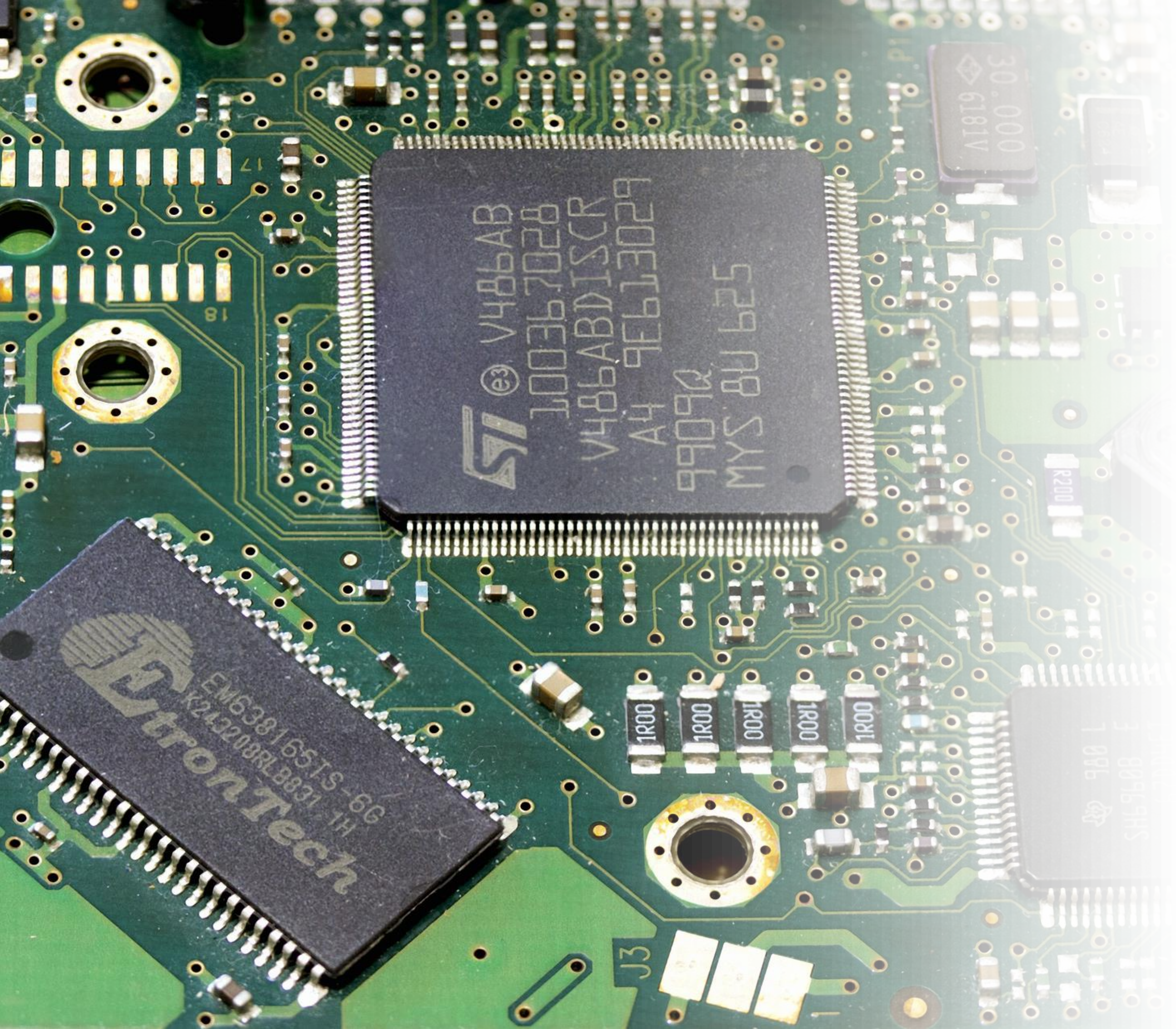


Digital Logic and System Design

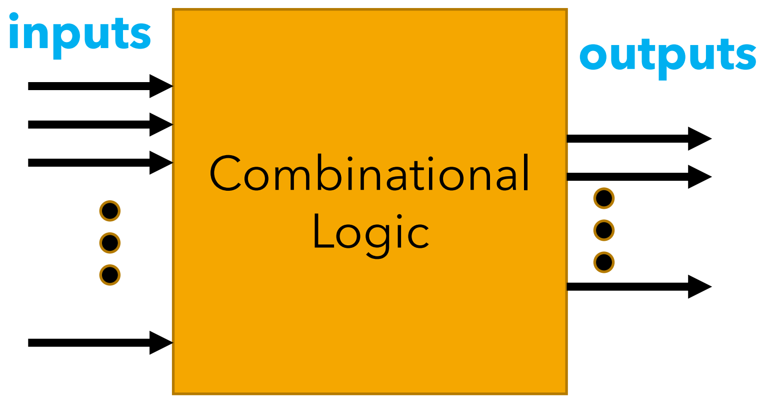
5. Combinational Logic

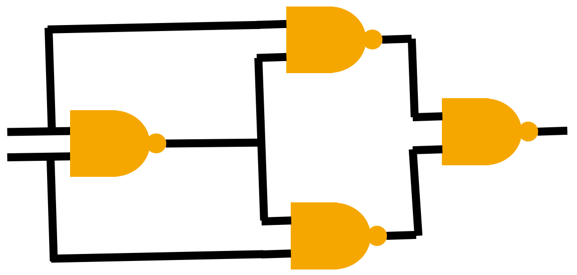
COL215, I Semester 2022-2023
Venue: LHC 111
'E' Slot: Tue, Wed, Fri 10:00-11:00

Instructor: Preeti Ranjan Panda
panda@cse.iitd.ac.in
www.cse.iitd.ac.in/~panda/
Dept. of Computer Science & Engg., IIT Delhi

Combinational Logic

- Output is function only of **present values** of inputs
- ...as opposed to **Sequential Logic**
  - where output could depend on **previous** values
- What netlists are NOT combinational?

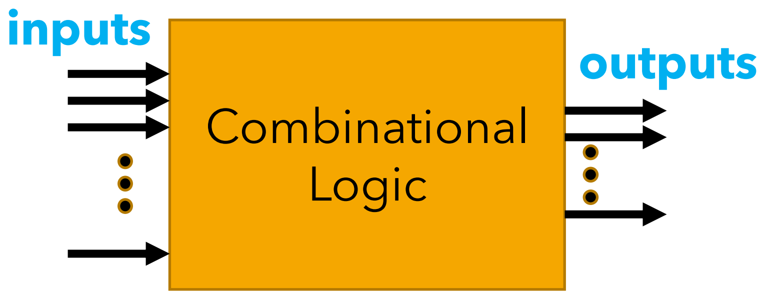




Example combinational circuit

Representing Combinational Logic

- Representing multiple outputs in Truth Table?
- K-Map representation?



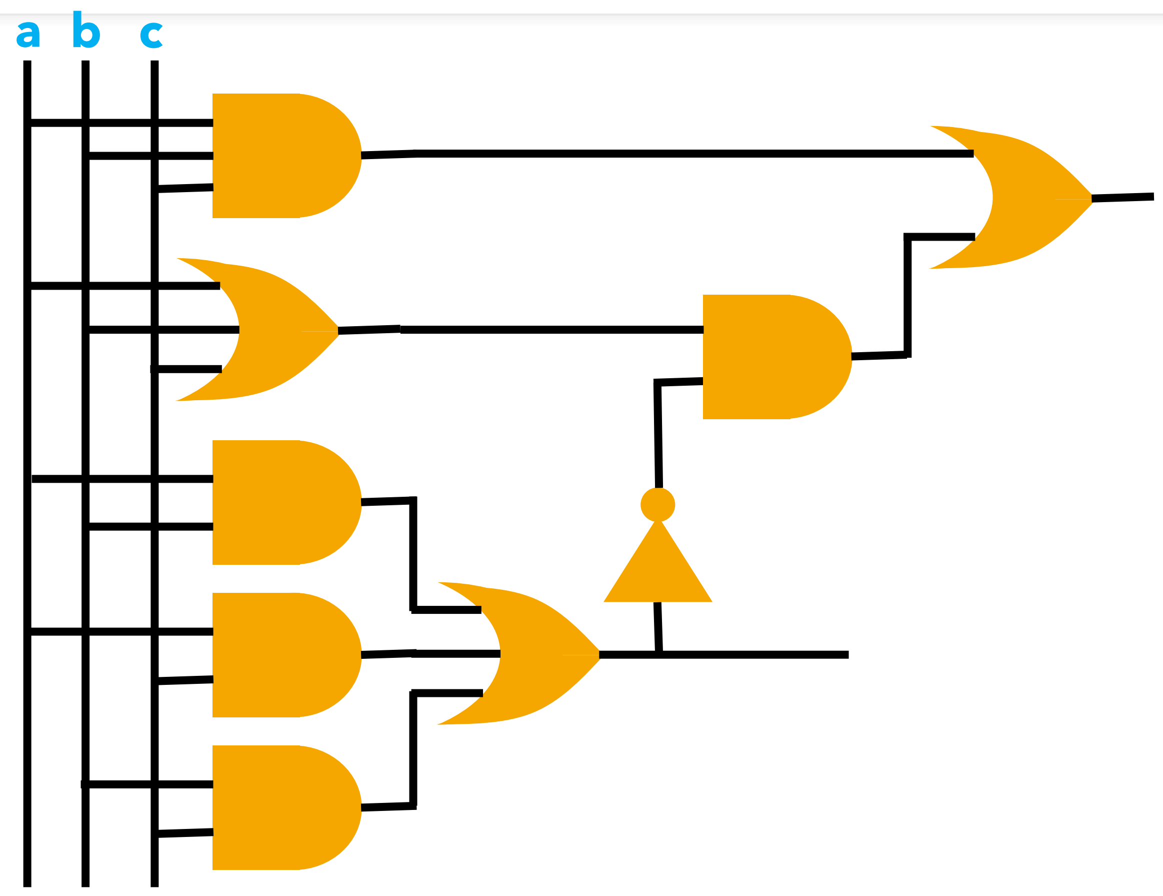
x y z	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

