CME 2003
Logic Design

Combinational Circuits
Design Methods / Arithmetic Circuits

Şerife YILMAZ

Combinational Circuits:
Design Methods/Arithmetic Circuits

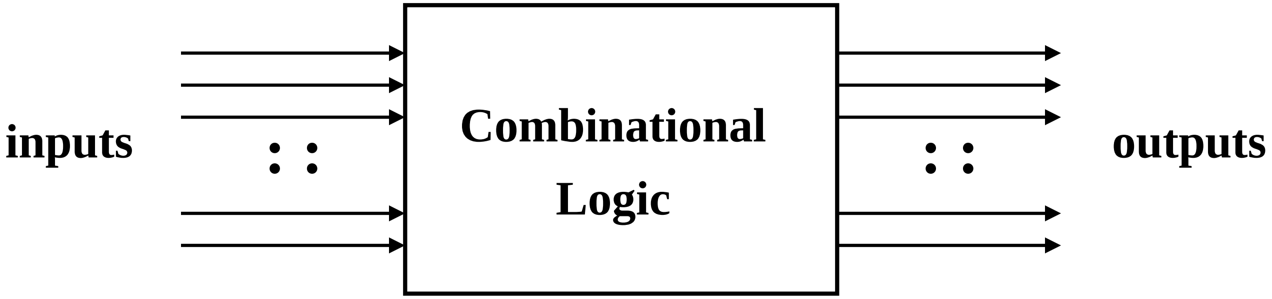
- Introduction
- Analysis Procedure
- Design Methods
- Gate-level (SSI) Design
  - Half Adder
  - Full Adder
  - BCD-to-Excess-3 Code Converter
- Block-Level Design
  - 4-bit Parallel Adder
  - BCD-to-Excess-3 Code Converter
  - 16-bit Parallel Adder
  - 4-bit Parallel Adder / Subtractor

Lecture 6: Combinational Circuits: Design Methods/Arithmetic Circuits

- Arithmetic Circuits
  - Adders
  - Parallel Adders
  - Cascading Adders
  - Parallel Adder-Subtractor
  - Comparator
- Calculations of Circuit Delays
- Faster Circuits
- Look-Ahead Carry Adder

Introduction (1/2)

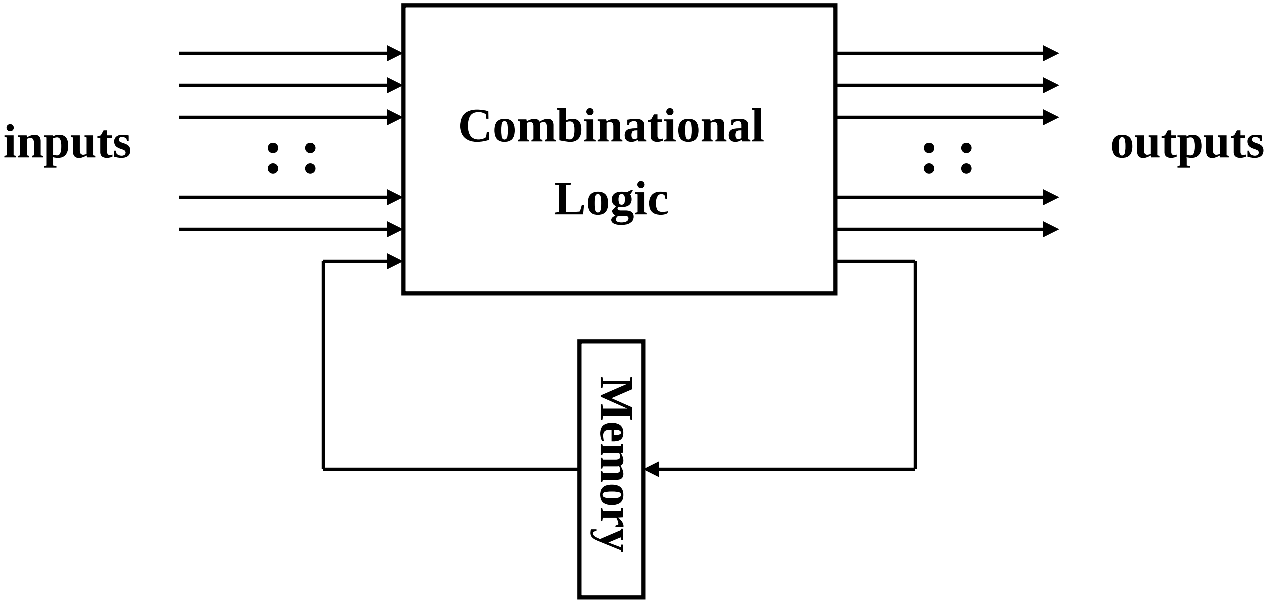
- Two classes of logic circuits:
  - combinational
  - sequential
- **Combinational Circuit**:



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

Introduction (2/2)

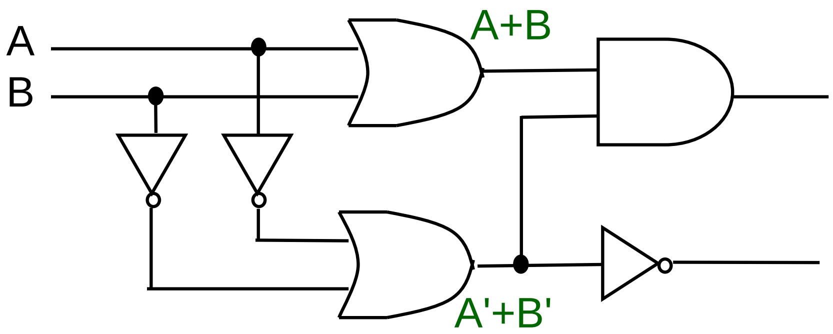
- **Sequential Circuit**: (not covered)



Output depends on both *present* and *past* inputs. Memory (via feedback loop) contains past information.

Analysis Procedure

- Given a combinational circuit, can you analyze its function?



F1 = (A+B).(A'+B')

F2 = (A'+B')' = A.B

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

- 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 ➲ half adder.

Design Methods (1/2)

- Different combinational circuit design methods:
  - Gate-level method (with logic gates)
  - Block-level design method
- Design methods make use of logic gates and useful functional blocks.
  - These are available as Integrated Circuit (IC) chips.

Design Methods (2/2)

- Type of IC chips (based on packing density) :
  - Small-scale integration (SSI): up to 12 gates
  - Medium-scale integration (MSI): 12-99 gates
  - Large-scale integration (LSI): 100-9999 gates
  - Very large-scale integration (VLSI): 10,000-99,999 gates
  - Ultra large-scale integration (ULSI): > 100,000 gates
- Main objectives of circuit design:
  - (i) reduce cost
    - reduce number of gates (for SSI circuits)
    - reduce IC packages (for complex circuits)
  - (ii) increase speed
  - (iii) design simplicity (reuse blocks where possible)

Gate-level (SSI) Design: Half Adder (1/2)

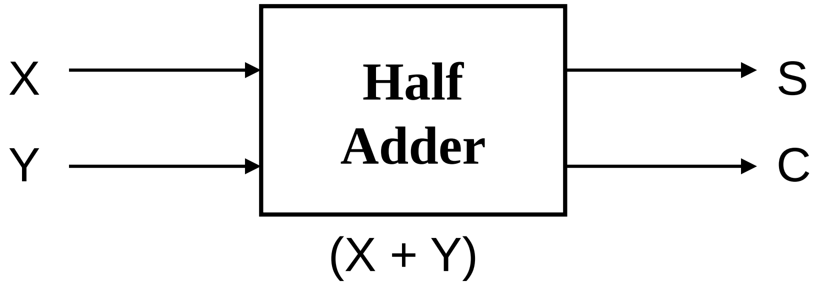
- Design procedure:

  1) State Problem

     Example: Build a Half Adder to add two bits

  2) Determine and label the inputs & outputs of circuit.

     Example: Two inputs and two outputs labelled, as follows:



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

3) Draw truth table.

Gate-level (SSI) Design: Half Adder (2/2)

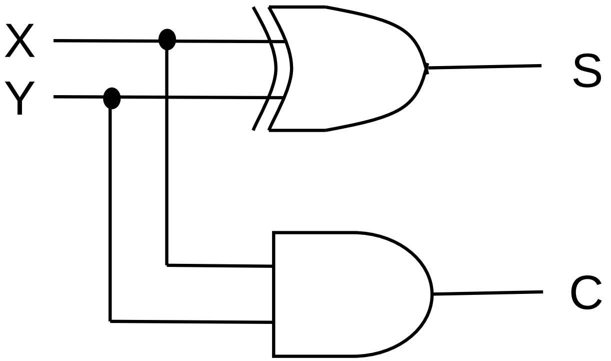
4) Obtain simplified Boolean function.

Example: C = X.Y

S = X'.Y + X.Y' = X⊕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.



