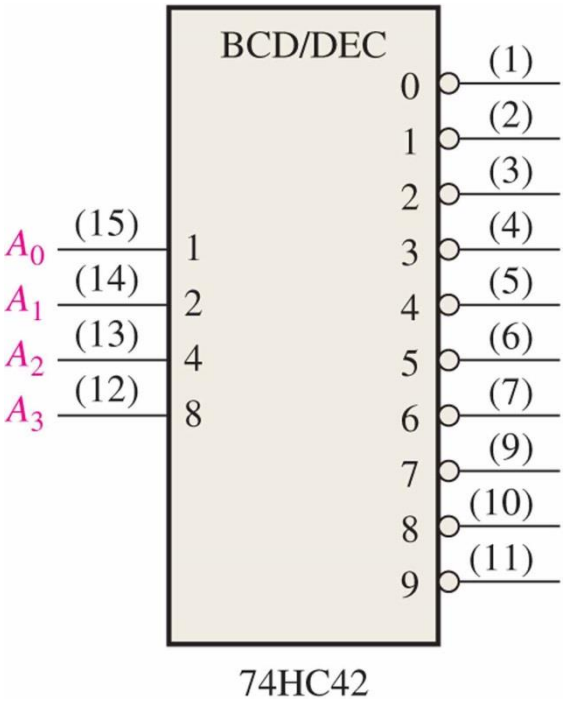
Decoder application example

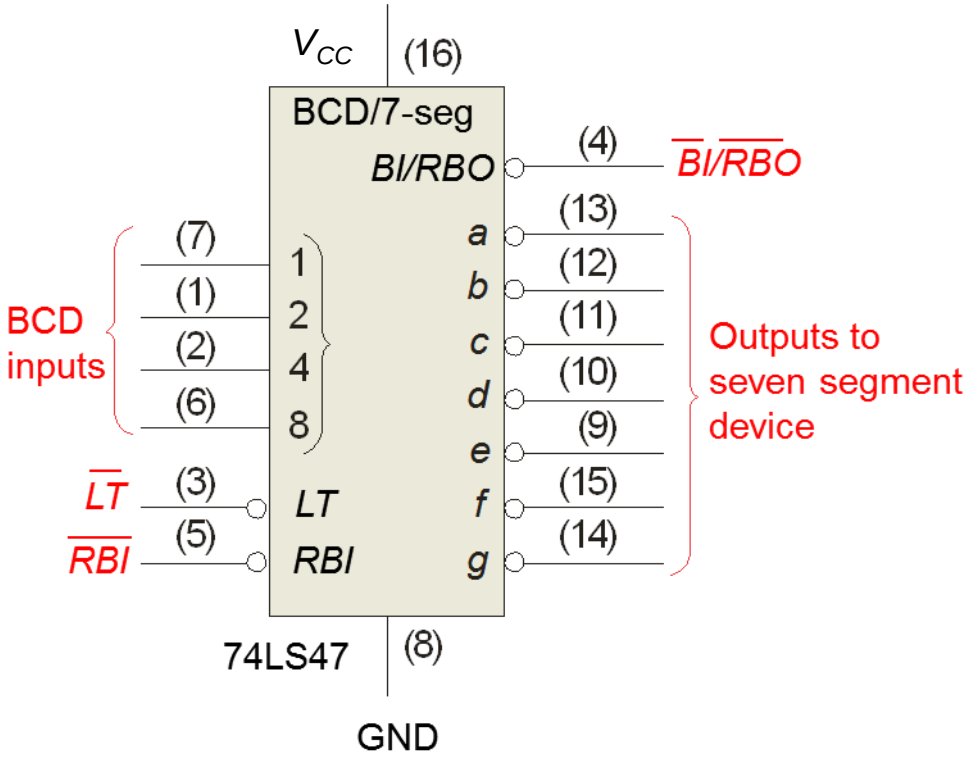
- Common use of decoder is I/O selection – computers communicate with multiple external devices that communicate through I/O ports
- In a simple arrangement all devices share the data bus which is a set or parallel lines sending/receiving data
- To indicate which device a computer wants to communicate with, it uses the I/O port number → this number is *decoded* to activate (i.e. enable) the corresponding device whilst the others remain silent



Other typical decoders

- BCD-to-decimal (4-line-to-10-line or 1-of-10) decoder
- BCD-to-7-segment decoder





Summary

- Adders implement important functions in a modern CPU
- Ripple carry adders are simple to implement in logic but slow in process
- Look-ahead carry adders eliminate ripple delays in parallel adders
- Parallel adders can be combine to create adders that can support multiple bits
- Decoders detect specific bit patterns and creates output based on the detected input