Tasks with Combinational Logic Circuits

- Analyse the behaviour of a logic circuit
- Synthesise a circuit for a given behaviour
  - Manually
  - Specify using Hardware Description Language (HDL)
- Study standard combinational circuits
  - Arithmetic operations (addition, multiplication,...)

Analysing a Combinational Circuit (Netlist)

- What Boolean function does a gate netlist implement?
- Follow the netlist from inputs to output
    - identify Boolean functions at intermediate stages



Synthesising a Combinational Circuit

- Capturing informal **specification** in precise language
- Identify **input** and **output** variables
- **Represent** the logic
  - Truth tables
  - Boolean expressions
- **Simplify** Boolean expressions
- Implement gate netlist
- **Verify**: simulation

Example Design: Gray Code Converter

- Specification:
  Given a 3-bit Binary Code, convert to Gray Code

	Binary Code	Gray Code
0:	000	000
1:	001	001
2:	010	011
3:	011	010
4:	100	110
5:	101	111
6:	110	101
7:	111	100

Example: Inputs and Outputs, Representation

Inputs	Outputs
a b c	x y z
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

a \ bc	00	01	11	10
0	0	0	0	0
1	1	1	1	1

x (a, b, c)

a \ bc	00	01	11	10
0	0	0	1	1
1	1	1	0	0

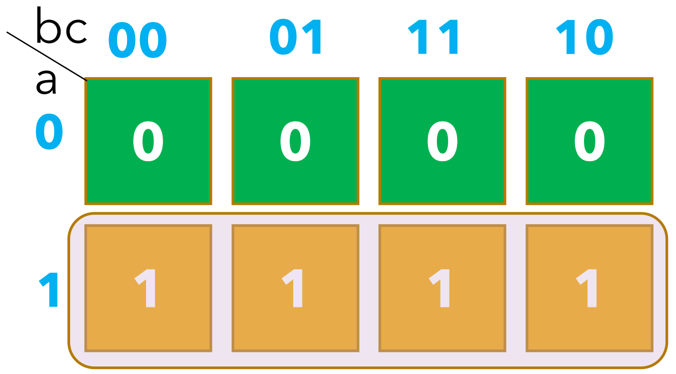
y (a, b, c)

a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	0	1

z (a, b, c)

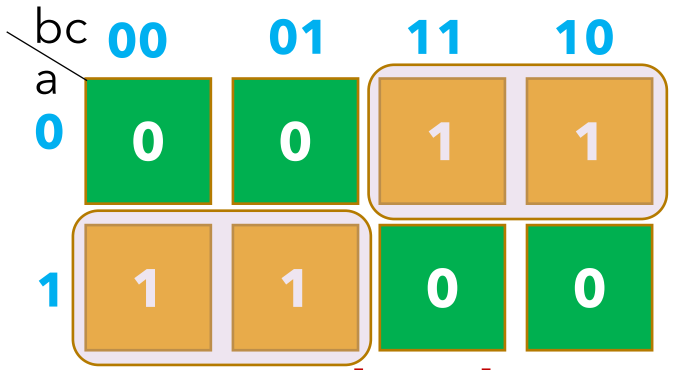
Example: Boolean Simplification

Inputs	Outputs
a b c	x y z
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100



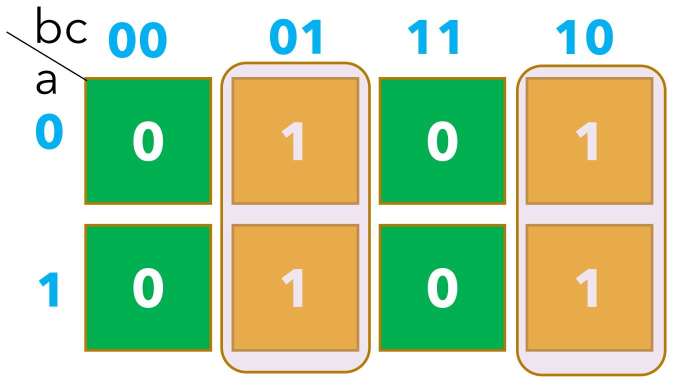
a \ bc	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$x = a$



a \ bc	00	01	11	10
0	0	0	1	1
1	1	1	0	0

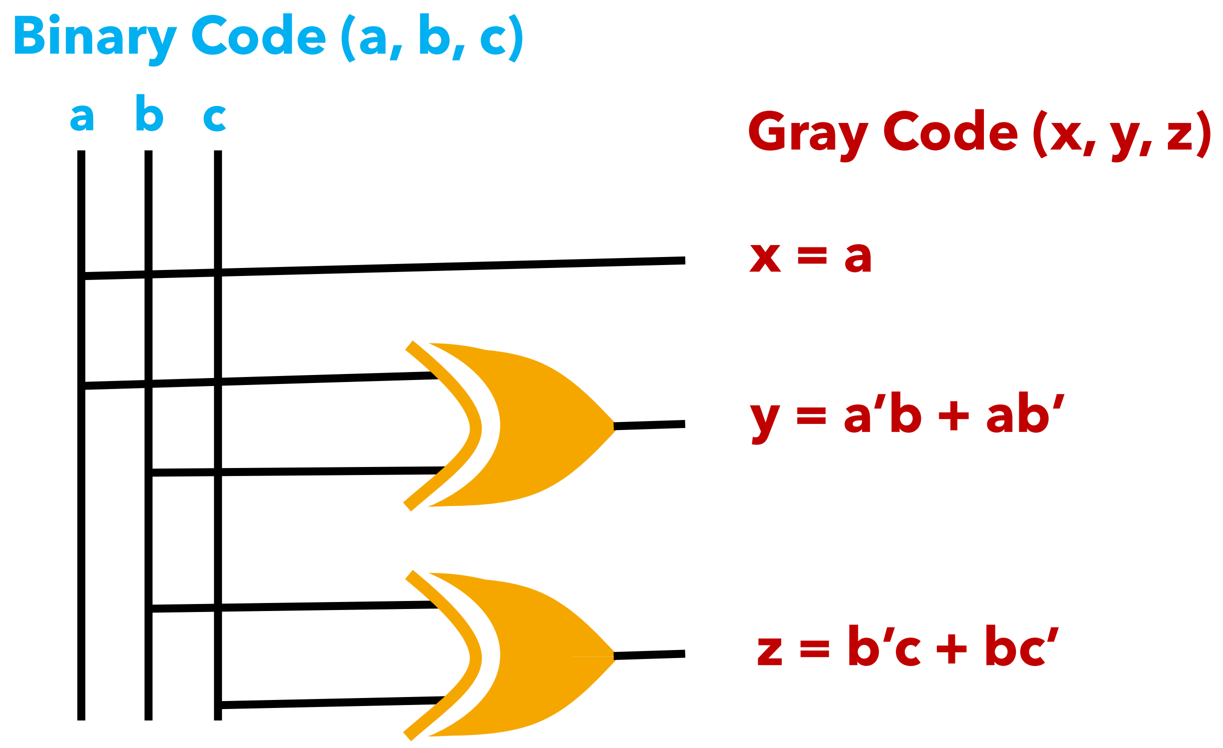
$y = a'b + ab'$



a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$z = b'c + bc'$

Gate Implementation



Designing a 1-bit Adder

- **Specification**: single-bit binary addition
- **Inputs**: x, y
- **Outputs**: sum (s), carry (c)
- Truth Table
- Boolean simplification

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline =\ 0 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline =\ 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline =\ 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline =\ 10 \end{array}$$

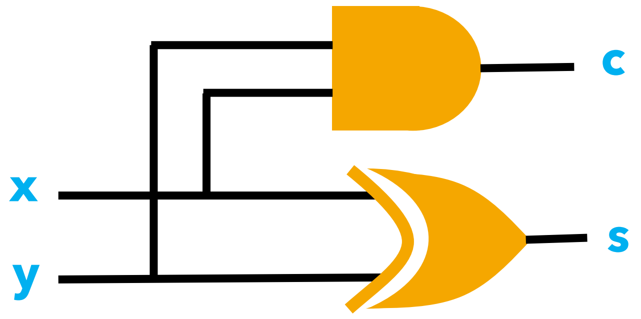
x y	c s
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