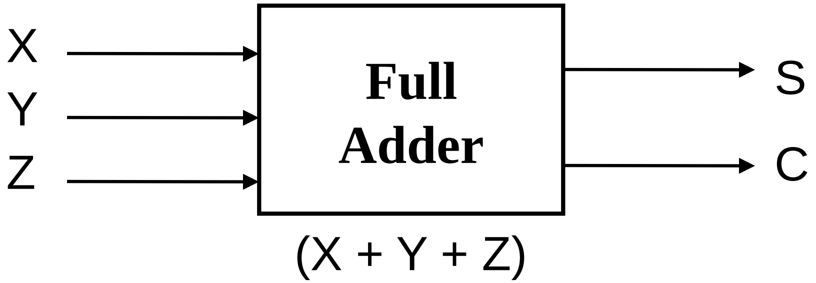
Half Adder

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)

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	

- Using K-map, simplified SOP form:

C = X.Y + X.Z + Y.Z

S = X'.Y'.Z + X'.Y.Z'+X.Y'.Z'+X.Y.Z

Gate-level (SSI) Design: Full Adder (3/5)

- Alternative formulae using algebraic manipulation:

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

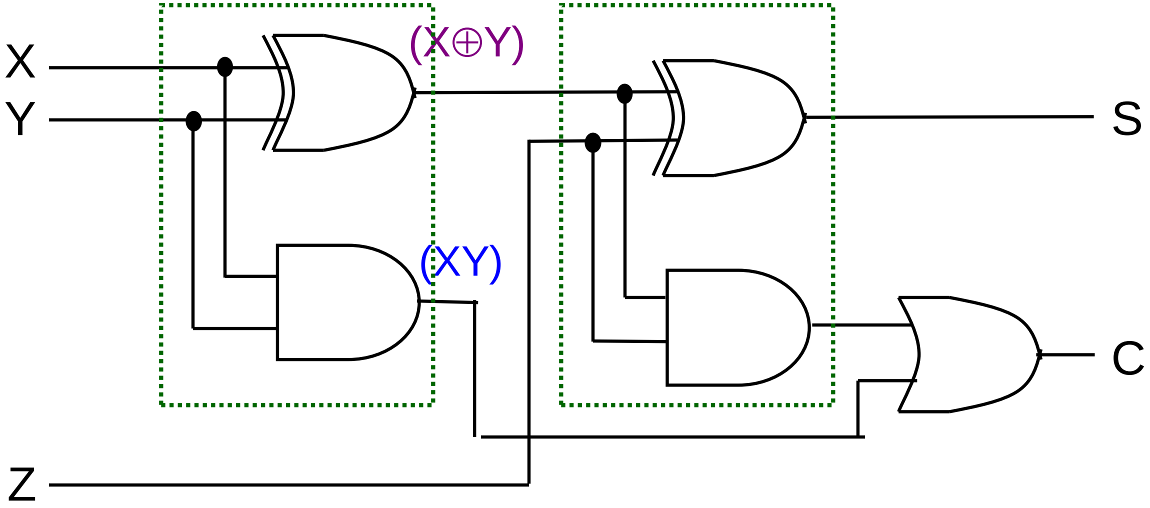
$$
\begin{aligned}
S &= X'.Y'.Z + X'.Y.Z' + X.Y'.Z' + X.Y.Z \\
&= X\text{‘}.(Y'.Z + Y.Z') + X.(Y'.Z' + Y.Z) \\
&= X'.(Y \oplus Z) + X.(Y \oplus Z)' \\
&= X \oplus (Y \oplus Z) \quad \text{or} \quad (X \oplus Y) \oplus Z
\end{aligned}
$$

Gate-level (SSI) Design: Full Adder (4/5)

- Circuit for above formulae:

  $C = X.Y + (X \oplus Y).Z$

  $S = (X \oplus Y) \oplus Z$



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

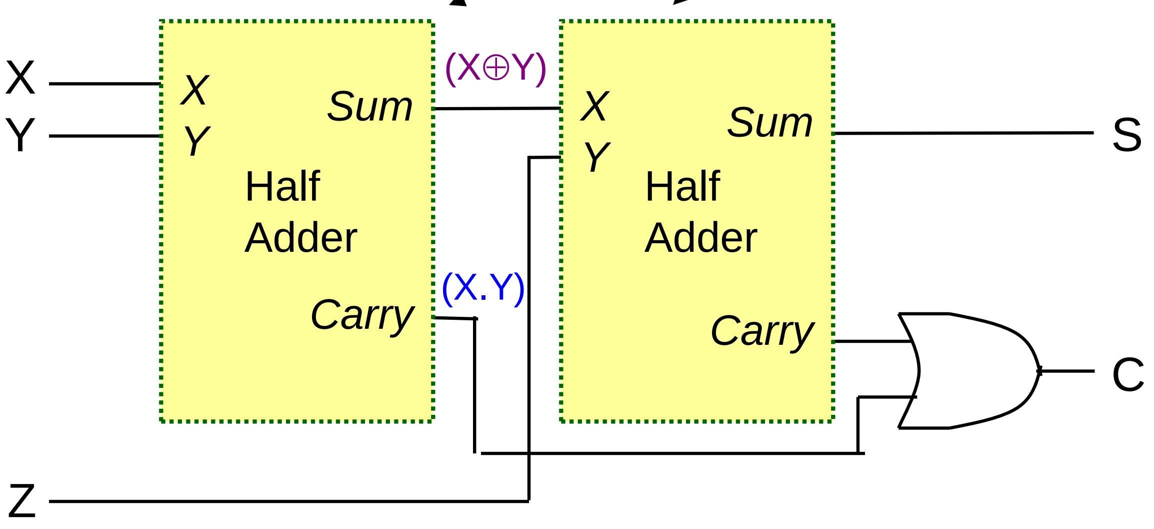
Gate-level (SSI) Design: Full Adder (5/5)

- Circuit for above formulae:

$C = X.Y + (X \oplus Y).Z$

$S = (X \oplus Y) \oplus Z$

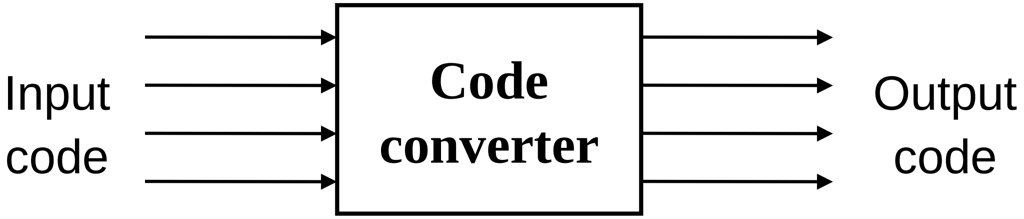
Block diagrams.



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

Code Converters

- Code converters – take an input code, translate to its equivalent output code.



- Example: BCD to Excess-3 Code Converter.

  Input: BCD digit

  Output: Excess-3 digit

BCD-to-Excess-3 Code 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:

				C		
	AB \ CD	00	01	11	10	
	00					
	01		1	1	1	B
A	11	X	X	X	X	
	10	1	1	X	X	
			D			

W

				C		
	AB \ CD	00	01	11	10	
	00		1	1	1	
	01	1				B
A	11	X	X	X	X	
	10		1	X	X	
			D			

X

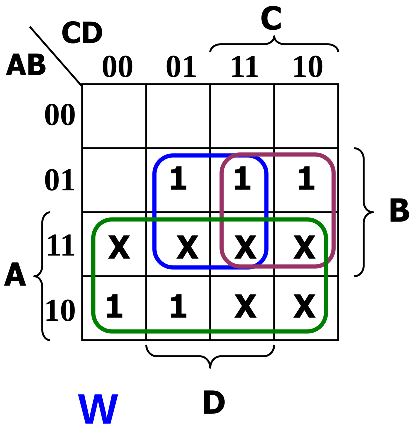
				C		
	AB \ CD	00	01	11	10	
	00	1		1		
	01	1		1		B
A	11	X	X	X	X	
	10	1		X	X	
			D			

Y

				C		
	AB \ CD	00	01	11	10	
	00	1			1	
	01	1			1	B
A	11	X	X	X	X	
	10	1		X	X	
			D			

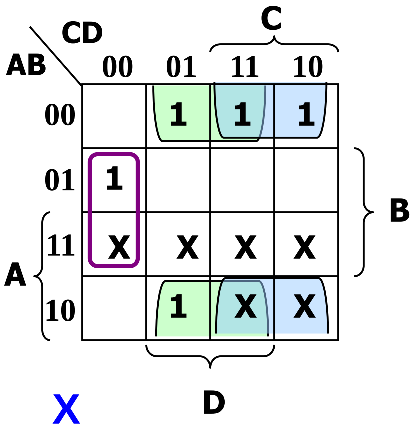
Z

BCD-to-Excess-3 Code Converter (2/2)



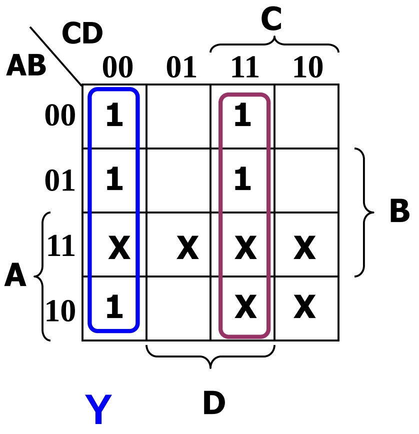
W

AB \ CD	00	01	11	10
00				
01		1	1	1
11	X	X	X	X
10	1	1	X	X



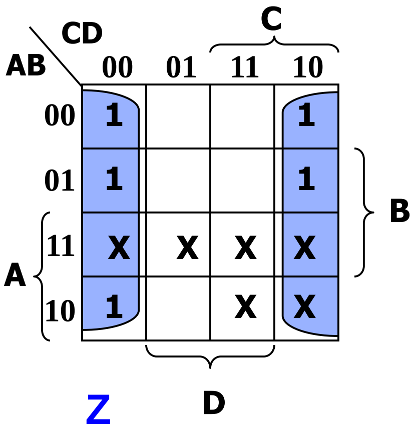
X

AB \ CD	00	01	11	10
00		1	1	1
01	1			
11	X	X	X	X
10		1	X	X



Y

AB \ CD	00	01	11	10
00	1		1	
01	1		1	
11	X	X	X	X
10	1		X	X



Z

AB \ CD	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

W = A + B.C + B.D

X = B'.C + B‘.D + B.C'.D'

Y = C.D + C'.D'

