ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

COMPUTER ORGANISATION
(TỔ CHỨC MÁY TÍNH)

Combinational Circuits

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

WHERE ARE WE NOW?

- Number systems and codes
- Boolean algebra
- Logic gates and circuits
- Simplification
- Combinational circuits
- Sequential circuits
- Performance
- Assembly language
- The processor: Datapath and control
- Pipelining
- Memory hierarchy: Cache
- Input/output

Preparation: 2 weeks

Logic Design: 3 weeks

Computer organisation

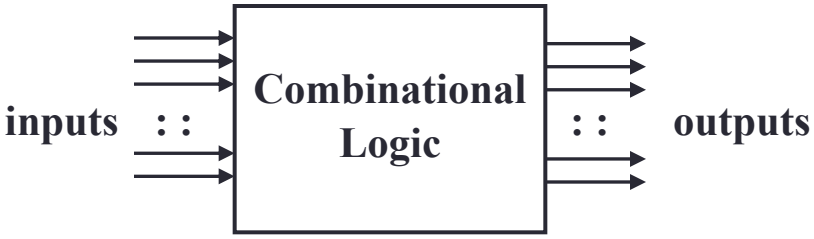
COMBINATIONAL CIRCUITS

- Introduction
- Analysis Procedure
- Design Methods
- Gate-level (SSI) Design
- Block-Level Design
- Arithmetic Circuits
- Circuit Delays
- Look-Ahead Carry Adder

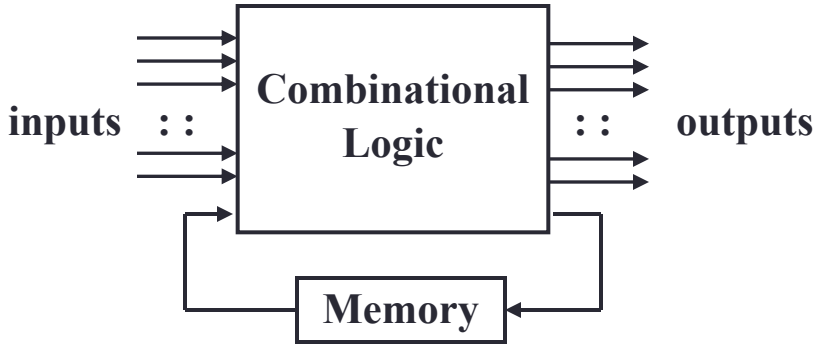
INTRODUCTION

- Two classes of logic circuits
  - Combinational
  - Sequential

- Combinational Circuit
  - Each output depends entirely on the immediate (present) inputs.

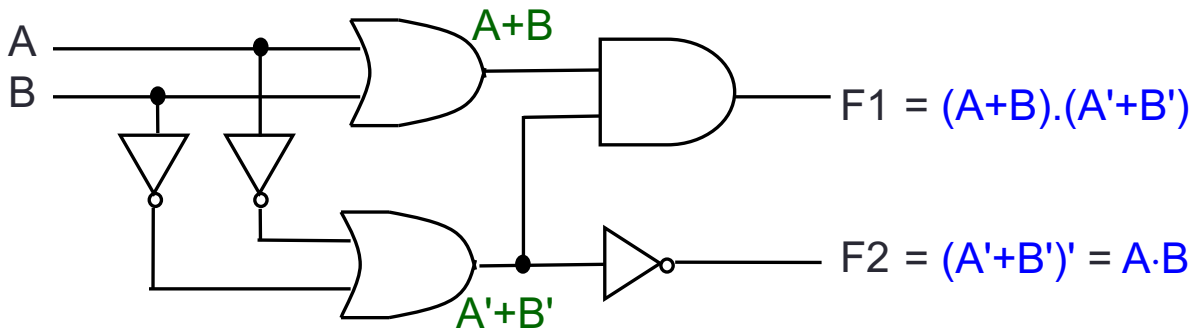


- Sequential Circuit
  - Each output depends on both present inputs and state.



ANALYSIS PROCEDURE

- Given a combinational circuit, how do you analyze its function?



- Steps:
  1. Label the inputs and outputs.
  2. Obtain the functions of intermediate points and the outputs.
  3. Draw the truth table.
  4. Deduce the functionality of the circuit ➲

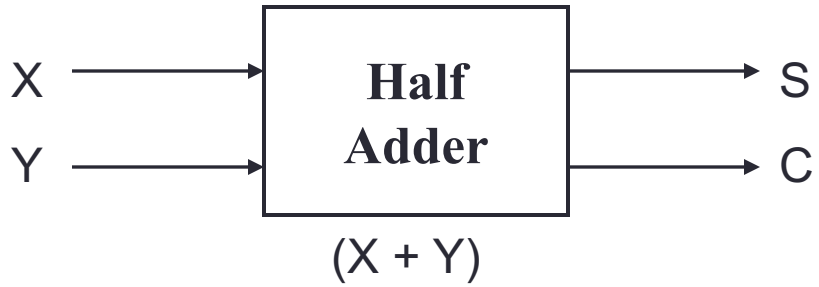
A	B	(A+B)	(A'+B')	F1	F2
0	0	0	1	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	1	0	0	1

DESIGN METHODS

- Different combinational circuit design methods:
  - Gate-level design method (with logic gates)
  - Block-level design method (with functional blocks)
- Design methods make use of logic gates and useful function blocks
  - These are available as Integrated Circuit (IC) chips.
  - Types of IC chips (based on packing density): SSI, MSI, LSI, VLSI, ULSI.
- Main objectives of circuit design:
  - Reduce cost (number of gates for small circuits; number of IC packages for complex circuits)
  - Increase speed
  - Design simplicity (re-use blocks where possible)

GATE-LEVEL (SSI) DESIGN: HALF ADDER (1/2)

- Design procedure:
  1. State problem
     Example: Build a Half Adder.
  2. Determine and label the inputs and outputs of circuit.
     Example: Two inputs and two outputs labelled, as shown below.



X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

3. Draw the truth table.

GATE-LEVEL (SSI) DESIGN: HALF ADDER (2/2)

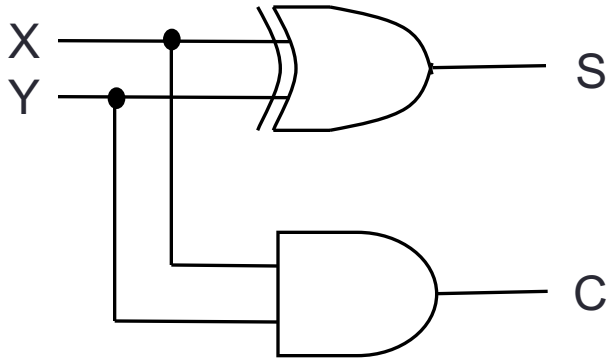
4. Obtain simplified Boolean functions.

   Example: $C = X \cdot Y$

   $S = X' \cdot Y + X \cdot Y' = X \oplus Y$

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

5. Draw logic diagram.



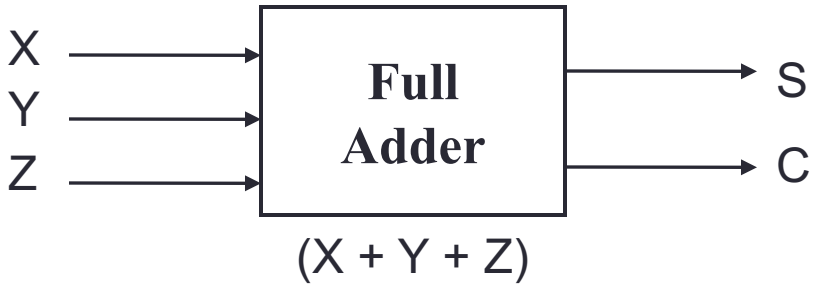
Half Adder

GATE-LEVEL (SSI) DESIGN: FULL ADDER (1/5)

- Half adder adds up only two bits.
- To add two binary numbers, we need to add 3 bits (including the *carry*).
  - Example:

	1	1	1		carry
	0	0	1	1	X
+	0	1	1	1	Y
	1	0	1	0	S

- Need Full Adder (so called as it can be made from two half adders).



GATE-LEVEL (SSI) DESIGN: FULL ADDER (2/5)

- Truth table:

X	Y	Z	C	S
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:

Z - carry in (to the current position)

C - carry out (to the next position)

- Using K-map, simplified SOP form:

C = ?

S = ?

C

X \ YZ	00	01	11	10
0			1	
1		1	1	1

S

X \ YZ	00	01	11	10
0		1		1
1	1		1	

GATE-LEVEL (SSI) DESIGN: FULL ADDER (3/5)

- Alternative formulae using algebraic manipulation:

$$
\begin{aligned}
C &= X\cdot Y + X\cdot Z + Y\cdot Z \\
&= X\cdot Y + (X + Y)\cdot Z \\
&= X\cdot Y + ((X\oplus Y) + X\cdot Y)\cdot Z \\
&= X\cdot Y + (X\oplus Y)\cdot Z + X\cdot Y\cdot Z \\
&= X\cdot Y + (X\oplus Y)\cdot Z
\end{aligned}
$$