Adder: Simplification and Implementation

- Boolean simplification
- Gate implementation

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$c = xy$$
$$s = x'y + xy' = x \oplus y$$

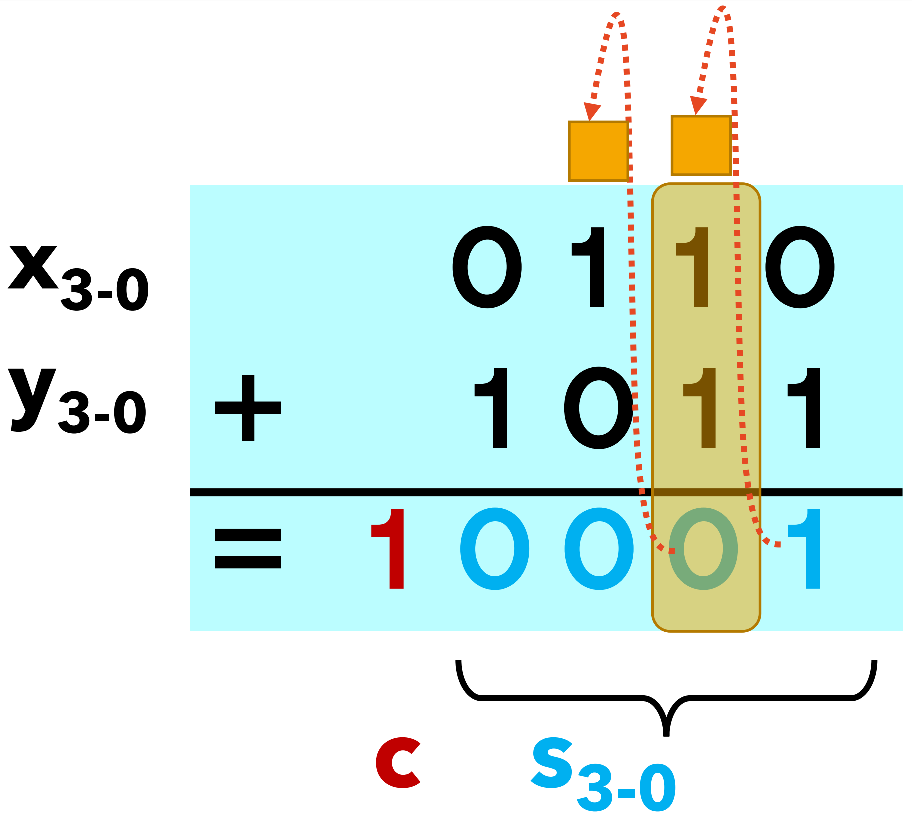


4-bit Adder

- **Specification**: 4-bit binary addition
- **Inputs**: $\mathbf{x_{3-0}}$, $\mathbf{y_{3-0}}$
- **Outputs**: sum (s_{3-0}), carry (**c**)
- Truth Table?
- **Composing larger designs out of smaller ones**

$$\begin{array}{r l} x_{3-0} & \quad\ 0\,1\,1\,0 \\ y_{3-0} & +\quad 1\,0\,1\,1 \\ \hline & =\ \underset{c}{1}\,\underbrace{0\,0\,0\,1}_{s_{3-0}} \end{array}$$

Identify repeating pattern

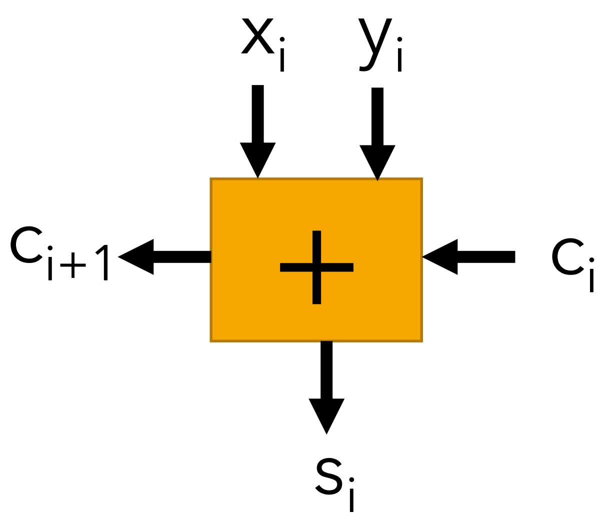


At each bit position i:

Inputs: x_i, y_i, c_i

Outputs: s_i, c_{i+1}

x_i y_i c_i	c_{i+1}	s_i
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



Full Adder

Boolean Function for Full Adder

At each bit position i:

Inputs: a, b, c

Outputs: c′, s

a b c	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

Full Adder

a \ bc	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Carry Out:

c′ = ab + bc + ca

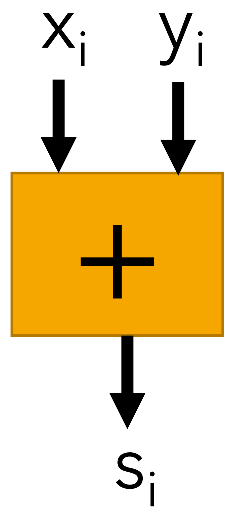
a \ bc	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Sum:

$$\begin{aligned} s &= ab'c' + a'b'c + a'bc' + abc \\ &= a\,(bc + b'c') + a'(b'c + bc') \\ &= a \oplus b \oplus c \end{aligned}$$

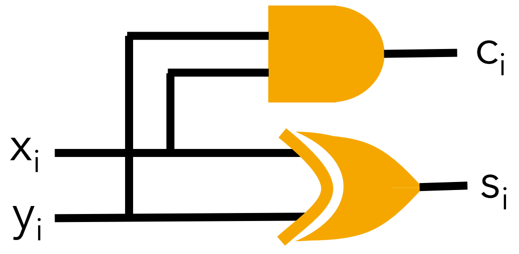
Half Adder vs. Full Adder

Half Adder

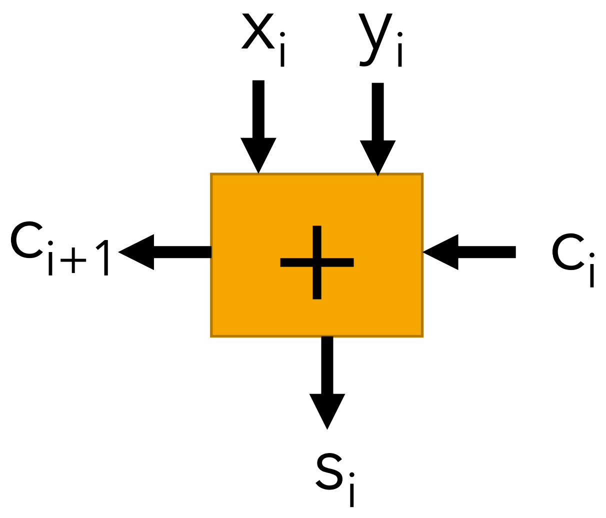


x_i y_i	c_i s_i
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

$$s_i = x_i'y_i + x_i y_i' = x_i \oplus y_i$$
$$c_i = x_i y_i$$

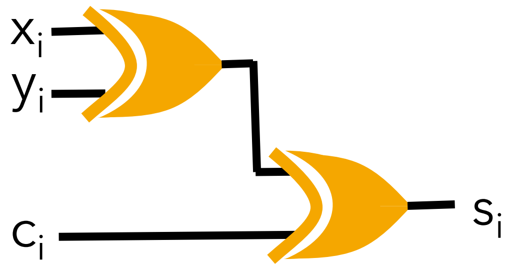


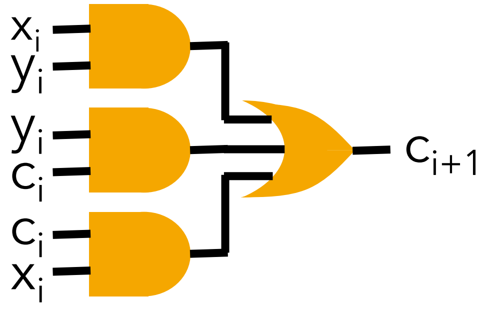
Full Adder



x_i y_i c_i	c_{i+1}	s_i
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

$$s_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$





Ripple Carry Adder (RCA)