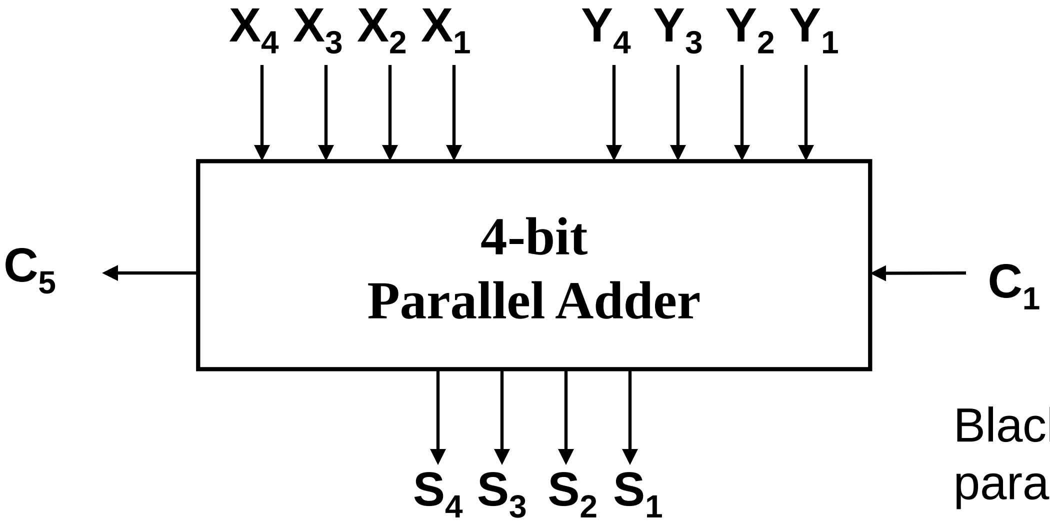
Z = D'

Block-Level Design Method

- 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 Conversion
  - (2) 16-Bit Parallel Adder
  - (3) Adder / 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.
- Truth table for 9 inputs very big, i.e. 2^9=512 entries:

$X_4X_3X_2X_1$	$Y_4Y_3Y_2Y_1$	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 very complicated.

4-bit Parallel Adder (3/4)

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

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

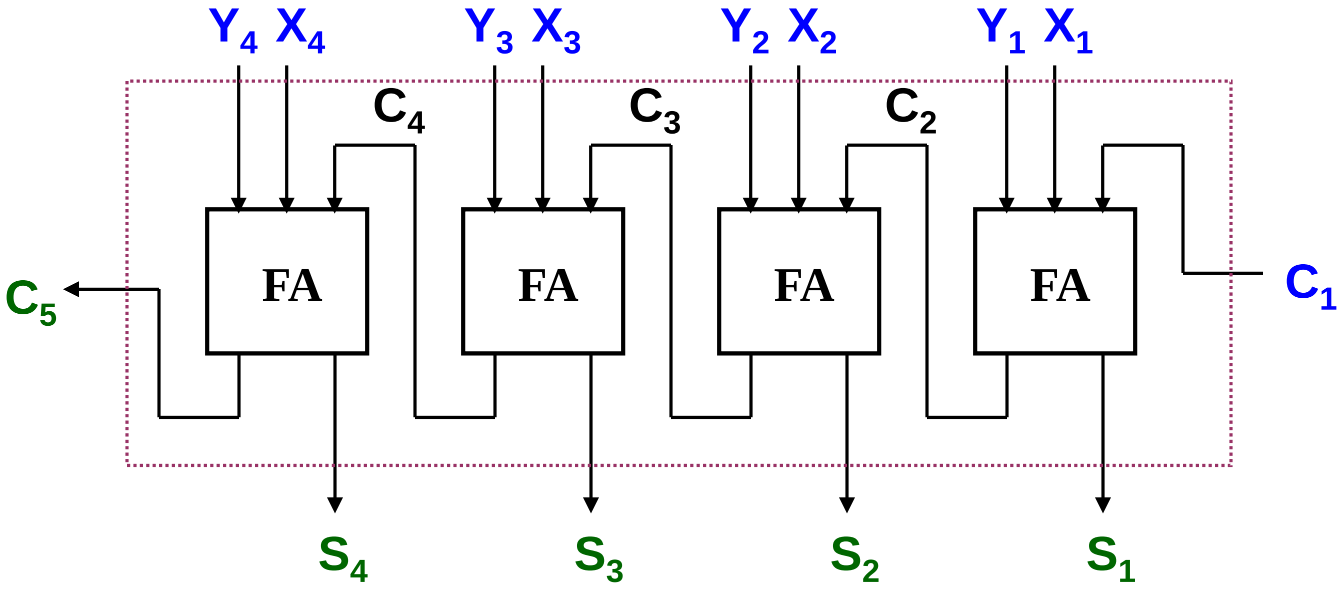
has the same function as a full adder.

$$C_{i+1} = X_i \, .Y_i + (X_i \oplus Y_i\,) \,.C_i$$

$$S_i = X_i \oplus Y_i \oplus C_i$$

4-bit Parallel Adder (4/4)

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



🟦 Input

🟩 Output

Parallel Adders

- Note that carry 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 Code 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 possible.
- Use problem-specific formulae:

Excess-3 Code

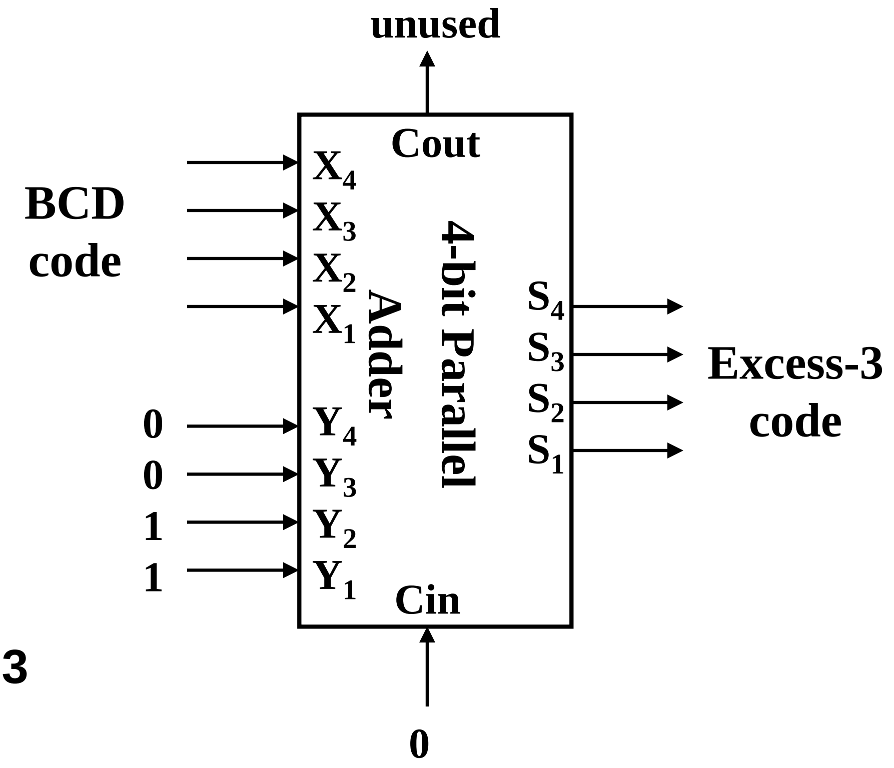
= **BCD** Code + $(0011)_2$

	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 Code Converter (2/2)

Excess-3 Code = **BCD** Code + $(0011)_2$

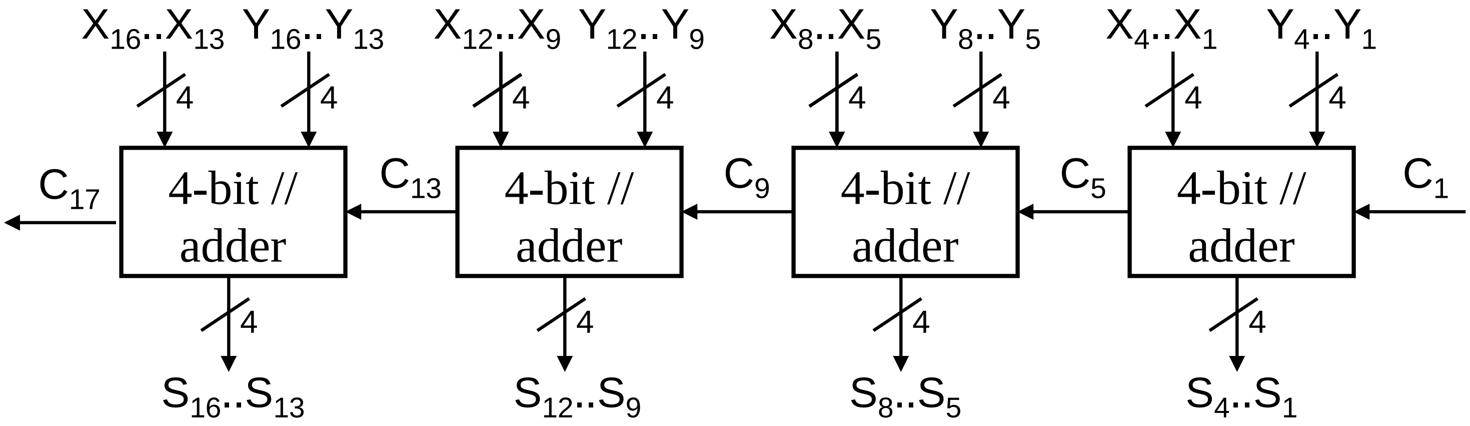
- Block-level circuit:



A BCD-to-Excess-3 Code Converter

16-bit Parallel Adder (1/2)

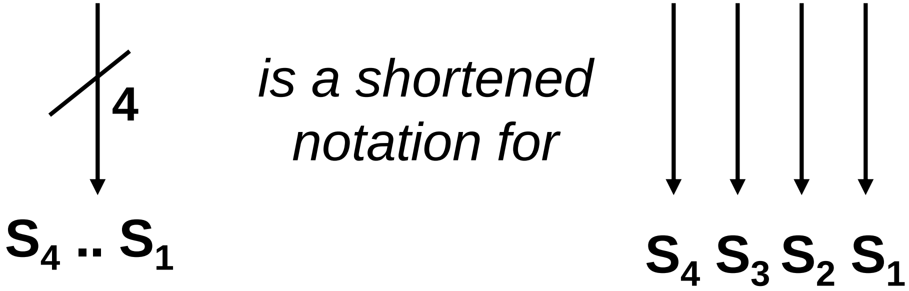
- Larger parallel adders can be built from smaller ones.
- Example: a 16-bit parallel adder can be constructed from four 4-bit parallel adders:



A 16-bit parallel adder

16-bit Parallel Adder (2/2)

- Shortened notation for multiple lines.