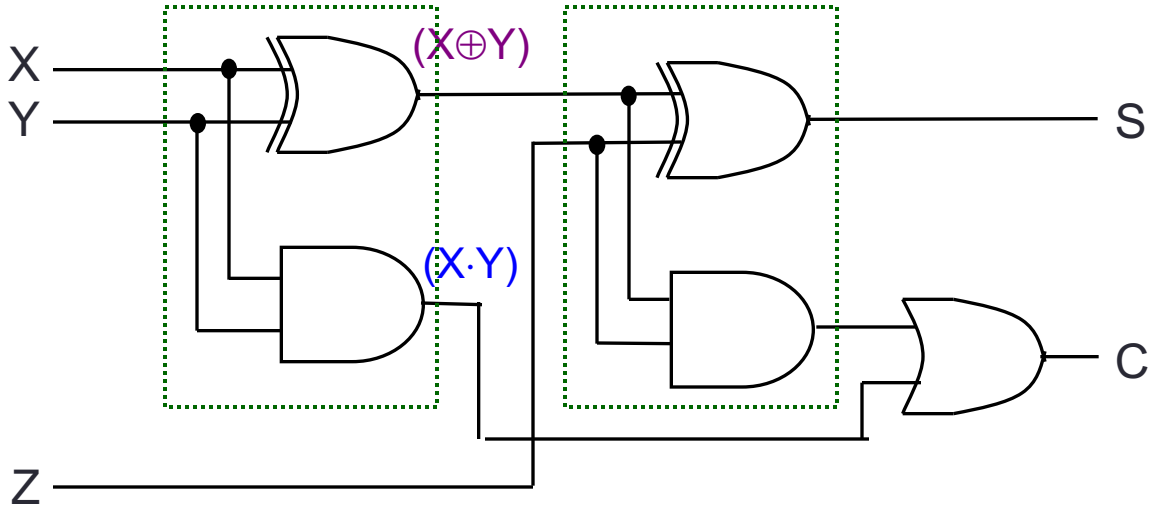
$$
\begin{aligned}
S &= X'\cdot Y'\cdot Z + X'\cdot Y\cdot Z' + X\cdot Y'\cdot Z' + X\cdot Y\cdot Z \\
&= X'\cdot(Y'\cdot Z + Y\cdot Z') + X\cdot(Y'\cdot Z' + Y\cdot Z) \\
&= X'\cdot(Y\oplus Z) + X\cdot(Y\oplus Z)' \\
&= X\oplus(Y\oplus Z)
\end{aligned}
$$

GATE-LEVEL (SSI) DESIGN: FULL ADDER (4/5)

- Circuit for above formulae:

  C = X·Y + (X⊕Y)·Z

  S = X⊕(Y⊕Z) = (X⊕Y)⊕Z (XOR is associative)



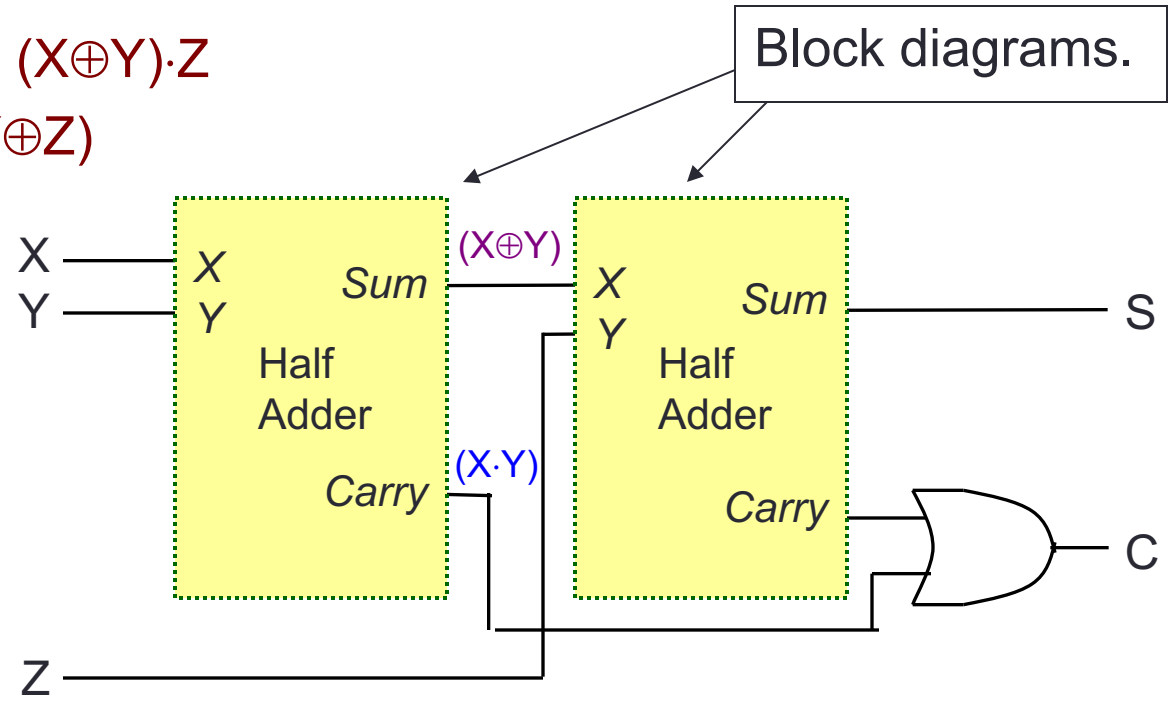
Full Adder made from two Half-Adders (+ an OR gate).

GATE-LEVEL (SSI) DESIGN: FULL ADDER (5/5)

- Circuit for above formulae:

  C = X·Y + (X⊕Y)·Z

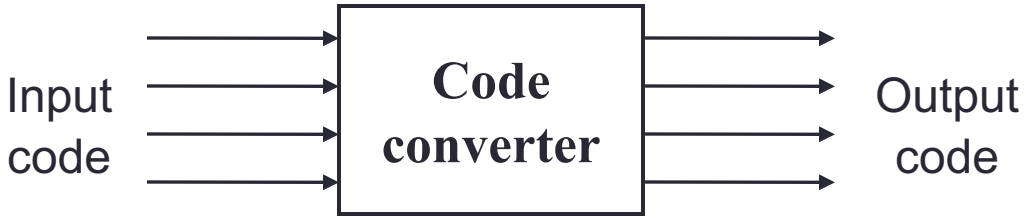
  S = X⊕(Y⊕Z)



Full Adder made from two Half-Adders (+ an OR gate).

CODE CONVERTERS

- Code converter – takes an input code, translates to its equivalent output code.



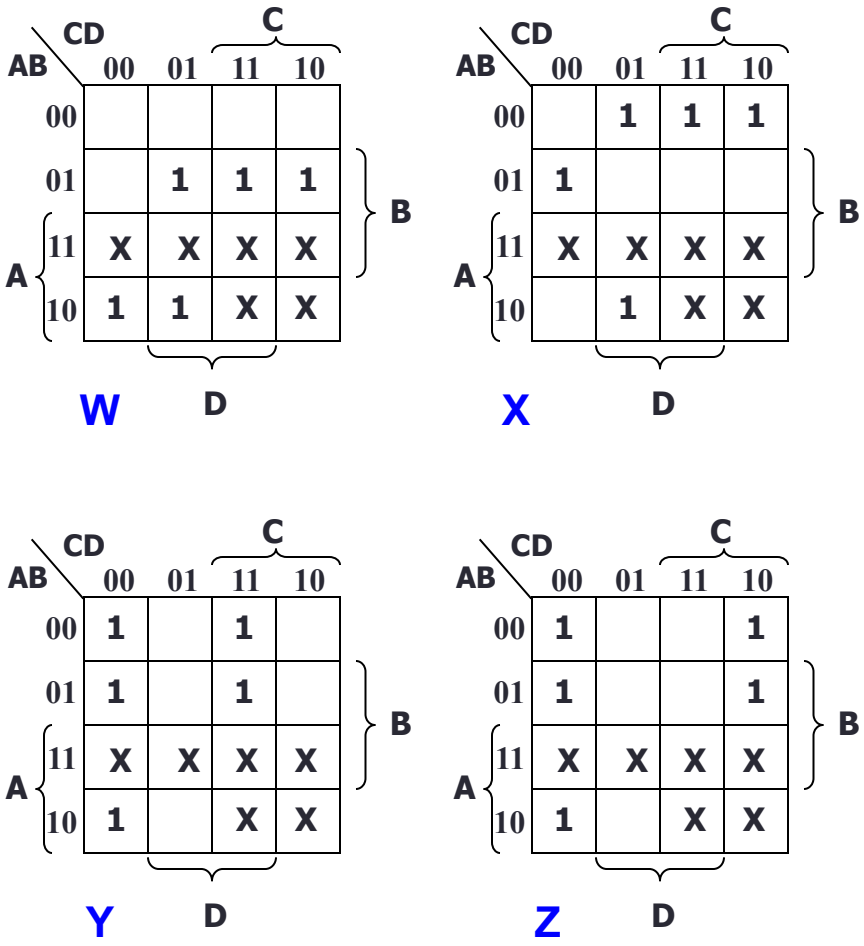
- Example: BCD to Excess-3 code converter.
  - Input: BCD code
  - Output: Excess-3 code

BCD-TO-EXCESS-3 CONVERTER (1/2)

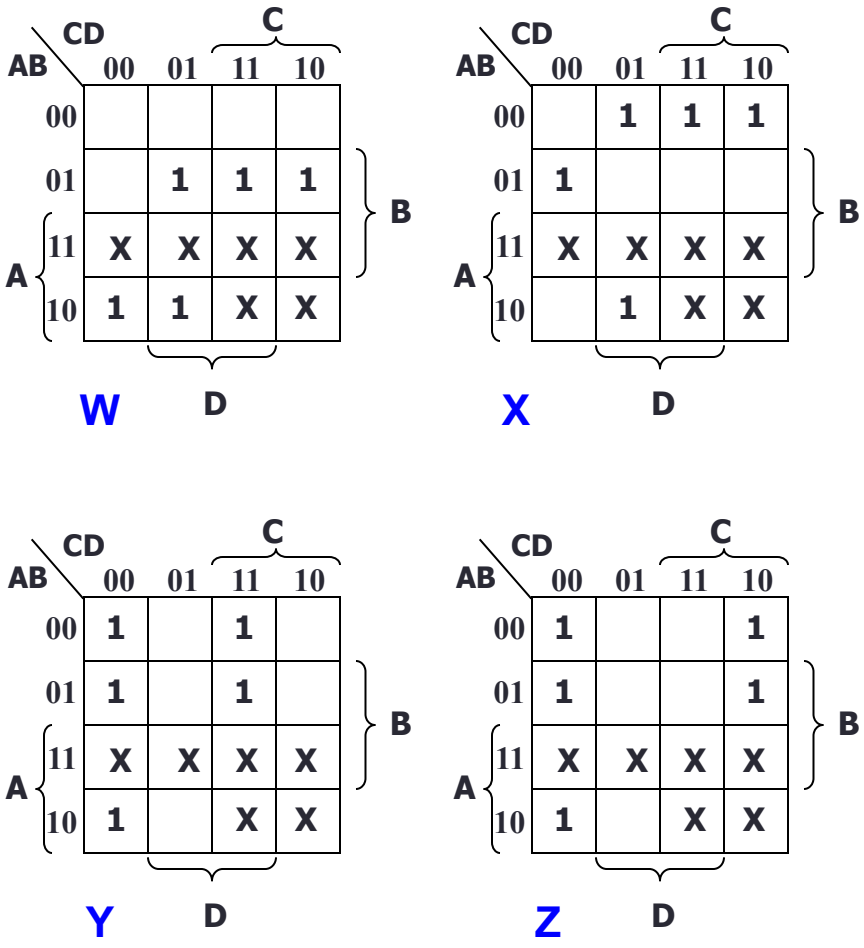
- Truth table:

	BCD				Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X

- K-maps:



BCD-TO-EXCESS-3 CONVERTER (2/2)



W = ?