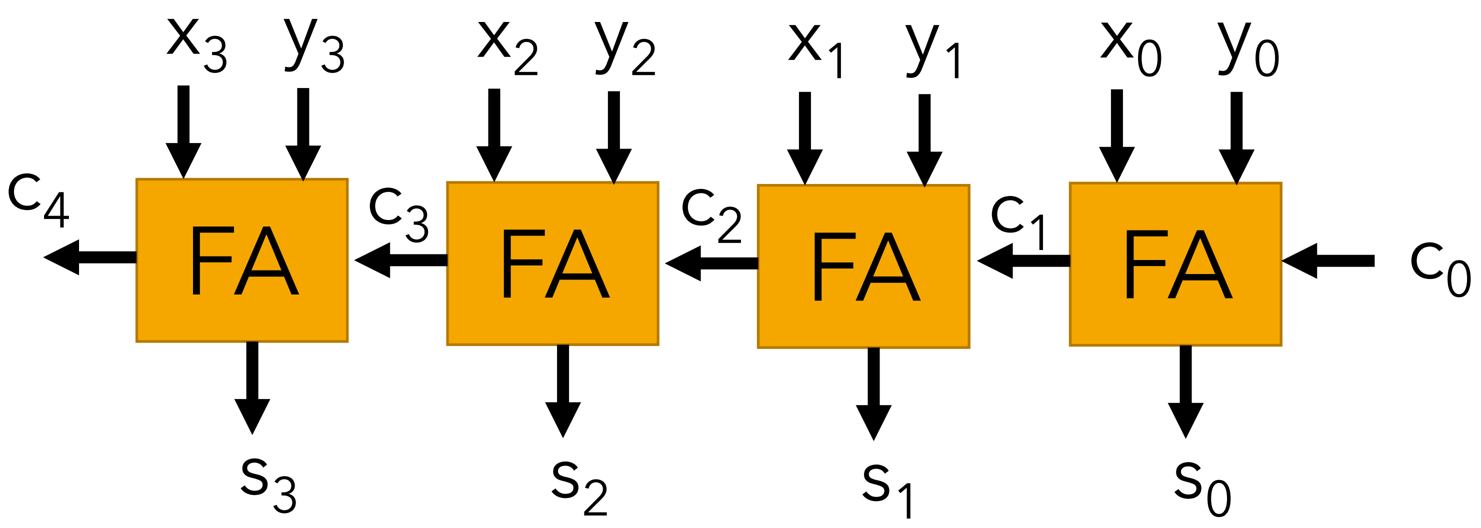
At each bit position i:

Inputs: x_i, y_i, c_i

Outputs: s_i, c_{i+1}

x_i	y_i	c_i	c_{i+1}	s_i
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

Full Adder



Chain of Full Adders

Adder delay analysis

- How many gate levels for final output?
- Delay for n-bit RCA?
- What if Full Adder **Sum** and **Carry** delays were different?
  - e.g., Sum: 8 ns and Carry: 5 ns
- Can we make it faster?
  - Use **faster gates** on Carry propagation path
  - Partial computation ahead of time: **Carry Lookahead**

Carry In and Out in Full Adder

- **Carry Generation**: When do we **generate** a carry out irrespective of input carry?
    - carry_out = 1 irrespective of carry_in values
- **Carry Propagation**: When do we **propagate** an input carry to the output irrespective of input values?
    - carry = carry_in irrespective of x, y values

x_i	y_i	c_i	c_{i+1}	s_i
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

Full Adder

$$G_i = x_i y_i$$
$$P_i = x_i \oplus y_i$$

Using Propagate and Generate Values

- **Sum** and **Carry_out** can be derived from P_i and G_i values
- **1 logic level** to generate P_i and G_i
  - treating AND and XOR as 1 gate level
- **1 logic level** to generate **Sum**

$s_i = x_i \oplus y_i \oplus c_i$
$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$

$G_i = x_i y_i$
$P_i = x_i \oplus y_i$

$s_i = P_i \oplus c_i$
$c_{i+1} = G_i + P_i c_i$ (verify)

Carry Lookahead Logic

$$c_{i+1} = G_i + P_i c_i$$

$c_1 = G_0 + P_0 c_0$

$c_2 = G_1 + P_1 c_1 = G_1 + P_1(G_0 + P_0 c_0) = G_1 + P_1 G_0 + P_1 P_0 c_0$

$c_3 = G_2 + P_2 c_2 = G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 c_0) = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$

$c_4 = G_3 + P_3 c_3 = G_3 + P_3(G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0)$
$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$

- **2 logic levels to generate c_4 from c_0**
- Approx: 5 i/p gate has same delay as 2 i/p gate

4-bit Carry Lookahead Adder (CLA)

$G_i = x_i y_i$ $s_i = P_i \oplus c_i$

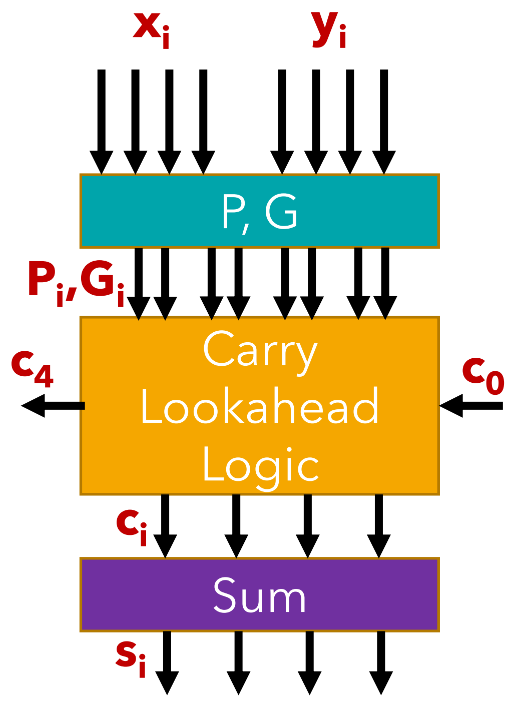
$P_i = x_i \oplus y_i$ $c_{i+1} = G_i + P_i c_i$

$c_1 = G_0 + P_0 c_0$

$c_2 = G_1 + P_1 G_0 + P_1 P_0 c_0$

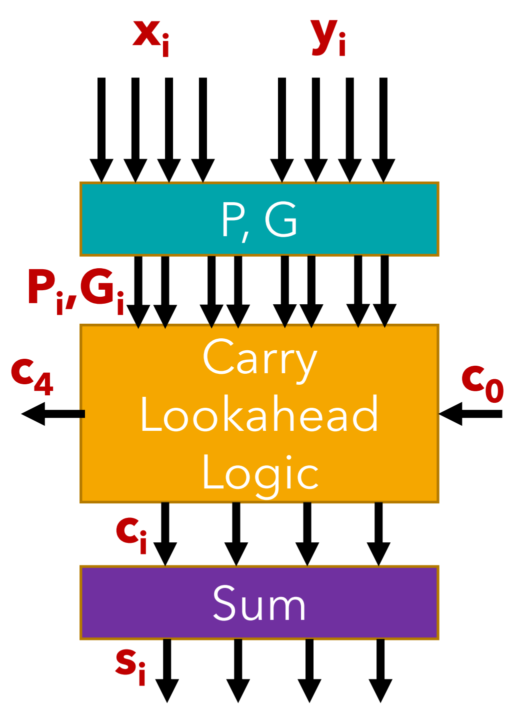
$c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$

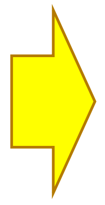
$c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$

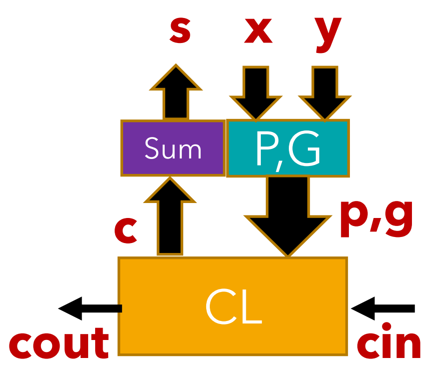


- **2 logic levels to generate c_4 from c_0**
- Approx: 5 i/p gate has same delay as 2 i/p gate
- **1 logic level to generate all P_i and G_i**
- **1 logic level to generate all sums s_i**
- 4-bit Adder delay: **1+2+1 = 4 levels**