4

S_4 .. S_1

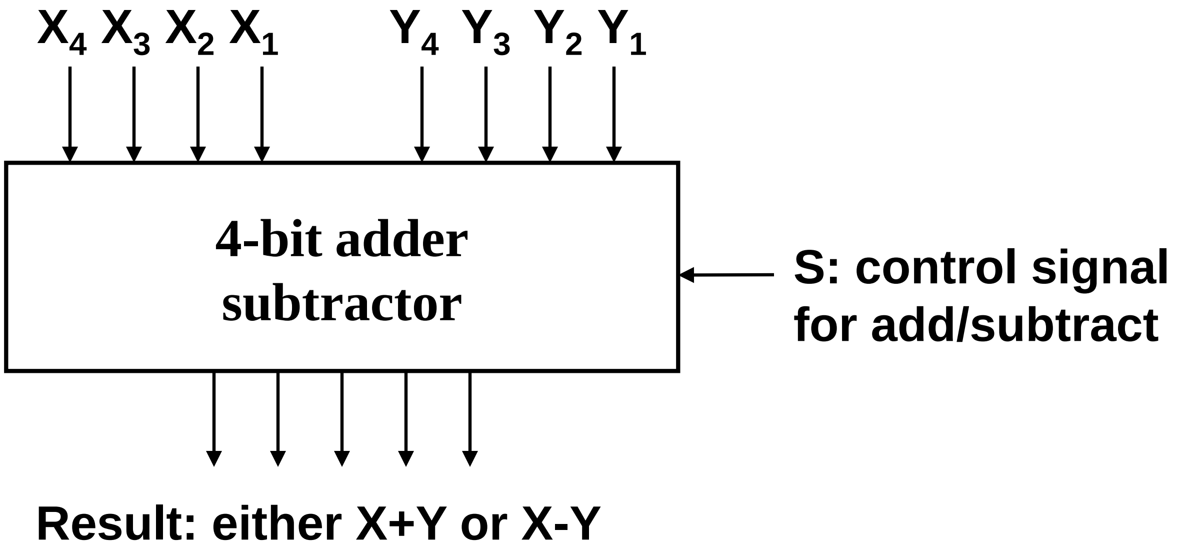
is a shortened notation for

S_4 S_3 S_2 S_1

16-bit parallel adder ripples carry from one 4-bit block to the next.
Such ripple-carry circuits are “slow” because of long delays needed to propagate the carries.

4-bit Parallel Adder/Subtractor (1/4)

- Subtraction can be performed through addition using 2s-complement numbers.
- Hence, we can design a circuit which can perform both addition and subtraction, using a parallel adder.



4-bit Parallel Adder / Subtractor (2/4)

- The control signal **S=0** means add
  S=1 means subtract

- Recall that:

X-Y = X + (-Y)

= X + (2's complement of Y)

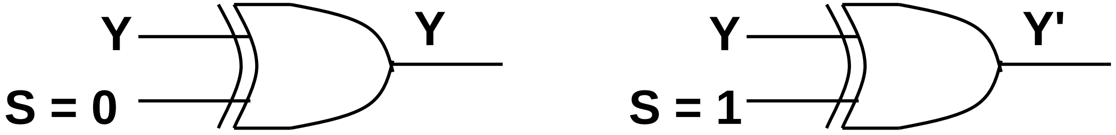
= X + $\underline{\text{(1's complement of Y) +1}}$

X+Y = X + $\underline{\text{(Y)}}$

4-bit Parallel Adder/Subtractor (3/4)

- Design requires:

  (i) XOR gates:

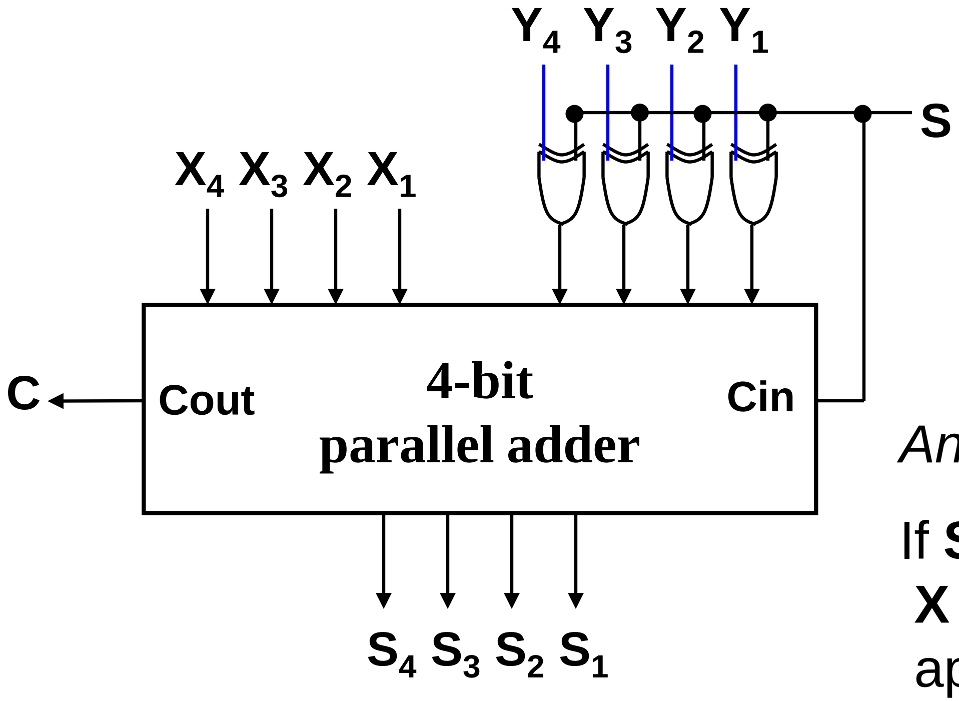


such that: output = Y when S=0
= Y' when S=1

(ii) S connected to carry-in.

4-bit Parallel Adder/Subtractor (4/4)

- Adder / subtractor circuit:



A 4-bit adder/subtractor

Analysis:

If **S**=1, then **X** + (1's complement of **Y**) +1 appears as the result.

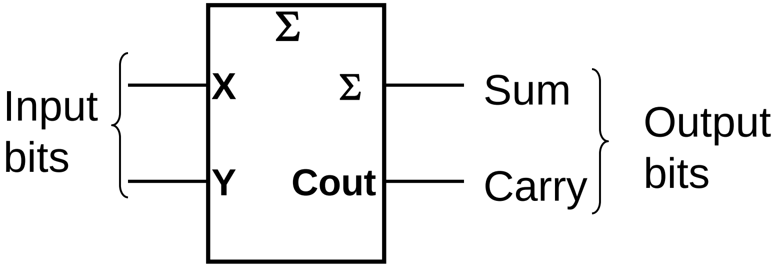
If **S**=0, then **X**+**Y** appears as the result.