X = ?

Y = ?

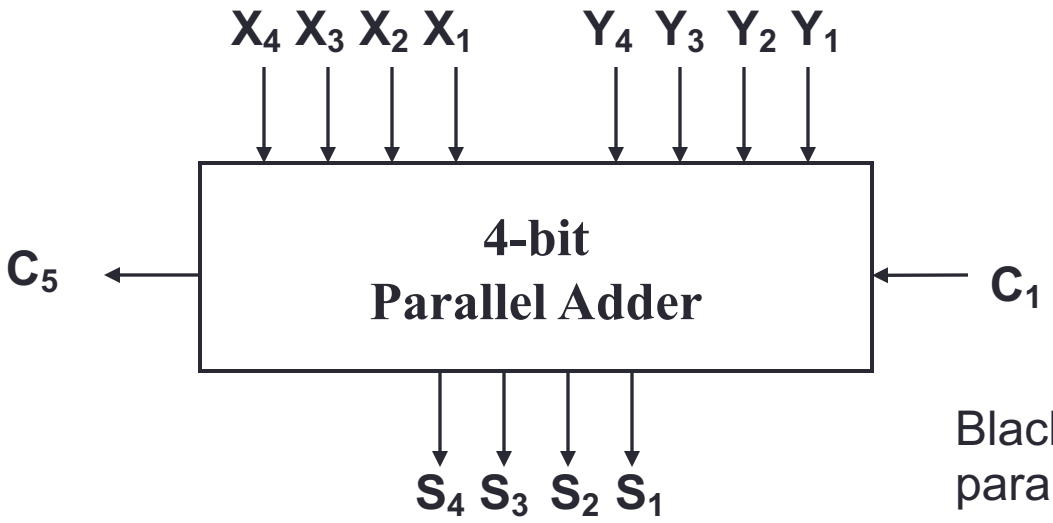
Z = ?

BLOCK-LEVEL DESIGN

- More complex circuits can also be built using block-level method.
- In general, block-level design method (as opposed to gate-level design) relies on algorithms or formulae of the circuit, which are obtained by decomposing the main problem to sub-problems recursively (until small enough to be directly solved by blocks of circuits).
- Simple examples using 4-bit parallel adder as building blocks:
  1. BCD-to-Excess-3 Code Converter
  2. 16-bit Parallel Adder
  3. Adder cum Subtractor

4-BIT PARALLEL ADDER (1/4)

- Consider a circuit to add two 4-bit numbers together and a carry-in, to produce a 5-bit result.



Black-box view of 4-bit parallel adder

- 5-bit result is sufficient because the largest result is:

$$1111_2 + 1111_2 + 1_2 = 11111_2$$

4-BIT PARALLEL ADDER (2/4)

- SSI design technique should not be used here.
- Truth table for 9 inputs is too big: $2^9 = 512$ rows!

$X_4X_3X_2X_1$	$Y_4Y_3Y_2Y_1$	C_1	C_5	$S_4S_3S_2S_1$
0 0 0 0	0 0 0 0	0	0	0 0 0 0
0 0 0 0	0 0 0 0	1	0	0 0 0 1
0 0 0 0	0 0 0 1	0	0	0 0 0 1
…	…	…	…	…
0 1 0 1	1 1 0 1	1	1	0 0 1 1
…	…	…	…	…
1 1 1 1	1 1 1 1	1	1	1 1 1 1

- Simplification becomes too complicated!

4-BIT PARALLEL ADDER (3/4)

- Alternative design possible.
- Addition formula for each pair of bits (with carry in),

$$C_{i+1}S_i = X_i + Y_i + C_i$$

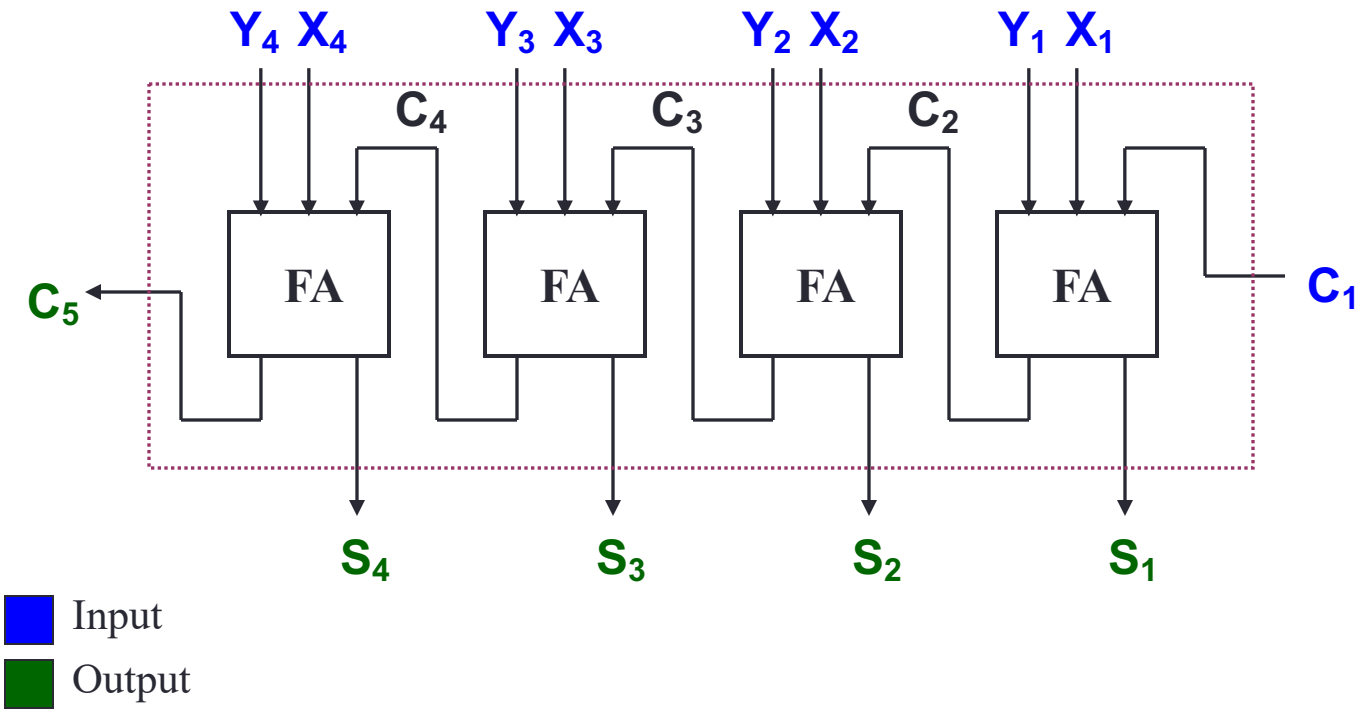
has the same function as a full adder:

$$C_{i+1} = X_i \cdot Y_i + (X_i \oplus Y_i) \cdot C_i$$
$$S_i = X_i \oplus Y_i \oplus C_i$$

C =	1 1 0 0
X =	1 0 1 0
Y =	1 1 1 1
X + Y =	1 1 0 0 1

4-BIT PARALLEL ADDER (4/4)

- Cascading 4 full adders via their carries, we get:



PARALLEL ADDERS

- Note that carry is propagated by cascading the carry from one full adder to the next.
- Called Parallel Adder because inputs are presented simultaneously (in parallel). Also called Ripple-Carry Adder.

BCD-TO-EXCESS-3 CONVERTER (1/2)

- Excess-3 code can be converted from BCD code using truth table:

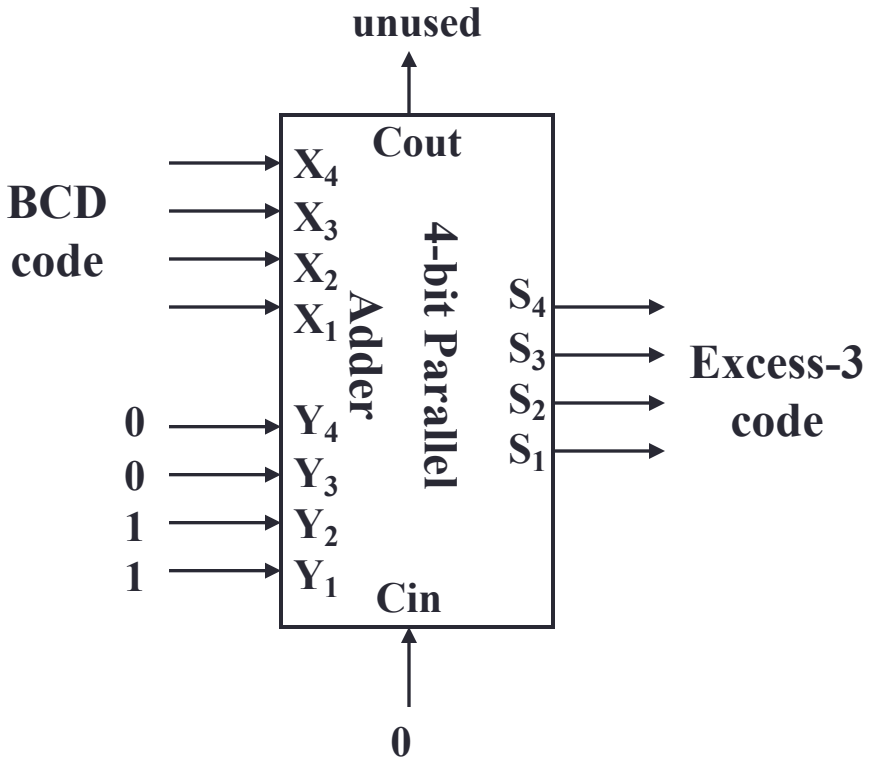
- Gate-level design can be used since only 4 inputs.
- However, alternative design is possible.
- Use problem-specific formula:

  Excess-3 code
  = BCD Code + 0011_2

	BCD				*Excess-3*			
	A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X

BCD-TO-EXCESS-3 CONVERTER (2/2)

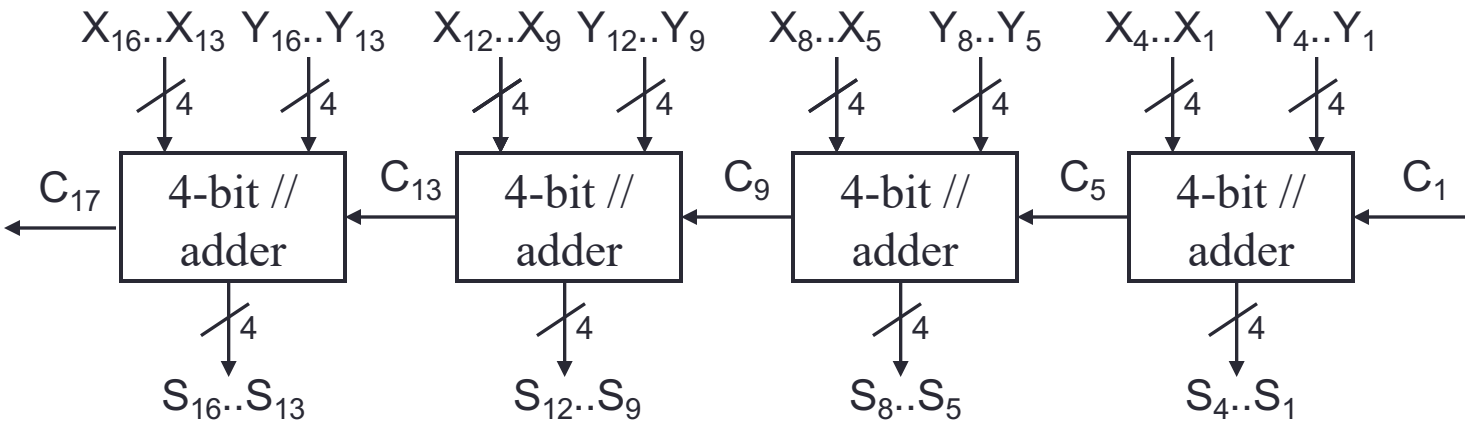
- Block-level circuit:



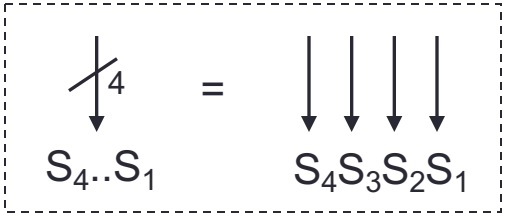
A BCD-to-Excess-3 Code Converter

16-BIT PARALLEL ADDER

- Larger parallel adders can be built from smaller ones.
- Example: A 16-bit parallel adder can be constructed from four 4-bit parallel adders:

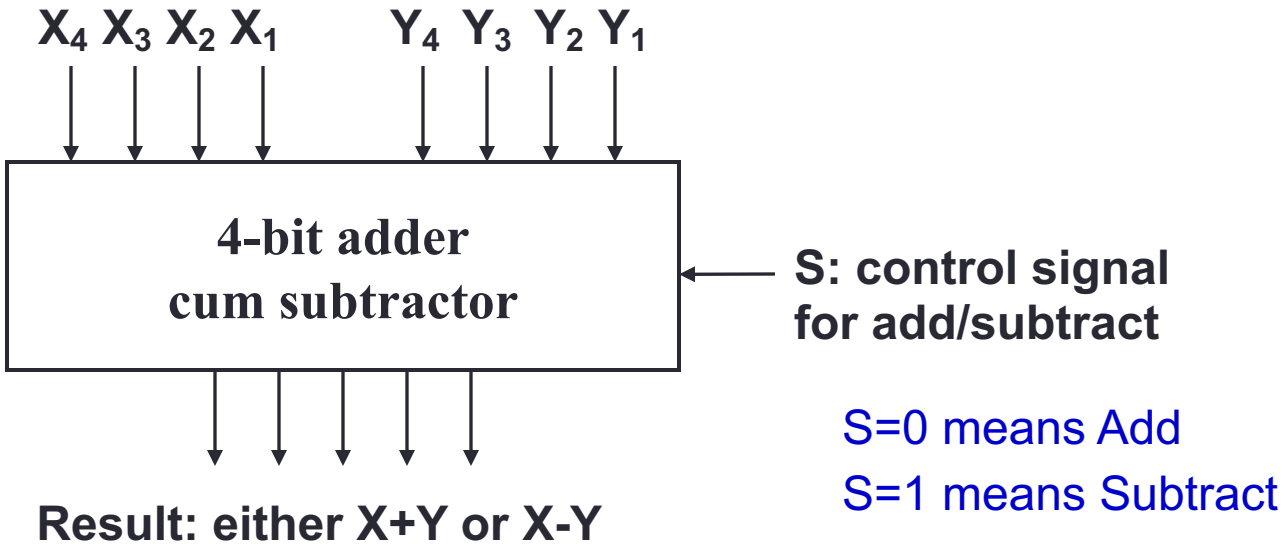


A 16-bit parallel adder



4-BIT ADDER CUM SUBTRACTOR (1/3)

- Recall: Subtraction can be done via addition with 2s-complement numbers.
- Hence, we can design a circuit to perform both addition and subtraction, using a parallel adder and some gates.



4-BIT ADDER CUM SUBTRACTOR (2/3)

- Recall:

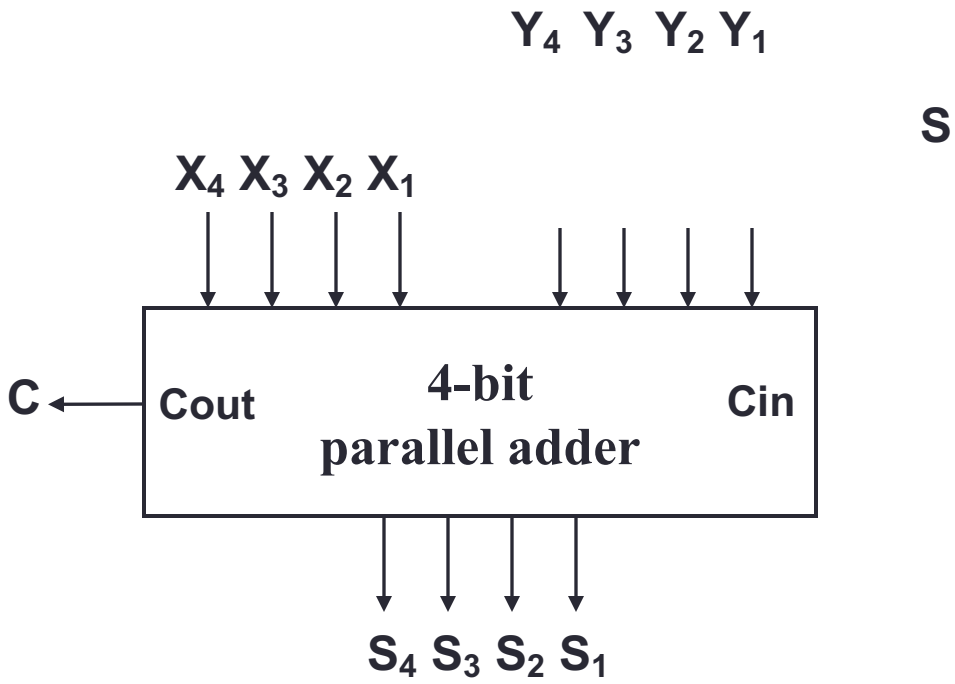
$$\begin{aligned} X - Y &= X + (-Y) \\ &= X + (\text{2s complement of } Y) \\ &= X + (\text{1s complement of } Y) + 1 \end{aligned}$$

- Design requires:

  (1) XOR gates, and (2) S connected to carry-in.

4-BIT ADDER CUM SUBTRACTOR (3/3)

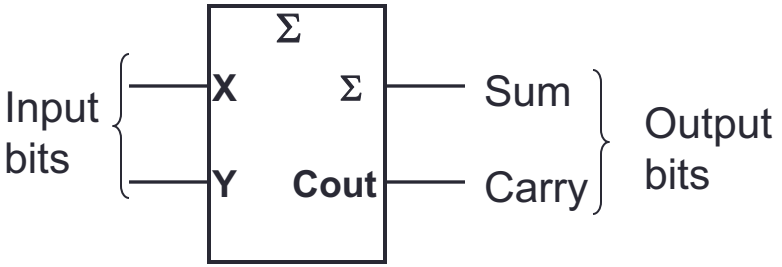
- 4-bit adder-cum-subtractor circuit:

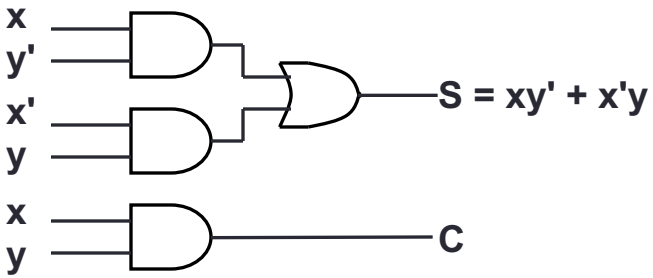


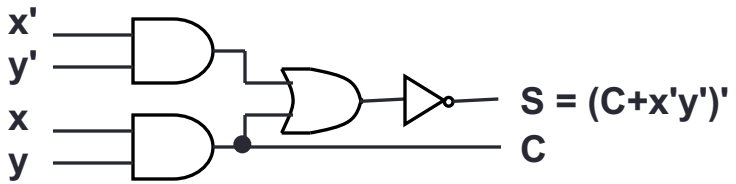
REVISION: HALF ADDER

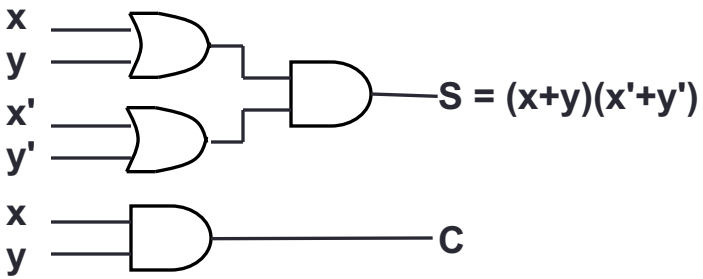
- Half adder

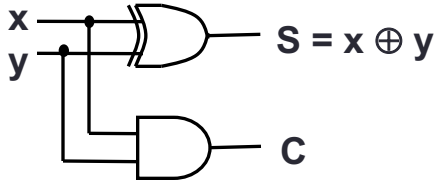
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0







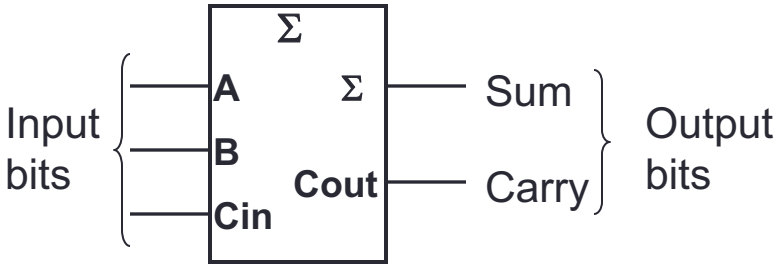


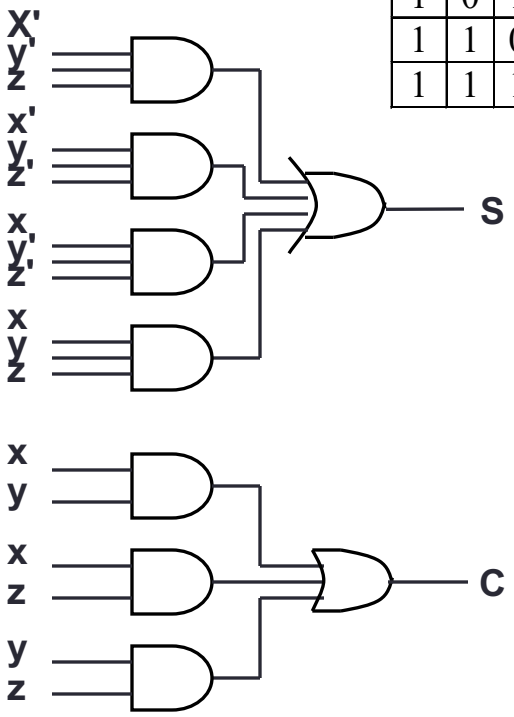


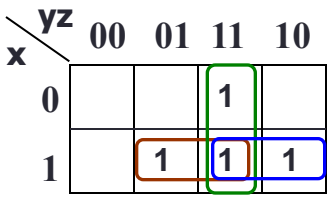
REVISION: FULL ADDER

- Full adder

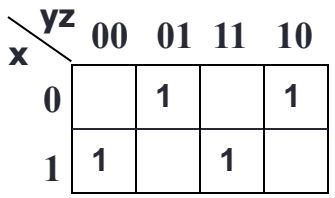
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



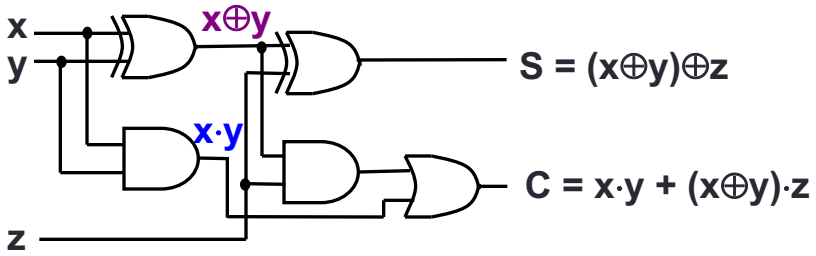




C = xy + xz + yz



S = x'y'z + x'yz' + xy'z' + xyz



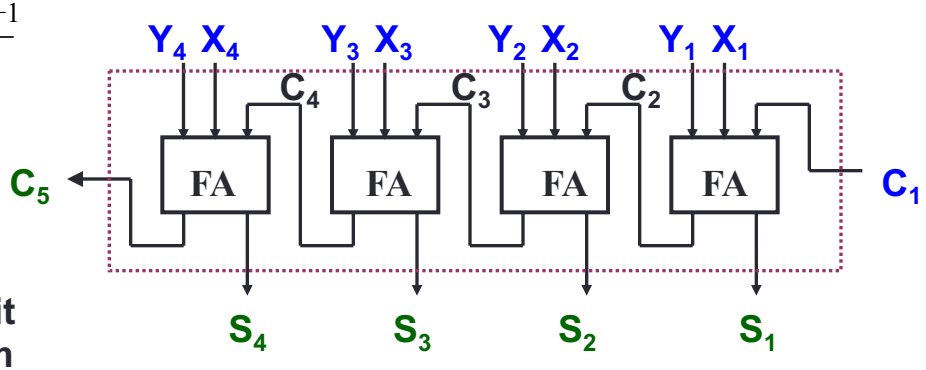
REVISION: PARALLEL ADDER

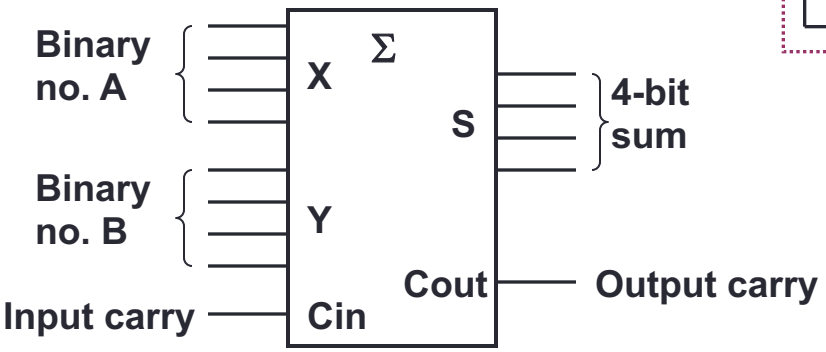
- 4-bit parallel adder

Subscript i	4	3	2	1	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

2 ways:

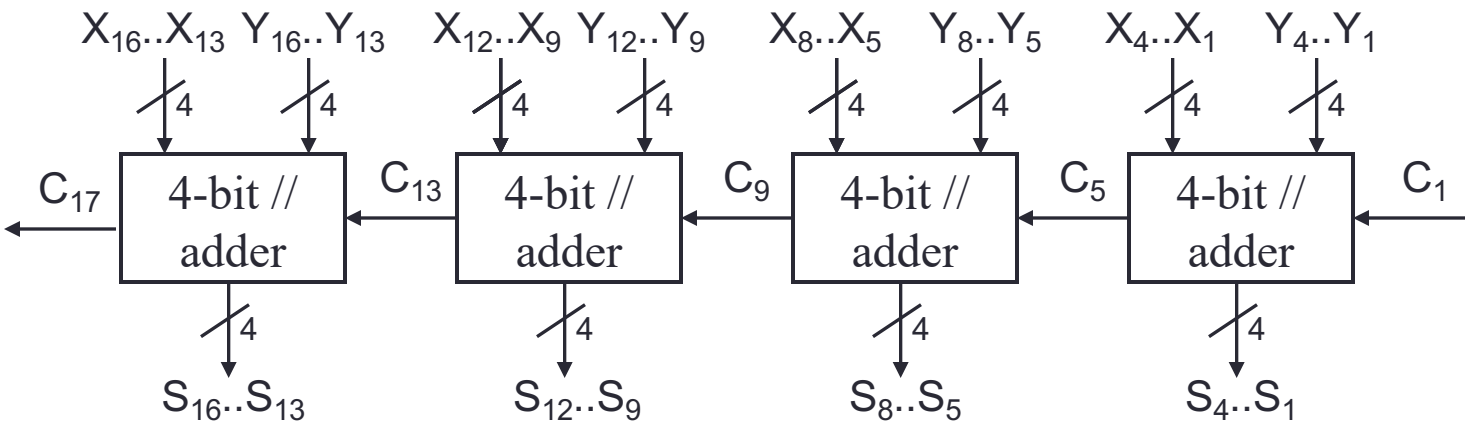
- Serial (one FA)
- Parallel (*n* FAs for *n* bits)