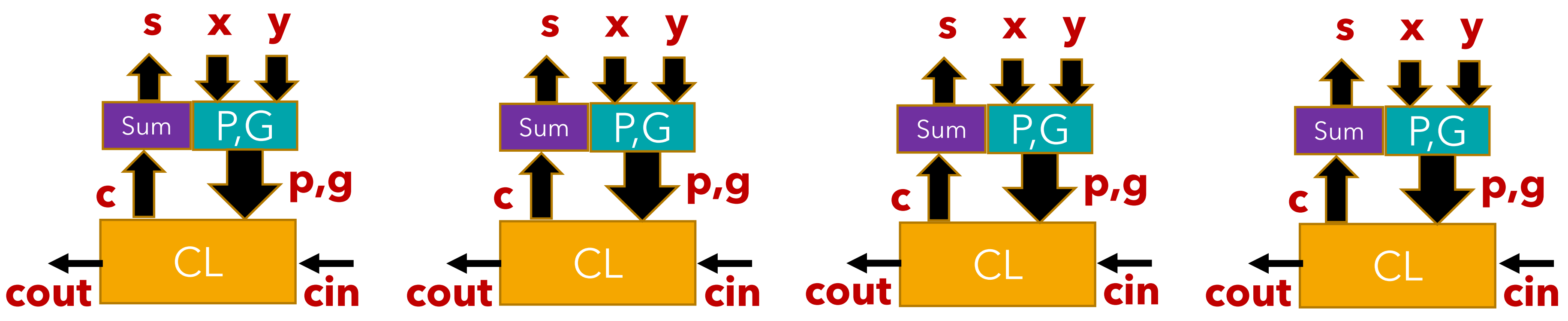
4-bit CLA: Simplified Diagram







16-bit Adder from 4-bit CLA

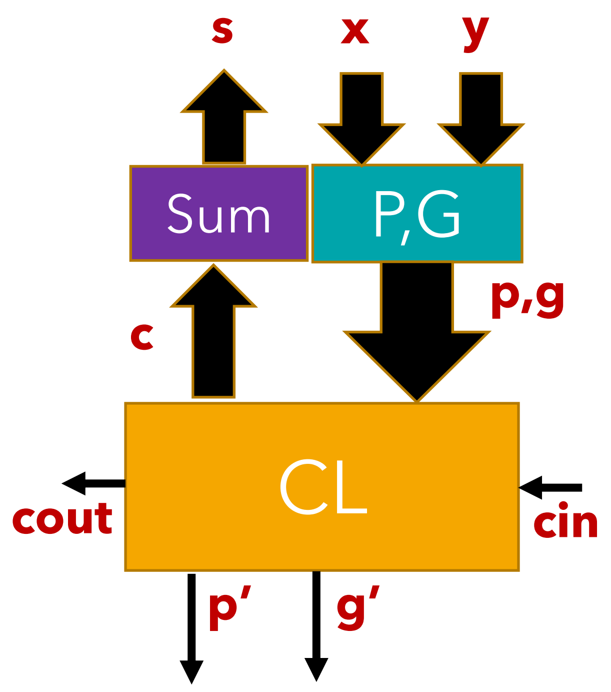


$g_i = x_i y_i$
$p_i = x_i \oplus y_i$

$s_i = p_i \oplus c_i$
$c_{i+1} = g_i + p_i c_i$

How do we extend the structure?

CL block-level carry propagate/generate

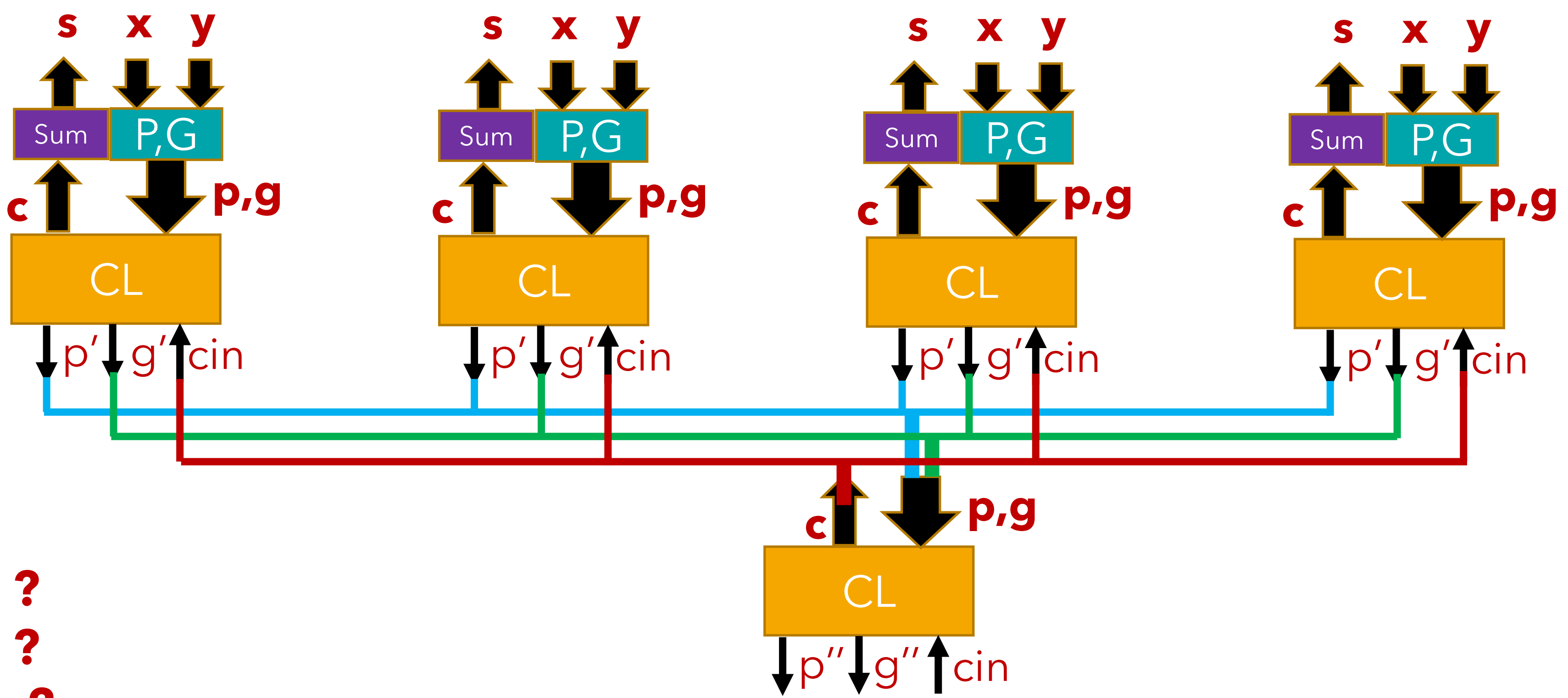


$g_i = x_i y_i$
$p_i = x_i \oplus y_i$

$s_i = p_i \oplus c_i$
$c_{i+1} = g_i + p_i c_i$

$p' = ?$
$g' = ?$

16-bit Adder from 4-bit CLA

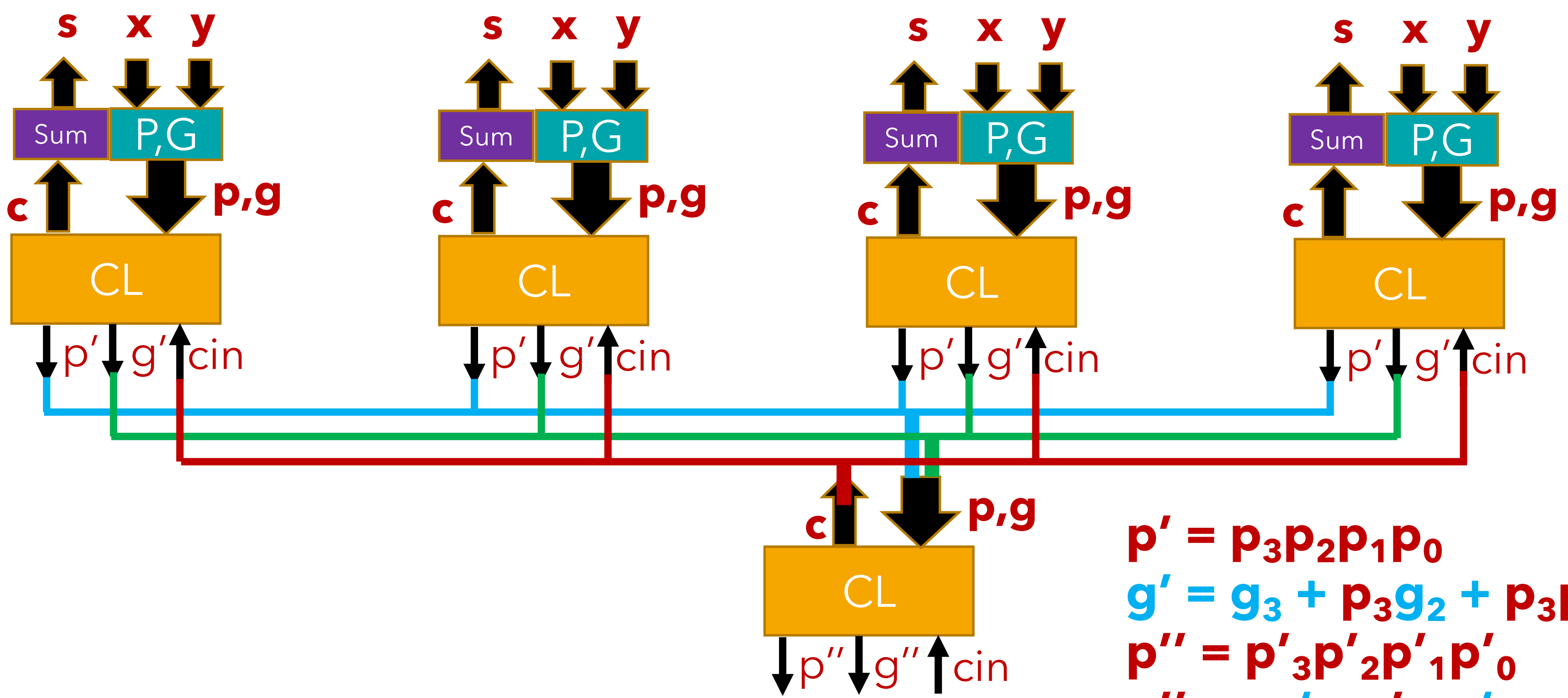


$g_i = x_i y_i$
$p_i = x_i \oplus y_i$

$s_i = p_i \oplus c_i$
$c_{i+1} = g_i + p_i c_i$

p′ = ?
g′ = ?
p″ = ?
g″ = ?