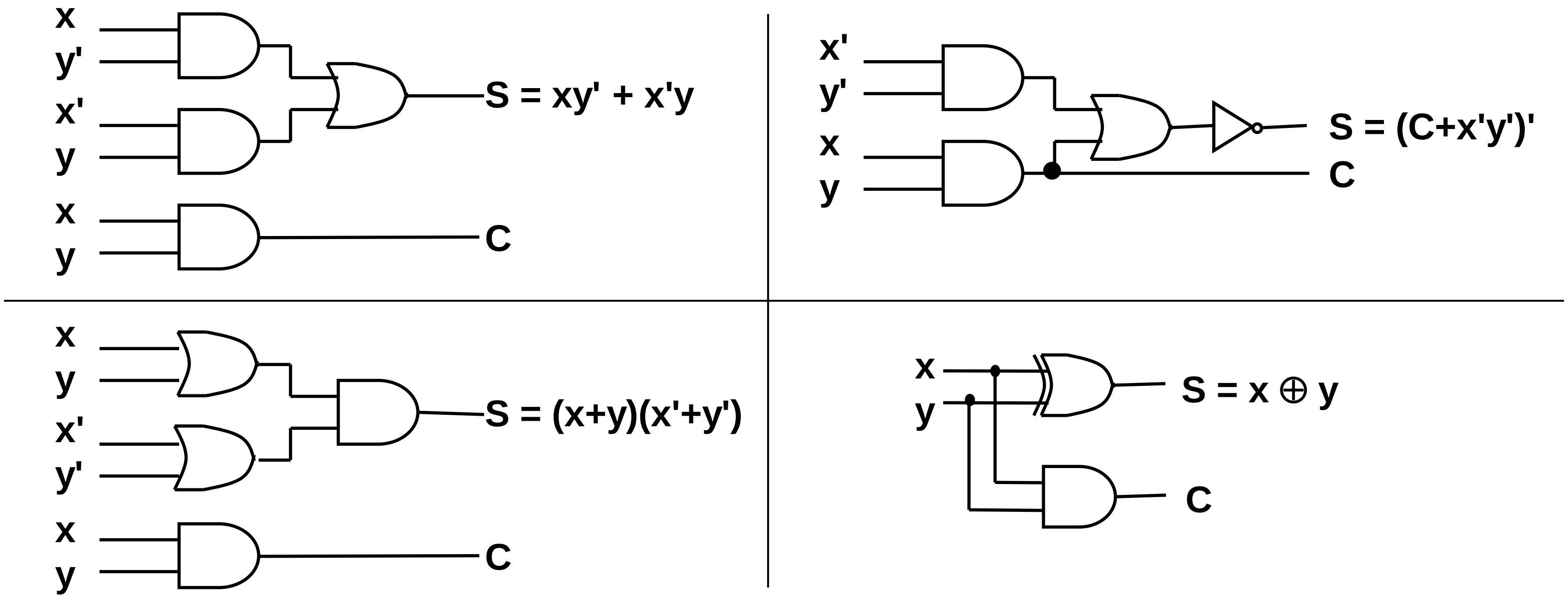
Arithmetic Circuits: Adders

Revision

- Half adder

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



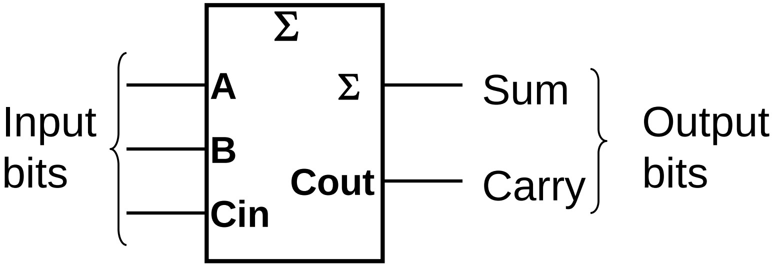


Arithmetic Circuits: Adders

Revision

- 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

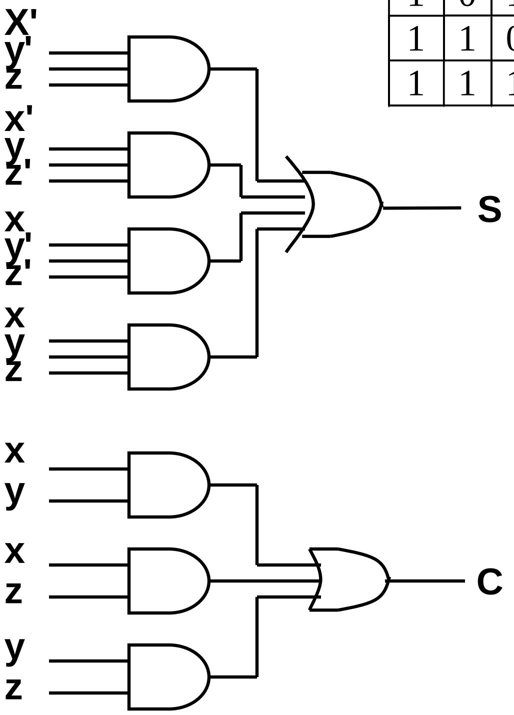


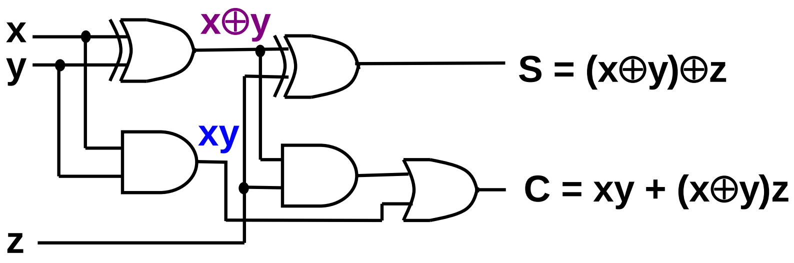
x yz	00	01	11	10
0			1	
1		1	1	1

C = xy + xz + yz

x yz	00	01	11	10
0		1		1
1	1		1	

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





Arithmetic Circuits: Parallel Adders

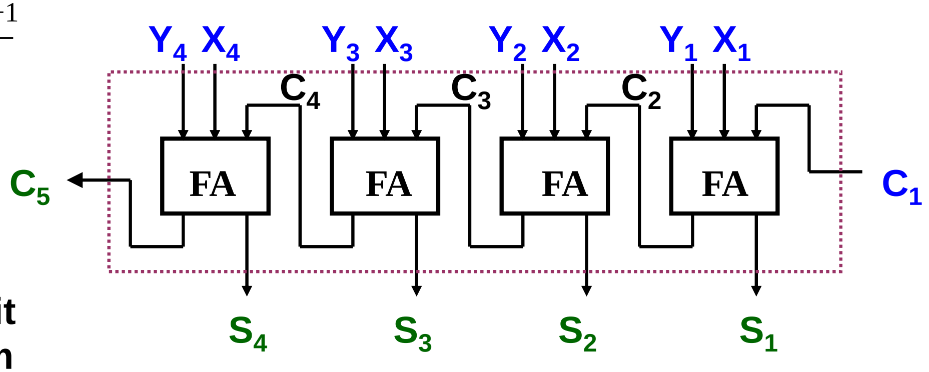
Revision

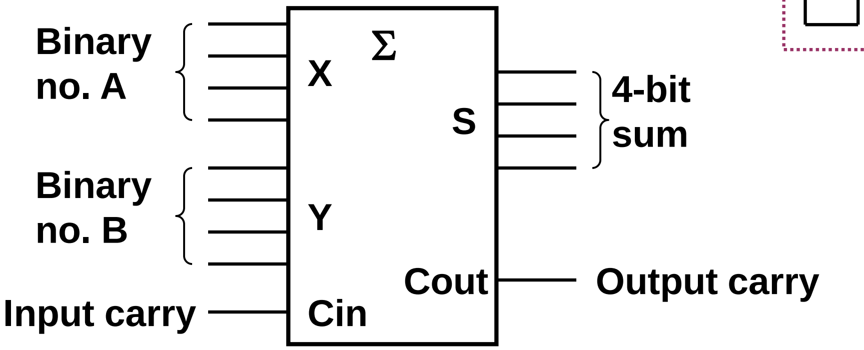
- Example: Adding two 4-bit numbers

Subscript i	4	3	2	1	
Input carry	0	1	1	0	C_i
	1	0	1	1	A_i
	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)

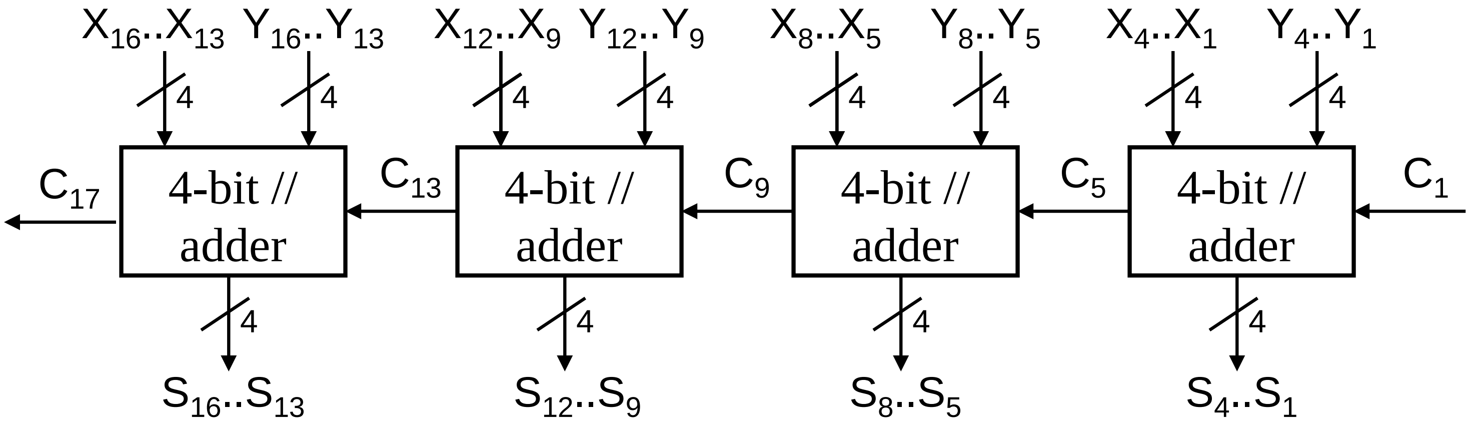




Arithmetic Circuits: Cascading Adders

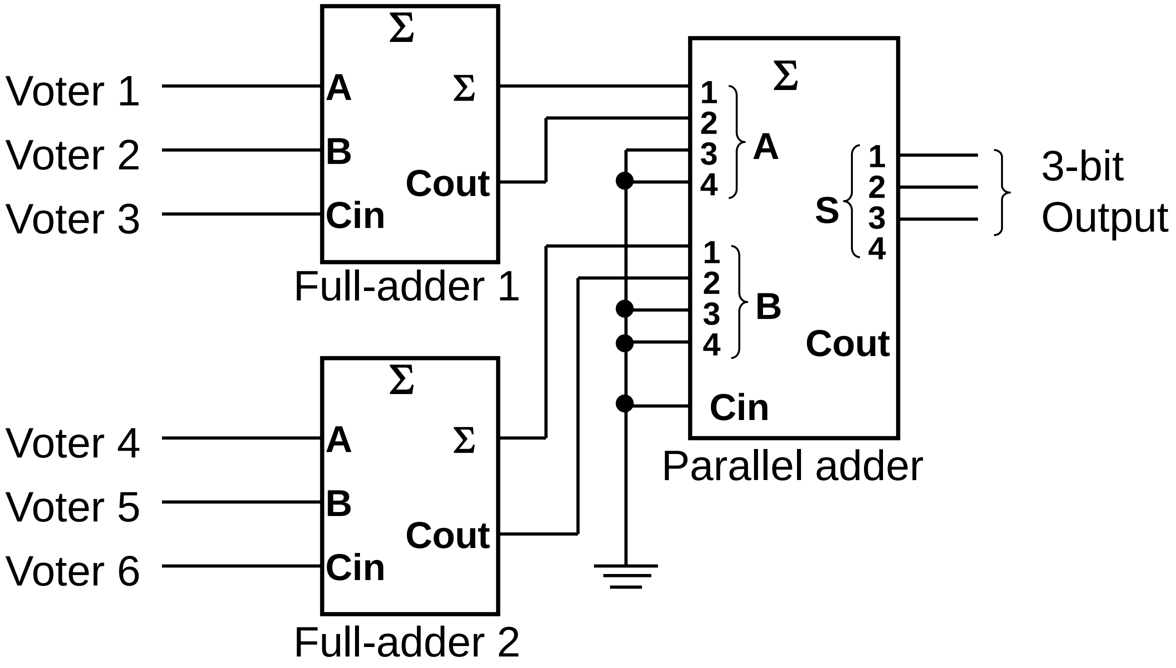
Revision

- 4-bit parallel adder:
  - cascade 4 full adders
  - classical method: 9 input variables ➔ $2^9 = 512$ rows in truth table!
- Cascading method can be extended to larger numbers, example: 16-bit parallel adder.