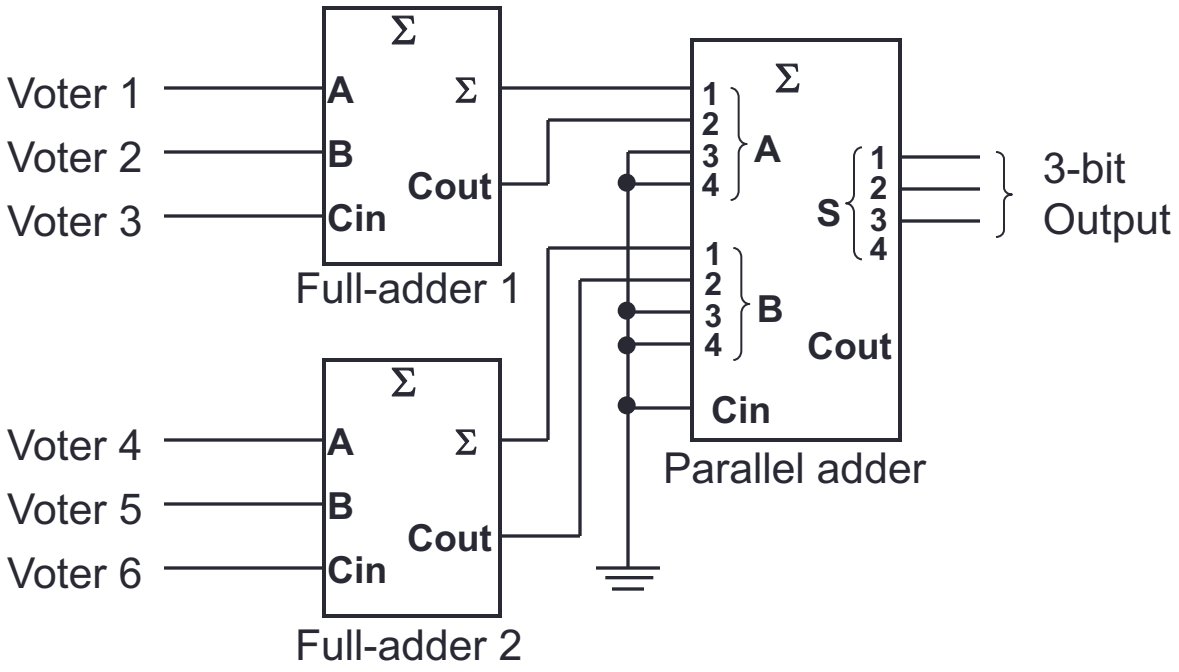
REVISION: CASCADING ADDERS

- Cascading 4 full adders (FAs) gives a 4-bit parallel adder.
  - Classical method: 9 input variables → 2^9 = 512 rows in truth table!
- Cascading method can be extended to larger adders.
  - Example: 16-bit parallel adder.



EXAMPLE

- Application: 6-person voting system.
  - Use FAs and a 4-bit parallel adder.
  - Each FA can sum up to 3 votes.

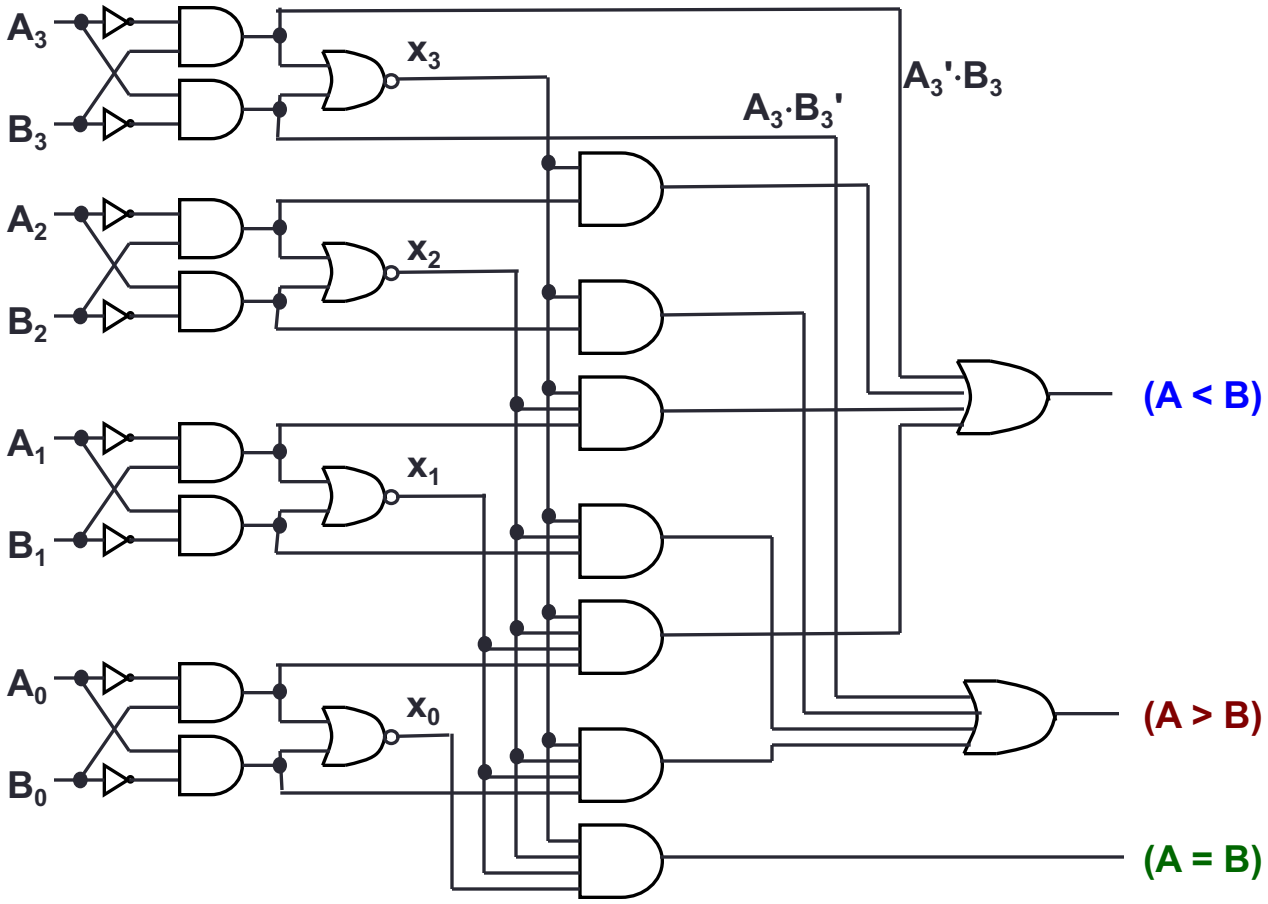


MAGNITUDE COMPARATOR (1/4)

- Magnitude comparator: compares 2 values A and B, to check if A>B, A=B, or A<B.
- To design an *n*-bit magnitude comparator using classical method, it would require 2^{2n} rows in truth table!
- We shall exploit regularity in our design.
- Question: How do we compare two 4-bit values A ($a_3a_2a_1a_0$) and B ($b_3b_2b_1b_0$)?

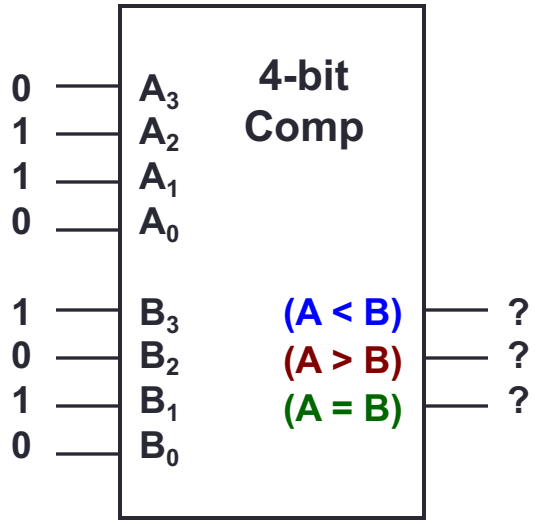
MAGNITUDE COMPARATOR (2/4)

Let $A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$; $x_i = A_i \cdot B_i + A_i' \cdot B_i'$



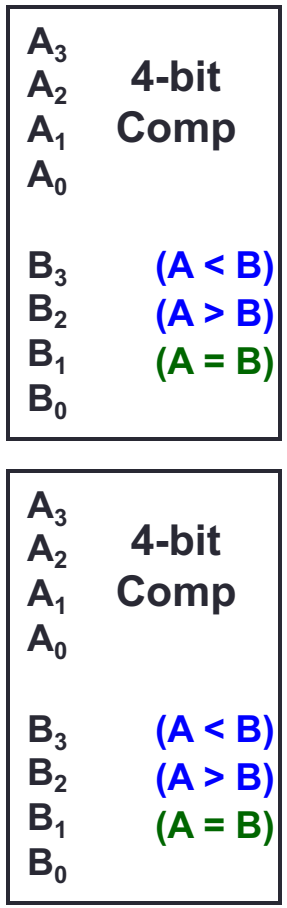
MAGNITUDE COMPARATOR (3/4)

- Block diagram of a 4-bit magnitude comparator



MAGNITUDE COMPARATOR (4/4)

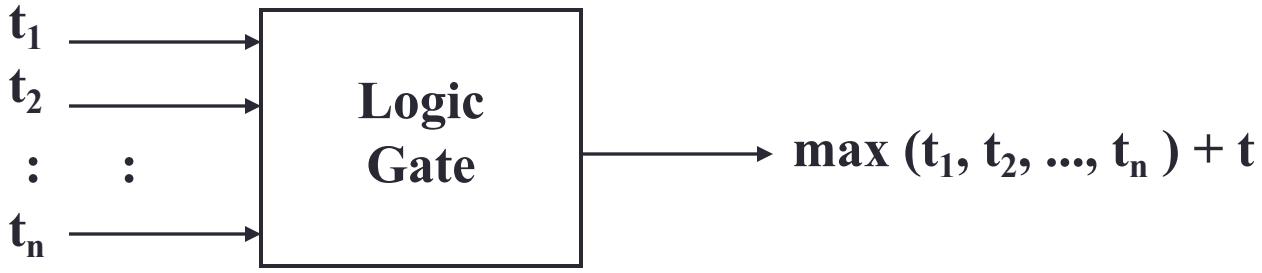
- A function F accepts a 4-bit binary value ABCD, and returns 1 if 3 ≤ ABCD ≤ 12, or 0 otherwise. How would you implement F using magnitude comparators and a suitable logic gate?