16-bit Adder from 4-bit CLA



$g_i = x_i y_i$

$p_i = x_i \oplus y_i$

$s_i = p_i \oplus c_i$

$c_{i+1} = g_i + p_i c_i$

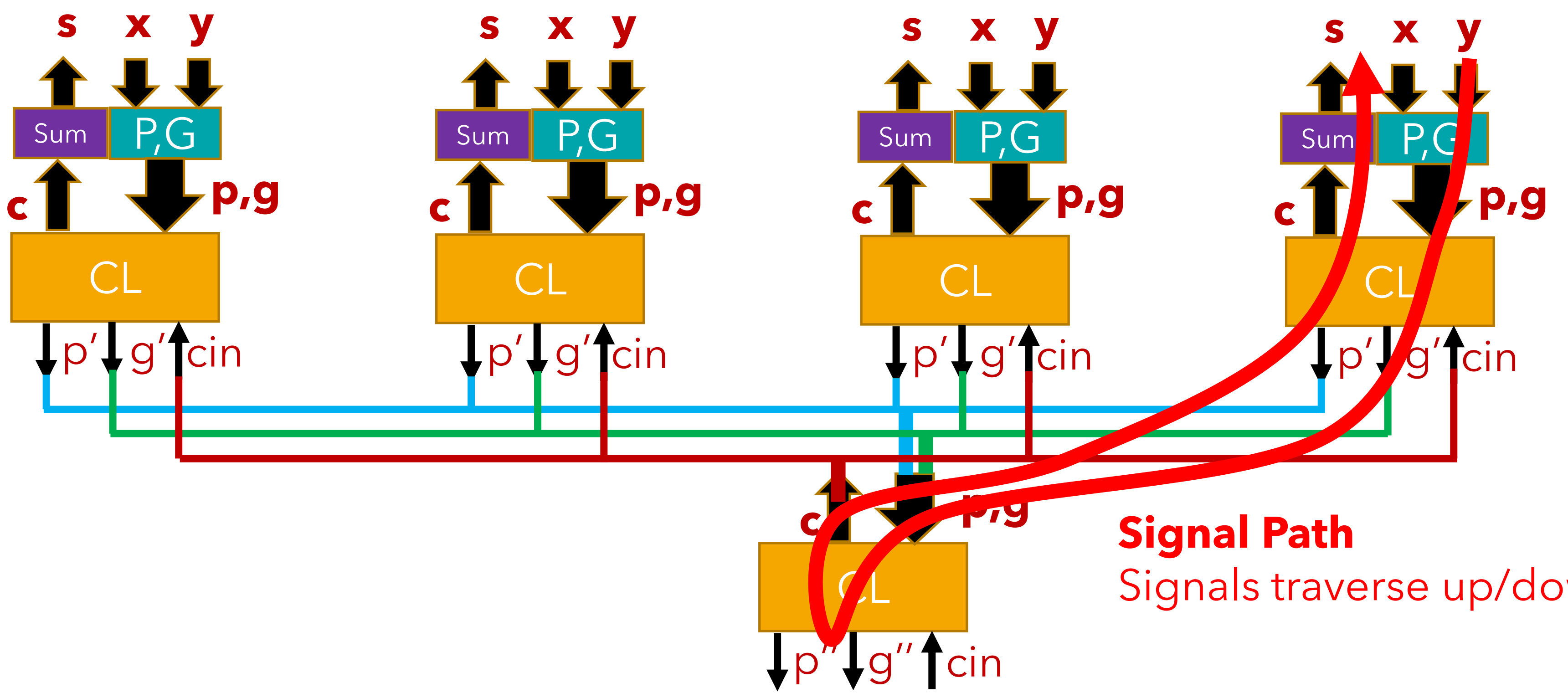
$p' = p_3 p_2 p_1 p_0$

$g' = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

$p'' = p'_3 p'_2 p'_1 p'_0$

$g'' = g_3' + p'_3 g_2' + p_3' p_2' g_1' + p_3' p_2' p_1' g_0'$

16-bit Adder from 4-bit CLA: Delay Analysis



$g_i = x_i y_i$

$p_i = x_i \oplus y_i$