Arithmetic Circuits: Cascading Adders

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



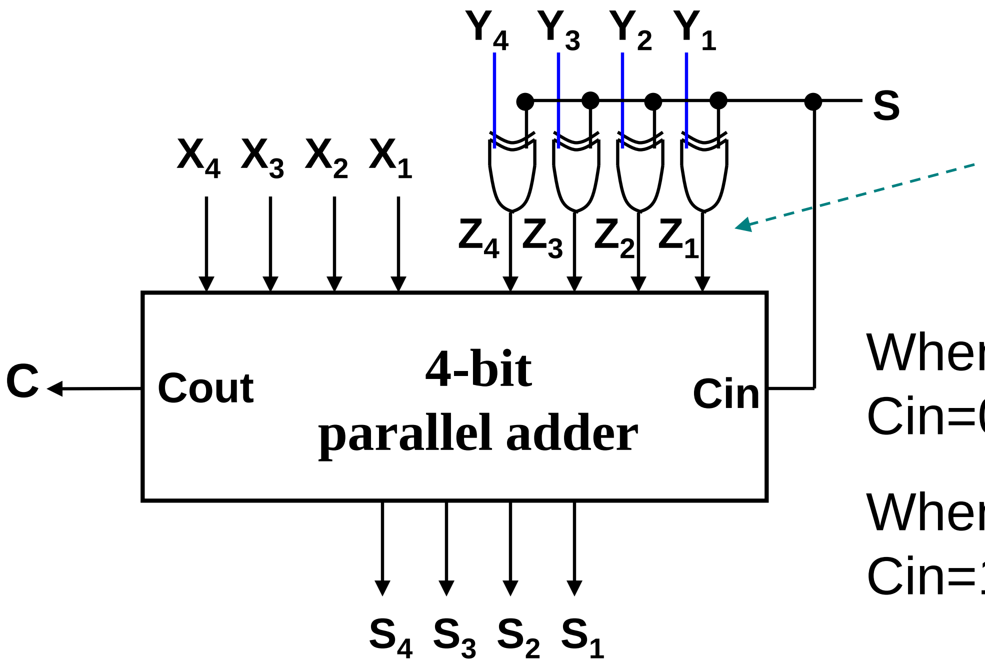
Arithmetic Circuits: Adder-Subtractor

Revision

- Make use of 2's complement:

$$X - Y = X + (-Y)$$

- 2's complement of Y = Inverting bits in Y and plus 1.



$Z_i = S.Y_i' + S'.Y_i$

When S=0,
Cin=0, $Z_i = Y_i$ → $S = X + Y$

When S=1,
Cin=1, $Z_i = Y_i'$ → $S = X + Y' + 1$

Arithmetic Circuits: Comparator (1/3)

- Magnitude comparator: compares 2 values A and B, to see if A>B, A=B or A<B.
- Classical method requires 2^{2n} rows in truth table!
- We exploit regularity.
- How do we compare two 4-bit values A ($a_3a_2a_1a_0$) and B ($b_3b_2b_1b_0$)?

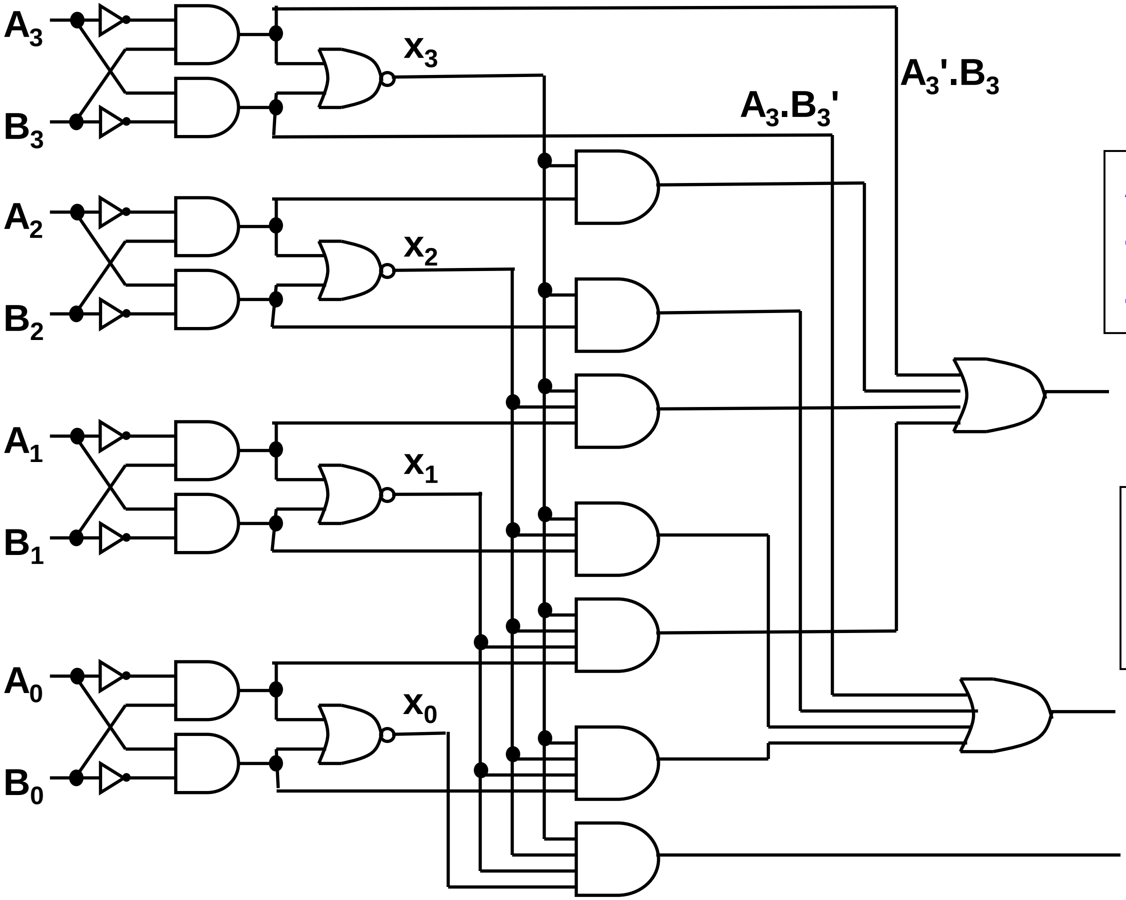
  If ($a_3 > b_3$) then A > B

  If ($a_3 < b_3$) then A < B

  If ($a_3 = b_3$) then if ($a_2 > b_2$) ….

Arithmetic Circuits: Comparator (2/3)

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



$A_3'.B_3 + x_3.A_2'.B_2$
$+ x_3.x_2.A_1'.B_1$
$+ x_3. x_2.x_1.A_0'.B_0$

(A < B)

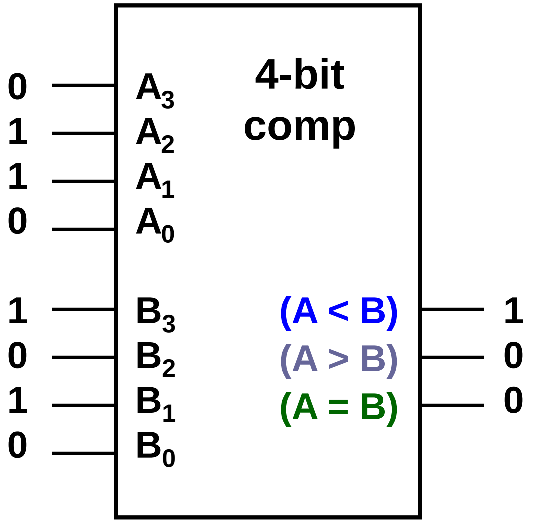
$A_3.B_3' + x_3.A_2.B_2'$
$+ x_3.x_2.A_1.B_1'$
$+ x_3. x_2.x_1.A_0.B_0'$

(A > B)

(A = B)

$x_3. x_2.x_1.x_0$

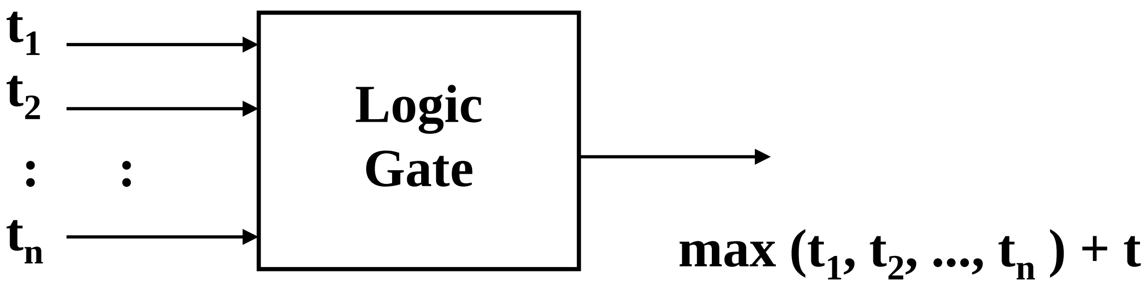
Arithmetic Circuits: Comparator (3/3)



Block diagram of a 4-bit magnitude comparator

Calculation of Circuit Delays (1/5)

- In general, given a logic gate with delay, t.