CIRCUIT DELAYS (1/5)

- Given a logic gate with delay t. If inputs are stable at times $t_1, t_2, \ldots, t_n$, then the earliest time in which the output will be stable is:

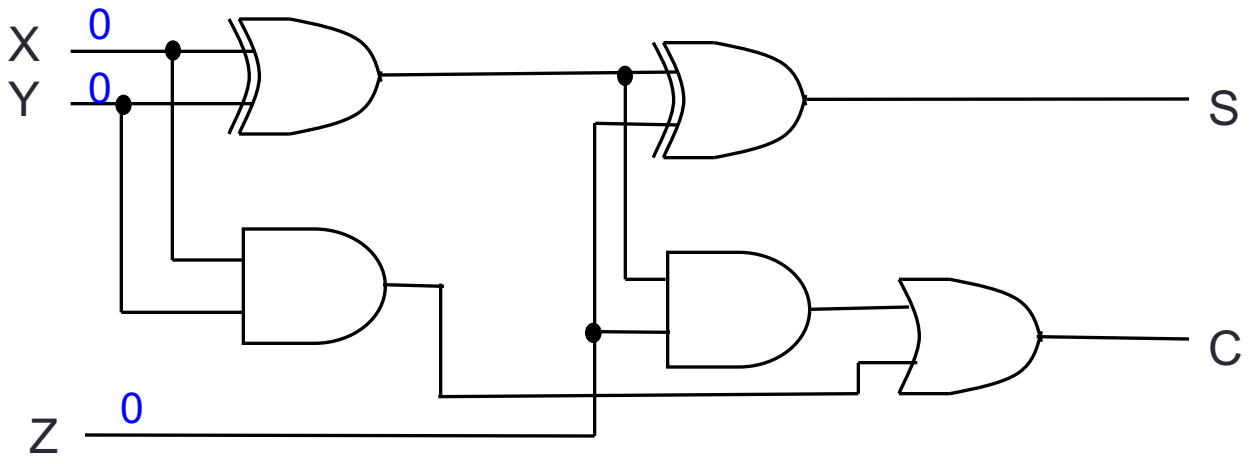
$$\max(t_1, t_2, \ldots, t_n) + t$$



- To calculate the delays of all outputs of a combinational circuit, repeat above rule for all gates.

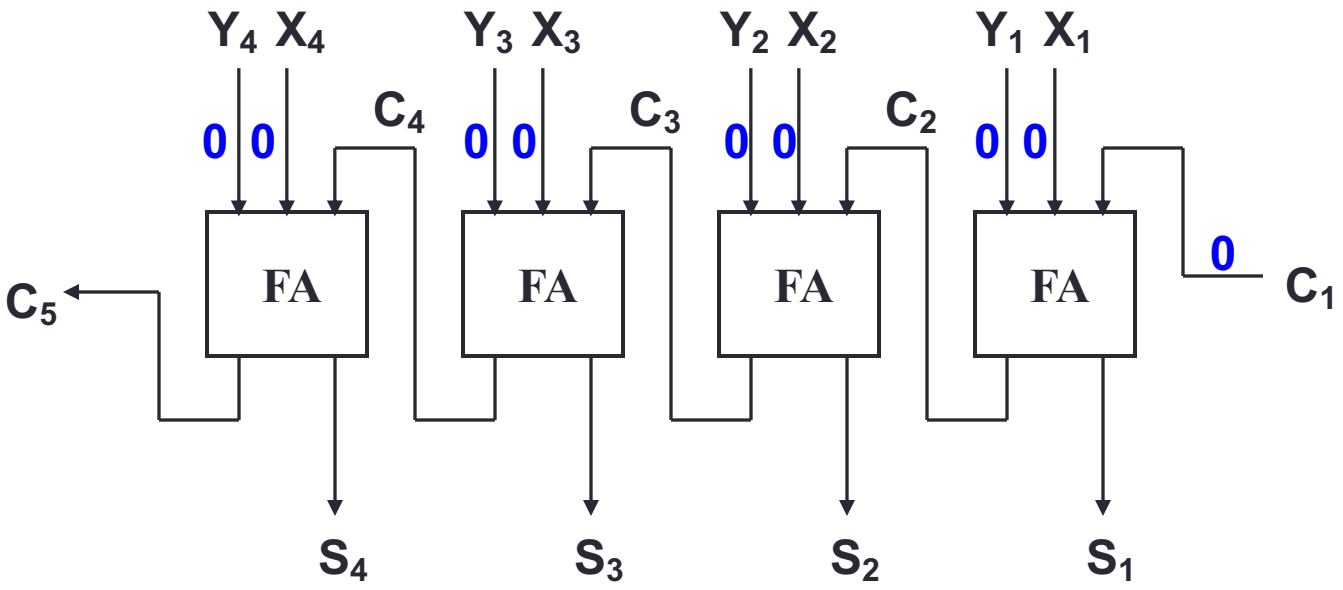
CIRCUIT DELAYS (2/5)

- As a simple example, consider the full adder circuit where all inputs are available at time 0. Assume each gate has delay t.



CIRCUIT DELAYS (3/5)

- More complex example: 4-bit parallel adder.

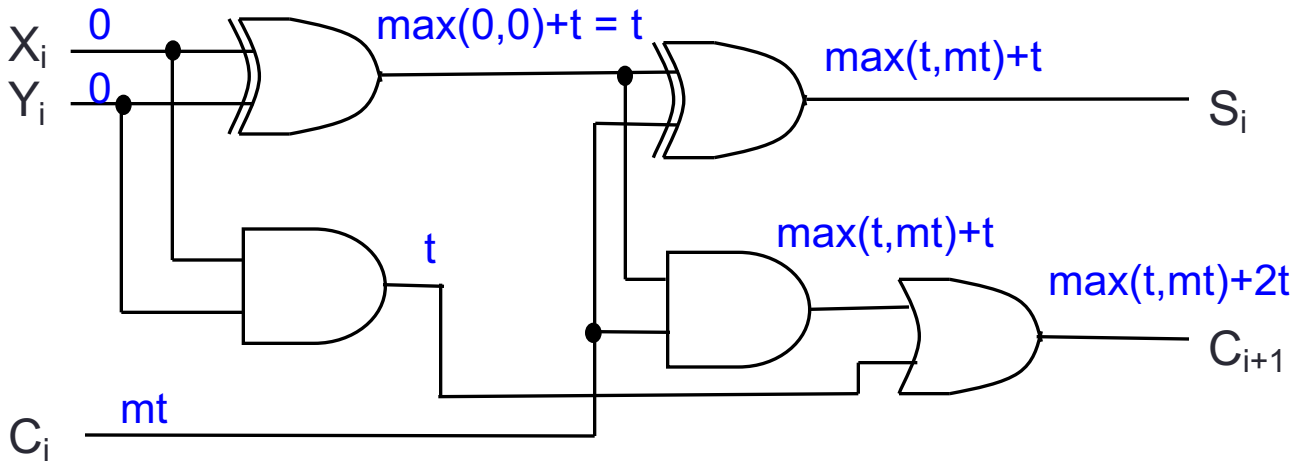


CIRCUIT DELAYS (4/5)

- Analyse the delay for the repeated block.

where X_i, Y_i are stable at 0t, while C_i is assumed to be stable at mt.

- Performing the delay calculation:



CIRCUIT DELAYS (5/5)

- Calculating:

  When i=1, m=0; S_1 = 2t and C_2 = 3t
  When i=2, m=3; S_2 = 4t and C_3 = 5t
  When i=3, m=5; S_3 = 6t and C_4 = 7t
  When i=4, m=7; S_4 = 8t and C_5 = 9t

- In general, an *n*-bit ripple-carry parallel adder will experience the following delay times:

  S_n = ?
  C_{n+1} = ?

- Propagation delay of ripple-carry parallel adders is proportional to the number of bits it handles.
- Maximum delay: ?

FASTER CIRCUITS

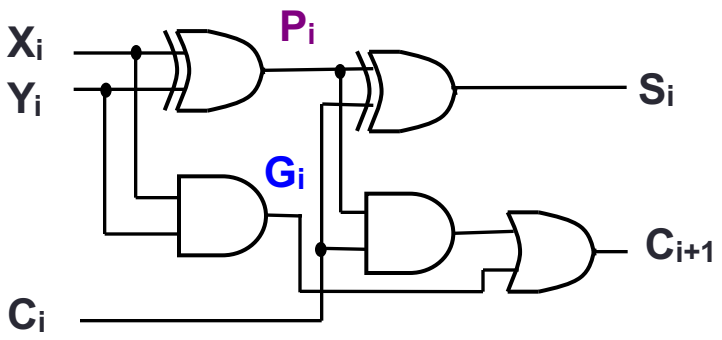
- Three ways of improving the speed of circuits:
  - Use better technology (eg. ECL faster than TTL gates) BUT
    - Faster technology is more expensive, needs more power, lower-level of integrations
    - Physical limits (eg. speed of light, size of atom)
  - Use gate-level designs to two-level circuits! (use sum-of-products/product-of-sums) BUT
    - Complicated designs for large circuits
    - Product/sum terms need MANY inputs!
  - Use clever look-ahead techniques BUT
    - There are additional costs (hopefully reasonable).

LOOK-AHEAD CARRY ADDER (1/6)

- Consider the FA, where intermediate signals are labelled as P_i and G_i:

  $P_i = X_i \oplus Y_i$

  $G_i = X_i \cdot Y_i$



- The outputs C_{i+1}, S_i, in terms of P_i, G_i, C_i are:

  $S_i = P_i \oplus C_i$ … (1)

  $C_{i+1} = G_i + P_i \cdot C_i$ … (2)

- Looking at equation (2):

  $G_i = X_i \cdot Y_i$ is a *carry generate* signal, and

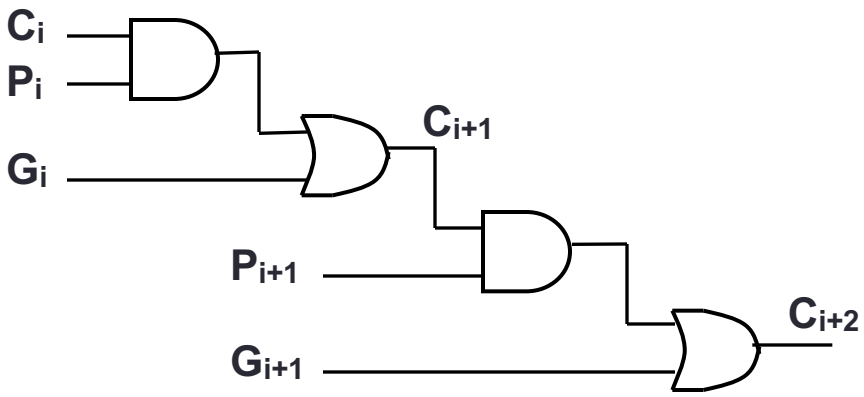
  $P_i = X_i \oplus Y_i$ is a *carry propagate* signal.