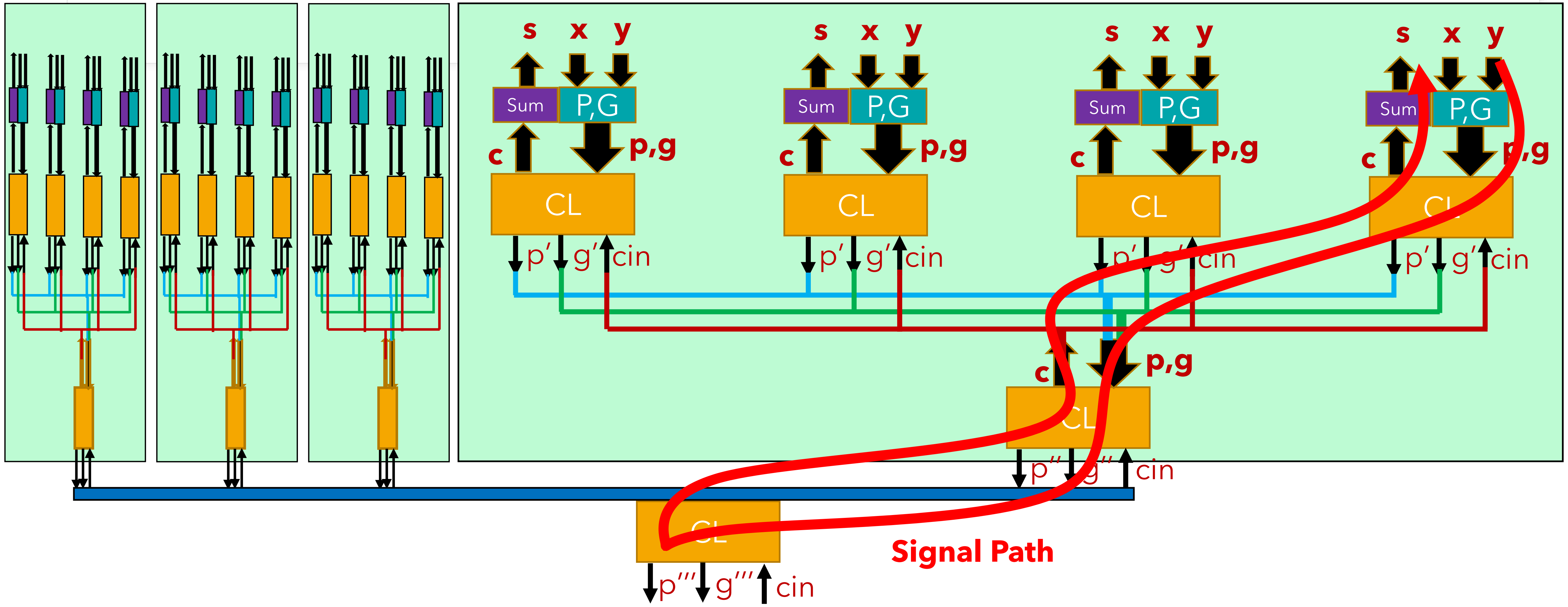
$s_i = p_i \oplus c_i$

$c_{i+1} = g_i + p_i c_i$

Signal Path

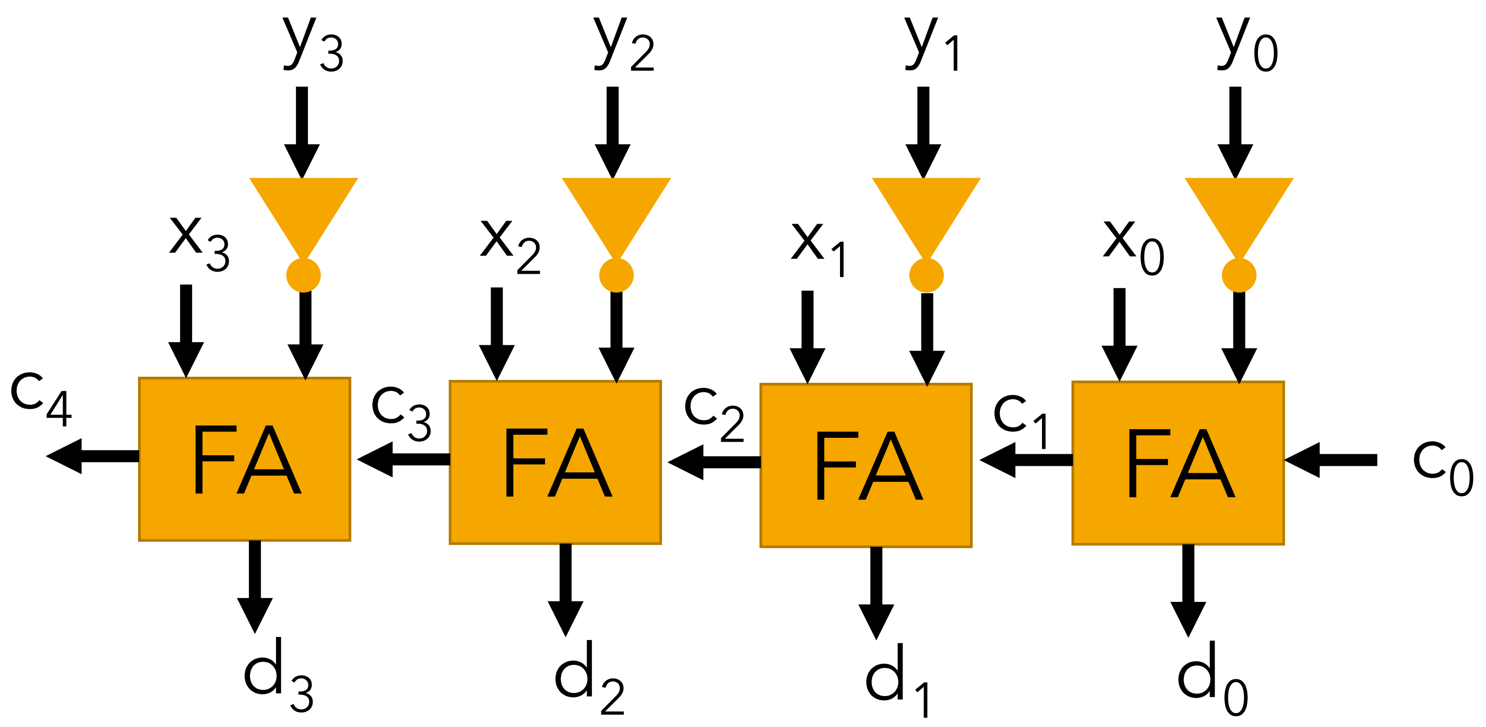
Signals traverse up/down, not left/right

64-bit Adder from 4-bit CLAs



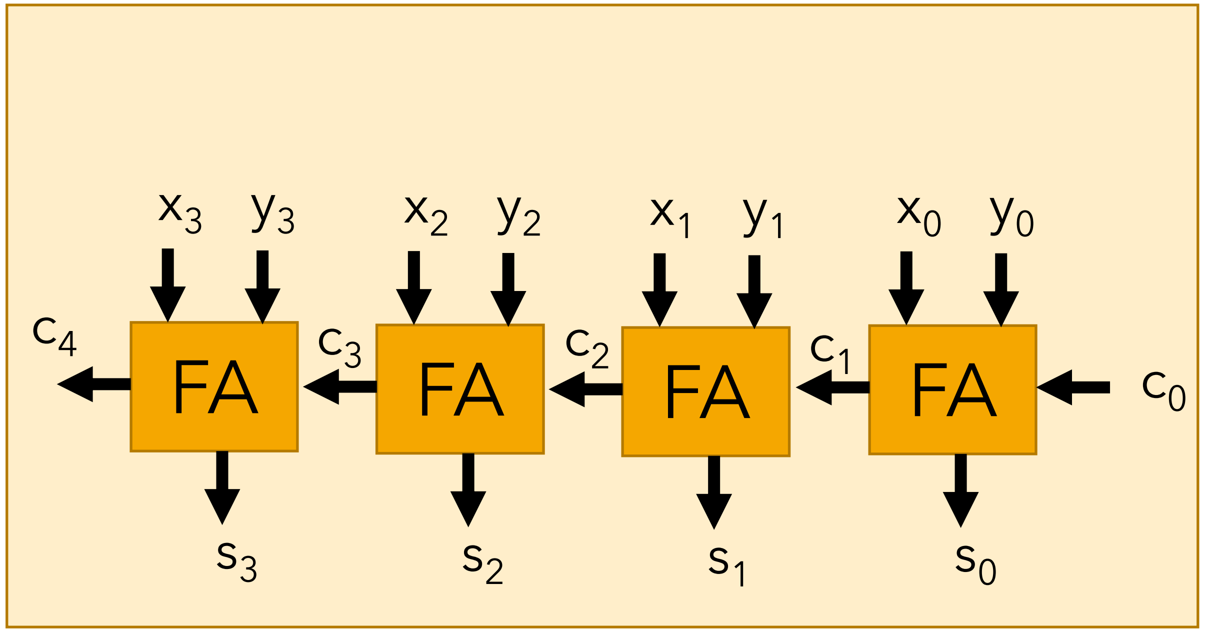
n-bit Subtraction

- **$d = x - y$**
- $d = \mathbf{x + (-y)}$
- $-y$: 2's complement of y
- $-y$: **$y' + 1$**
- y': **inverter**
- How do we **add 1**?