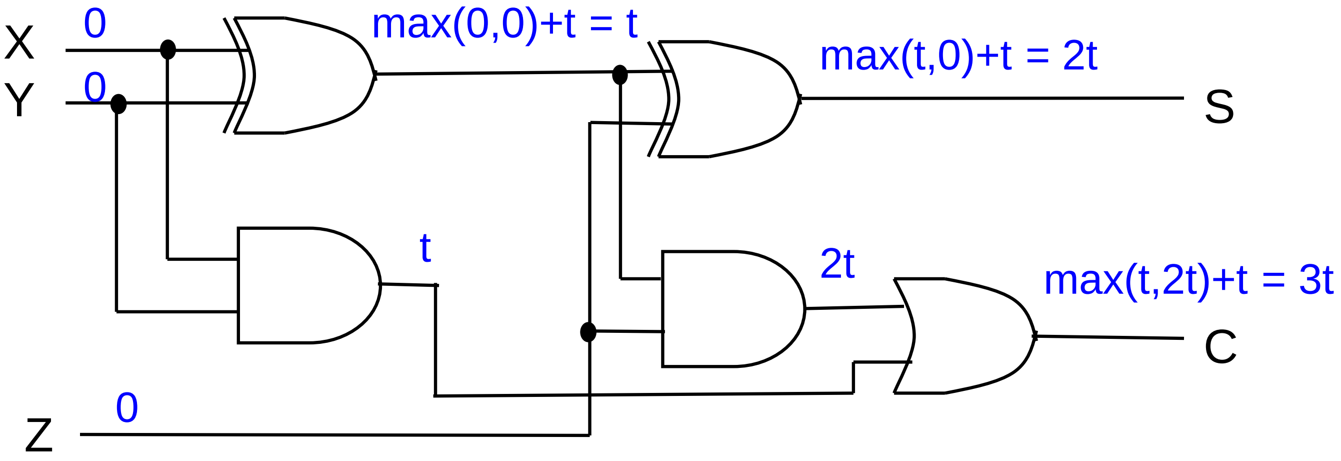
If inputs are stable at times $t_1,t_2,..,t_n$, respectively; then the earliest time in which the output will be stable is:

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

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

Calculation of Circuit Delays (2/5)

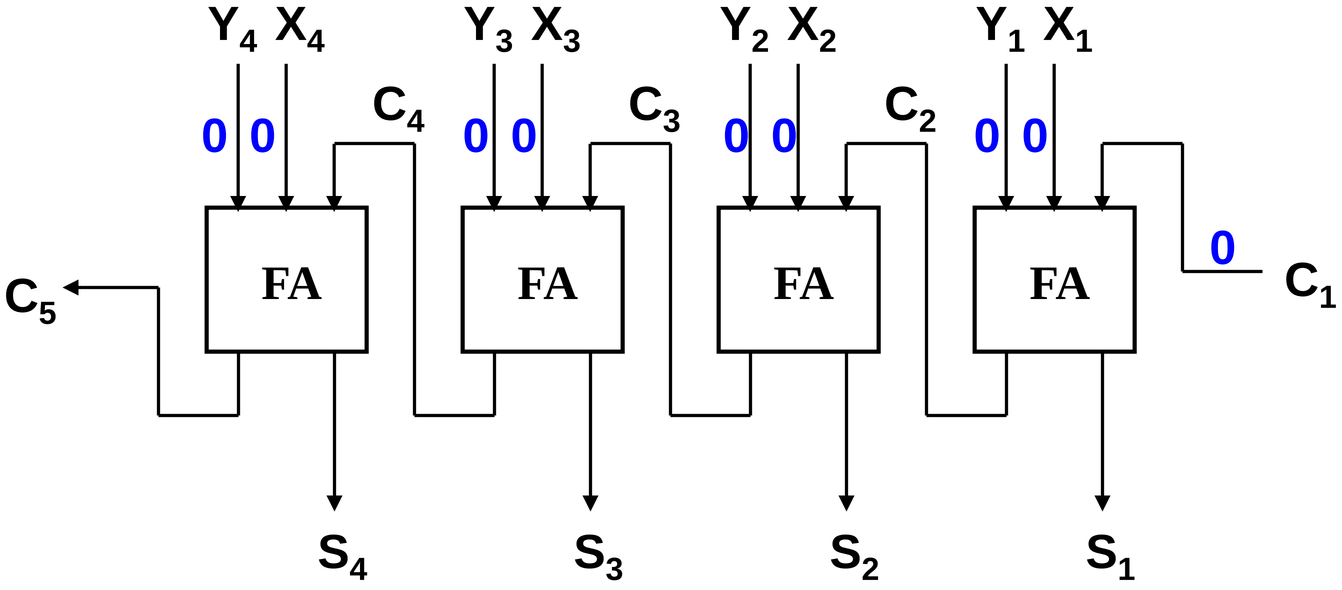
- As a simple example, consider the full adder circuit where all inputs are available at time 0. (Assume each gate has delay t.)



where outputs **S** and **C**, experience delays of 2t and 3t, respectively.

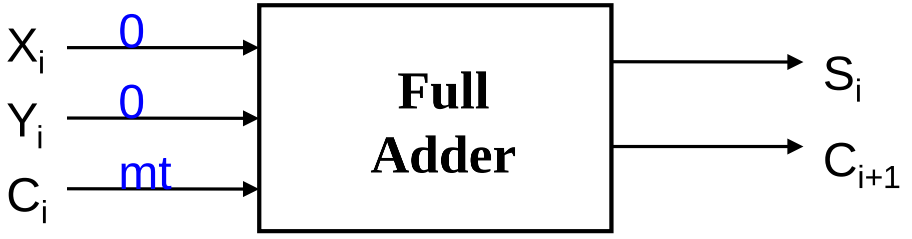
Calculation of Circuit Delays (3/5)

- More complex example: 4-bits parallel adder.



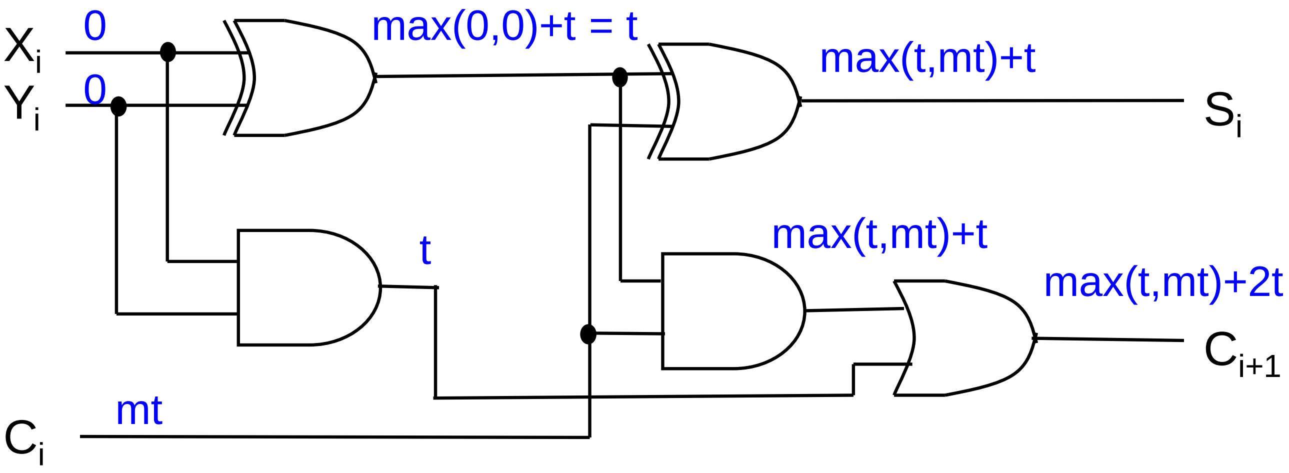
Calculation of Circuit Delays (4/5)

- 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 gives:



Calculation of 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:

  $S_n = ((n-1)*2+2)t$
  $C_{n+1} = ((n-1)*2+3)t$

  as their delay times.

- Propagation delay of ripple-carry parallel adders is proportional to the number of bits it handles.
- Maximum Delay: $((n-1)*2+3)t$

Faster Circuits

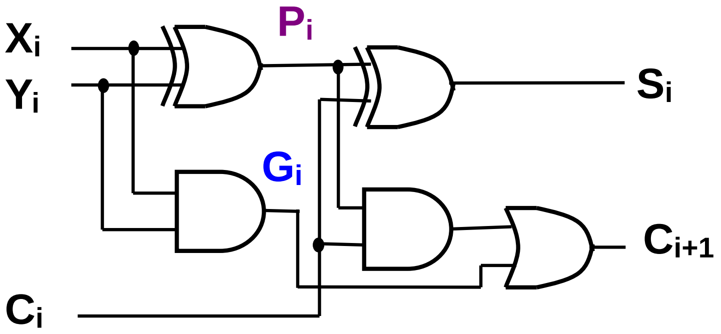
- Three ways of improving the speed of these circuits:
  - (i) Use better technology (e.g. ECL faster than TTL gates), BUT

    (a) faster technology is more expensive, needs more power, lower-level of integrations.