LOOK-AHEAD CARRY ADDER (2/6)

- For 4-bit ripple-carry adder, the equations for the four carry signals are:

$$C_{i+1} = G_i + P_i \cdot C_i$$
$$C_{i+2} = G_{i+1} + P_{i+1} \cdot C_{i+1}$$
$$C_{i+3} = G_{i+2} + P_{i+2} \cdot C_{i+2}$$
$$C_{i+4} = G_{i+3} + P_{i+3} \cdot C_{i+3}$$

- These formulae are deeply nested, as shown here for C_{i+2}:



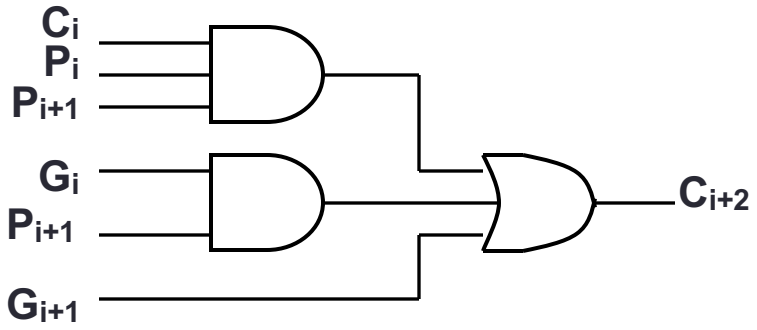
4-level circuit for $C_{i+2} = G_{i+1} + P_{i+1} \cdot C_{i+1}$

LOOK-AHEAD CARRY ADDER (3/6)

- Nested formulae/gates cause more propagation delay.
- Reduce delay by expanding and flattening the formulae for carries. Example, for C_{i+2}:

$$
\begin{aligned}
C_{i+2} &= G_{i+1} + P_{i+1} \cdot C_{i+1} \\
&= G_{i+1} + P_{i+1} \cdot (G_i + P_i \cdot C_i) \\
&= G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i \cdot C_i
\end{aligned}
$$

- New faster circuit for C_{i+2}:



LOOK-AHEAD CARRY ADDER (4/6)

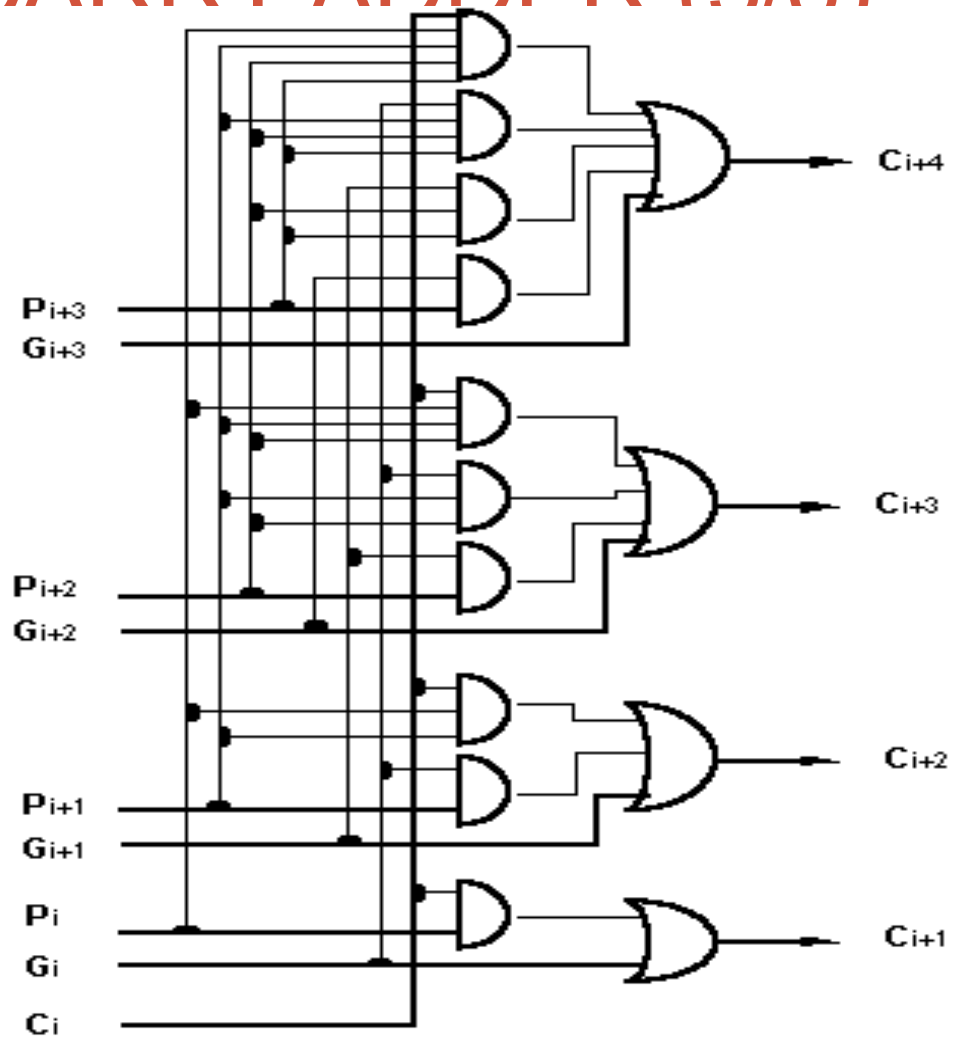
- Other carry signals can be similarly flattened:

$$
\begin{aligned}
C_{i+3} &= G_{i+2} + P_{i+2} \cdot C_{i+2} \\
&= G_{i+2} + P_{i+2} \cdot (G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i \cdot C_i) \\
&= G_{i+2} + P_{i+2} \cdot G_{i+1} + P_{i+2} \cdot P_{i+1} \cdot G_i + P_{i+2} \cdot P_{i+1} \cdot P_i \cdot C_i \\
C_{i+4} &= G_{i+3} + P_{i+3} \cdot C_{i+3} \\
&= G_{i+3} + P_{i+3} \cdot (G_{i+2} + P_{i+2} \cdot G_{i+1} + P_{i+2} \cdot P_{i+1} \cdot G_i + P_{i+2} \cdot P_{i+1} \cdot P_i \cdot C_i) \\
&= G_{i+3} + P_{i+3} \cdot G_{i+2} + P_{i+3} \cdot P_{i+2} \cdot G_{i+1} + P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot G_i + \\
&\quad P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot P_i \cdot C_i
\end{aligned}
$$

- Note that formulae gets longer with higher carries.
- Also, all carries are two-level sum-of-products expressions, in terms of the generate signals **G**s, the propagate signals **P**s, and the first carry-in $\mathbf{C_i}$.

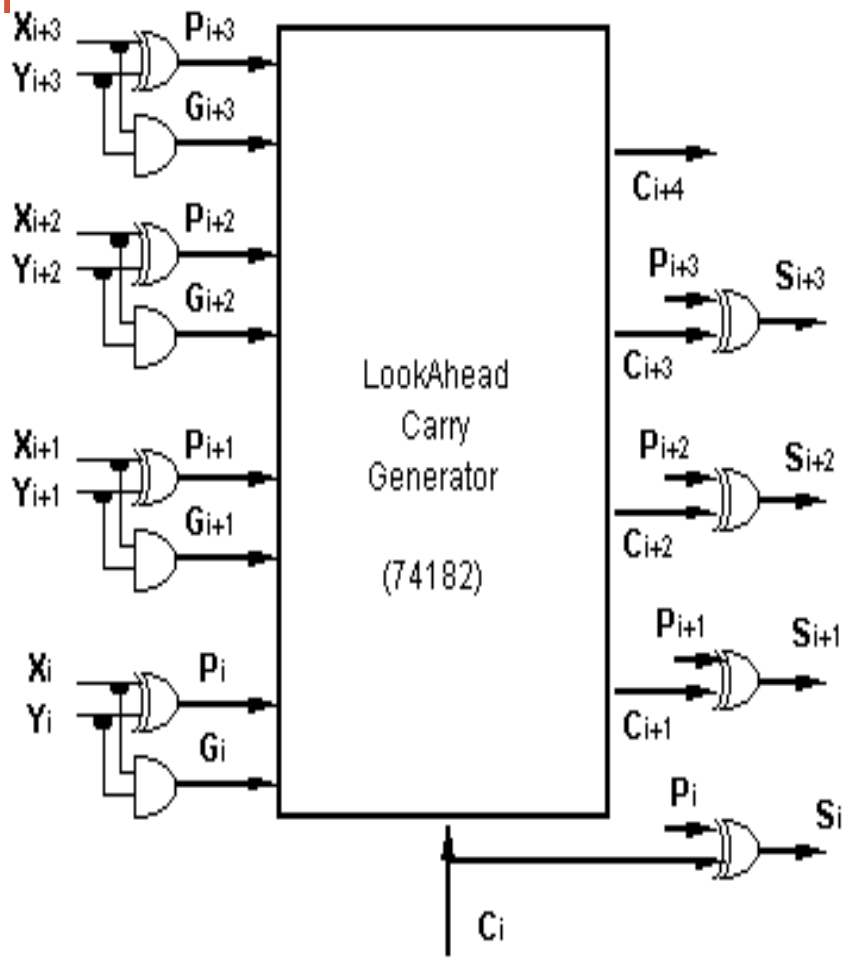
LOOK-AHEAD CARRY ADDER (5/6)

- We employ look-ahead formula in this lookahead-carry adder circuit:



LOOK-AHEAD CARRY ADDER (6/6)

- The 74182 IC chip allows faster lookahead adder to be built.
- Assuming gate delay is t, maximum propagation delay for circuit is hence 4t
  - t to get generate and propagate signals
  - 2t to get the carries
  - t for the sum signals



4-Bit Adder with Look-Ahead (74181)

QUICK REVIEW QUESTIONS

- DLD pages 128 - 129
  Questions 6-1 to 6-4.

Q&A