    (b) physical limits (e.g. speed of light, size of atom).
  - (ii) Use gate-level designs to two-level circuits! (use sum-of-products/product-of-sums) BUT
    - (a) complicated designs for large circuits.
    - (b) product/sum terms need MANY inputs!
  - (iii) Use clever look-ahead techniques BUT there are additional costs (hopefully reasonable).

Look-Ahead Carry Adder (1/6)

- Consider the full adder:



where intermediate signals are labelled as P_i, G_i, and defined as:

$P_i = X_i \oplus Y_i$

$G_i = X_i . Y_i$

- 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)

$C_{i+1} = G_i + P_i.C_i$ …(2)

- If you look at equation (2),

$G_i = X_i.Y_i$ is a *carry generate* signal

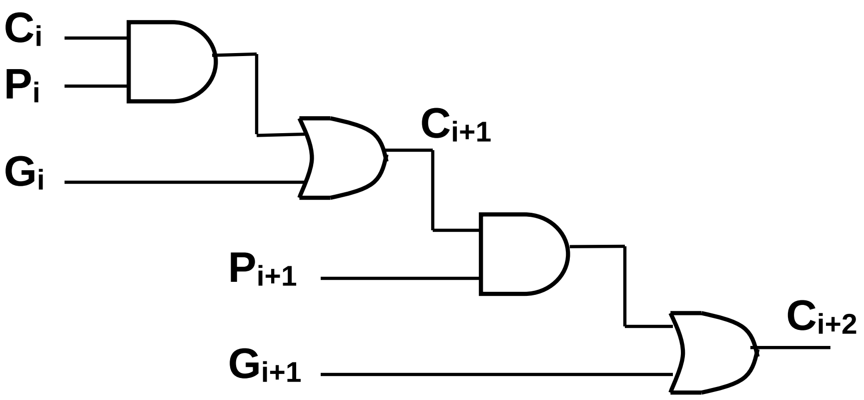
$P_i = X_i \oplus Y_i$ is a *carry propagate* signal

Look-Ahead Carry Adder (2/6)

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

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

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



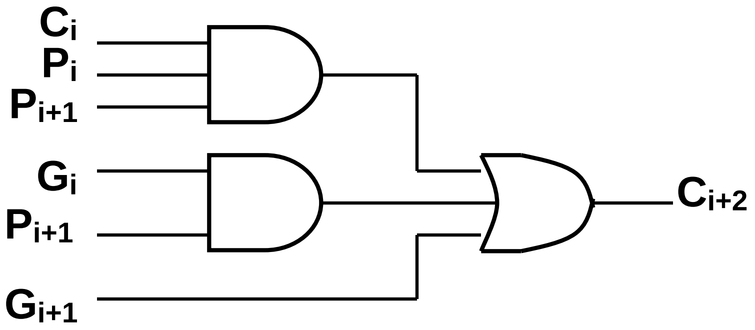
4-level circuit for $C_{i+2} = G_{i+1} + P_{i+1}.C_{i+1}$

Look-Ahead Carry Adder (3/6)

- Nested formula/gates cause ripple-carry propagation delay.
- Can reduce delay by expanding and flattening the formula for carries. For example, C_{i+2}

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

- New faster circuit for C_{i+2}



Look-Ahead Carry Adder (4/6)

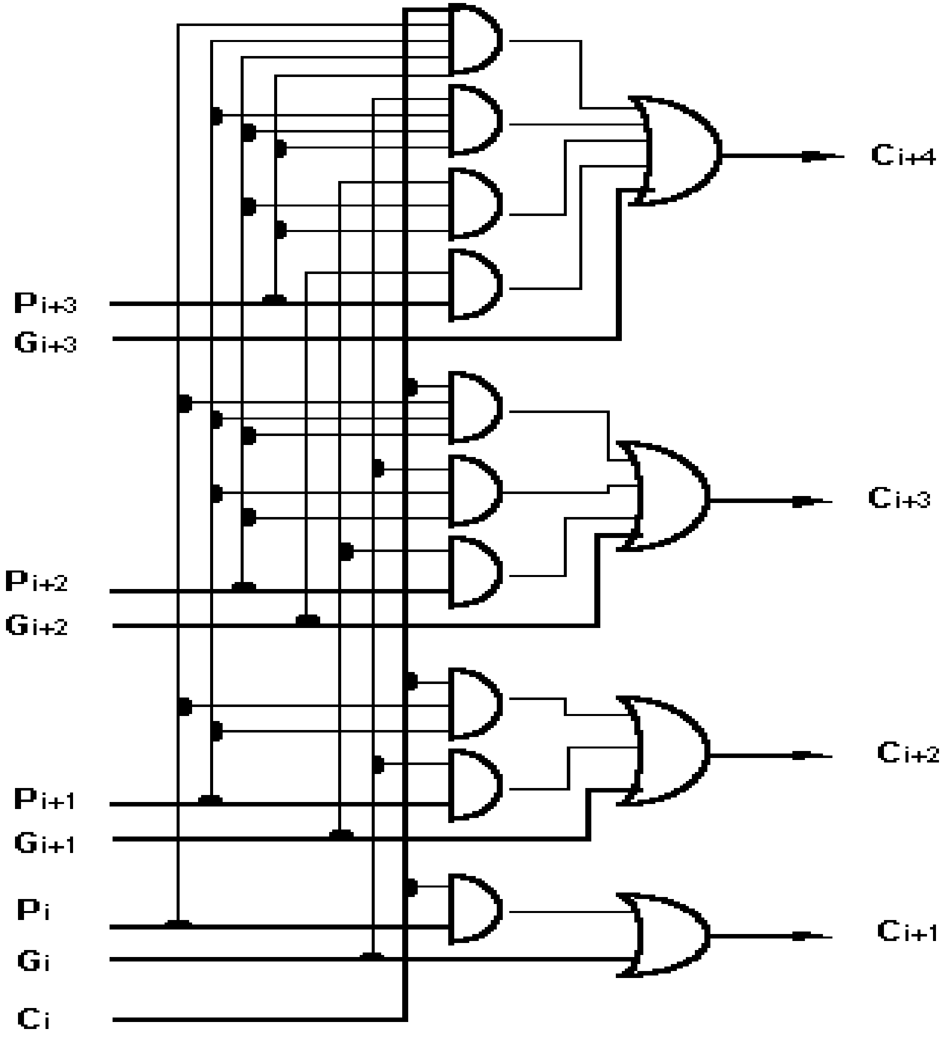
- Other carry signals can also be similarly flattened.

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

- Notice 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, $\mathbf{C_i}$.

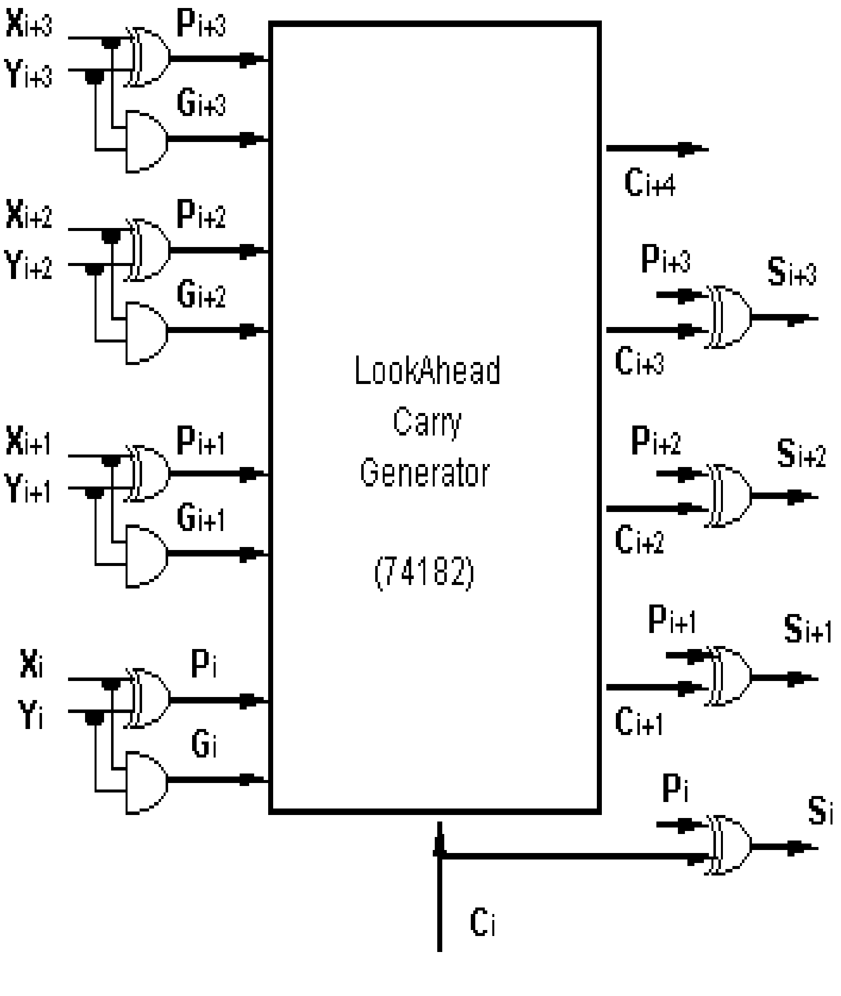
Look-Ahead Carry Adder (5/6)

- We employ the 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.
- Maximum propagation delay is 4t (t to get generate & propagate signals, 2t to get the carries and t for the sum signals) where t is the average gate delay.



4-Bit Adder with Look-Ahead (74181)