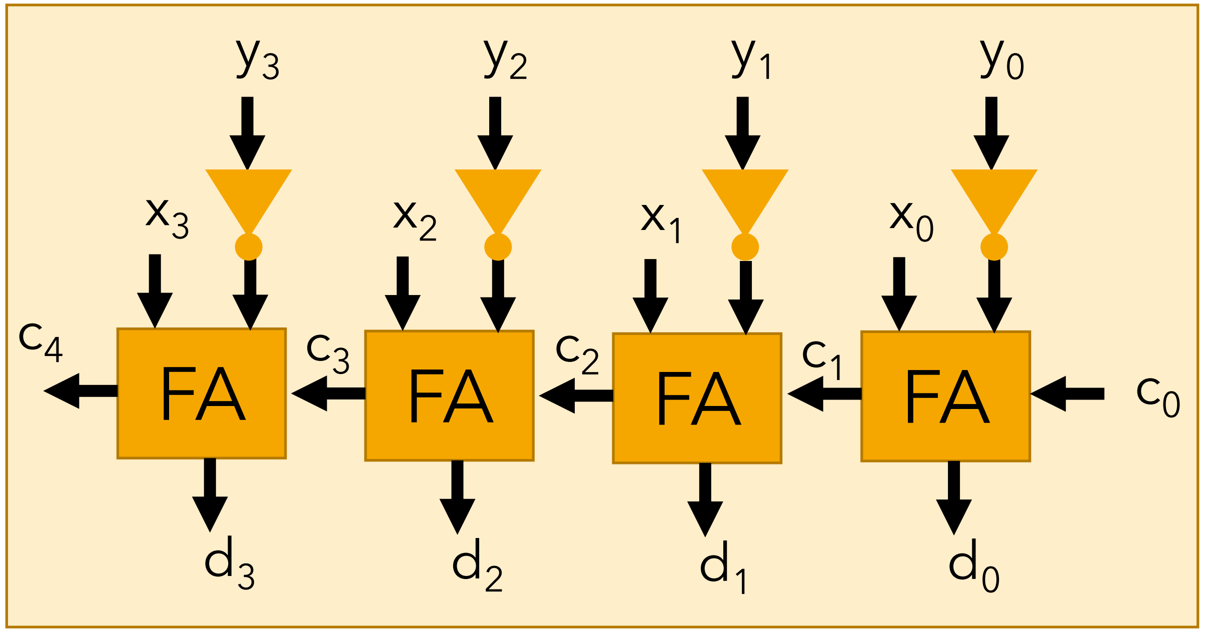


Programmable Adder/Subtractor

Adder

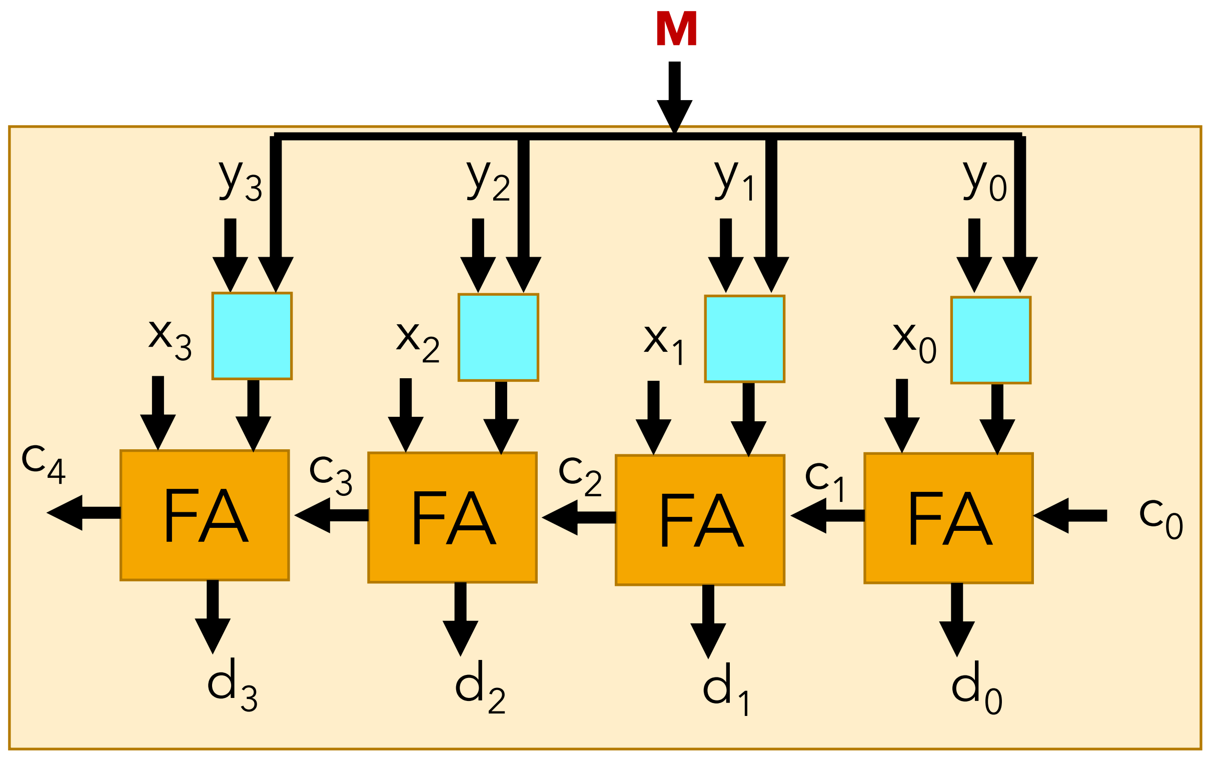


Subtractor

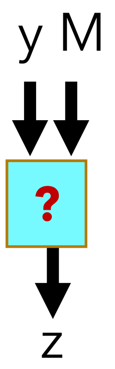


Very similar!
Can we combine into one structure?

Programmable Adder/Subtractor

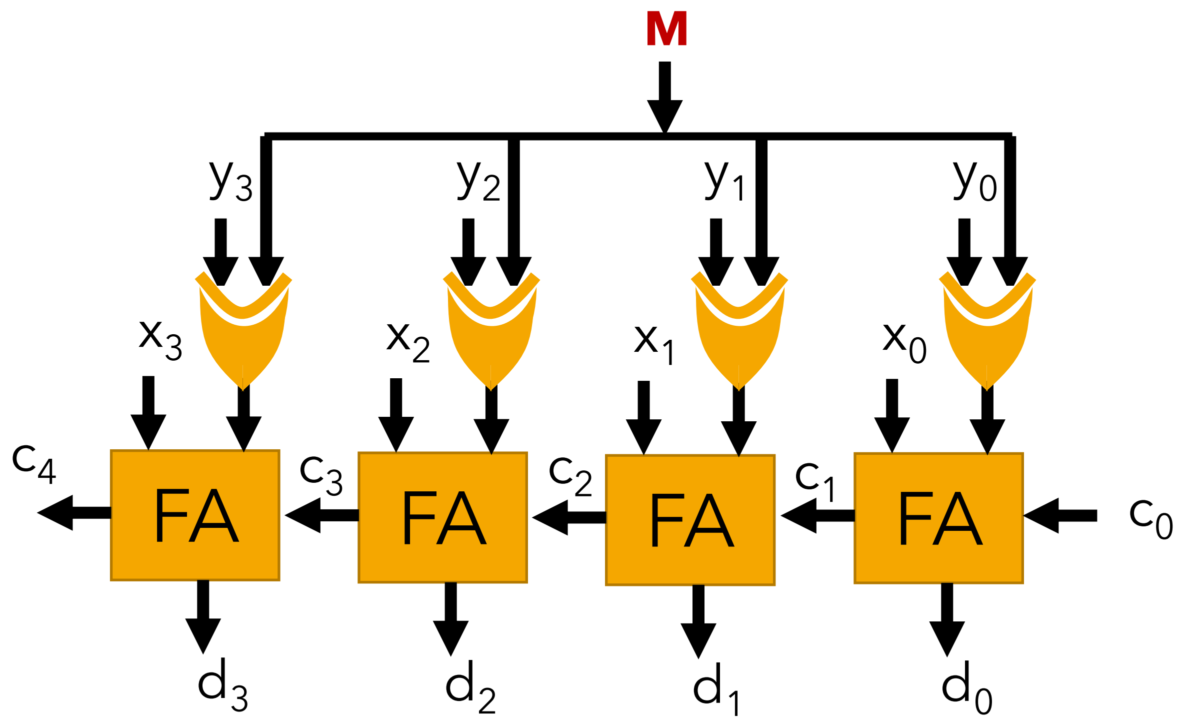


M = 0: Add
M = 1: Subtract

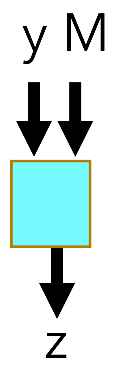


M = 0: z = y
M = 1: z = y′
What function is z (y, M)?

Programmable Adder/Subtractor



M = 0: Add
M = 1: Subtract



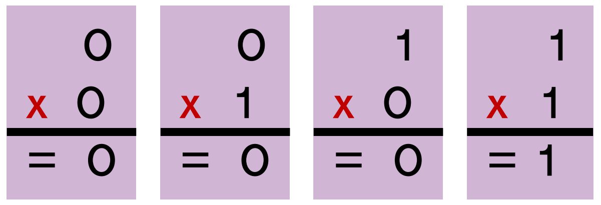
M = 0: z = y
M = 1: z = y′
What function is z (y, M)?

y	M	z
0	0	0
0	1	1
1	0	1
1	1	0

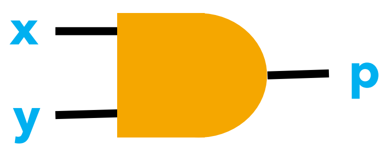
$\mathbf{z = M \oplus y}$

Binary Multiplier

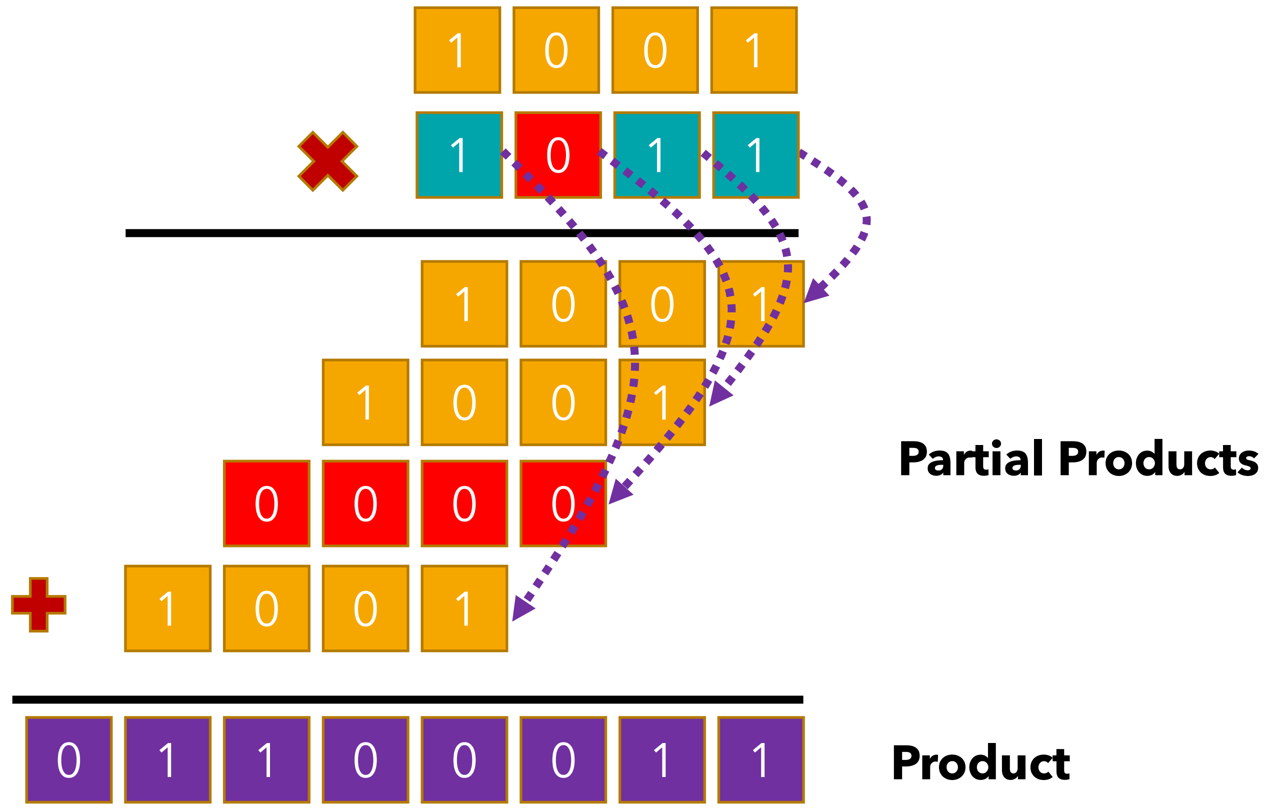
1x1 Multiplier



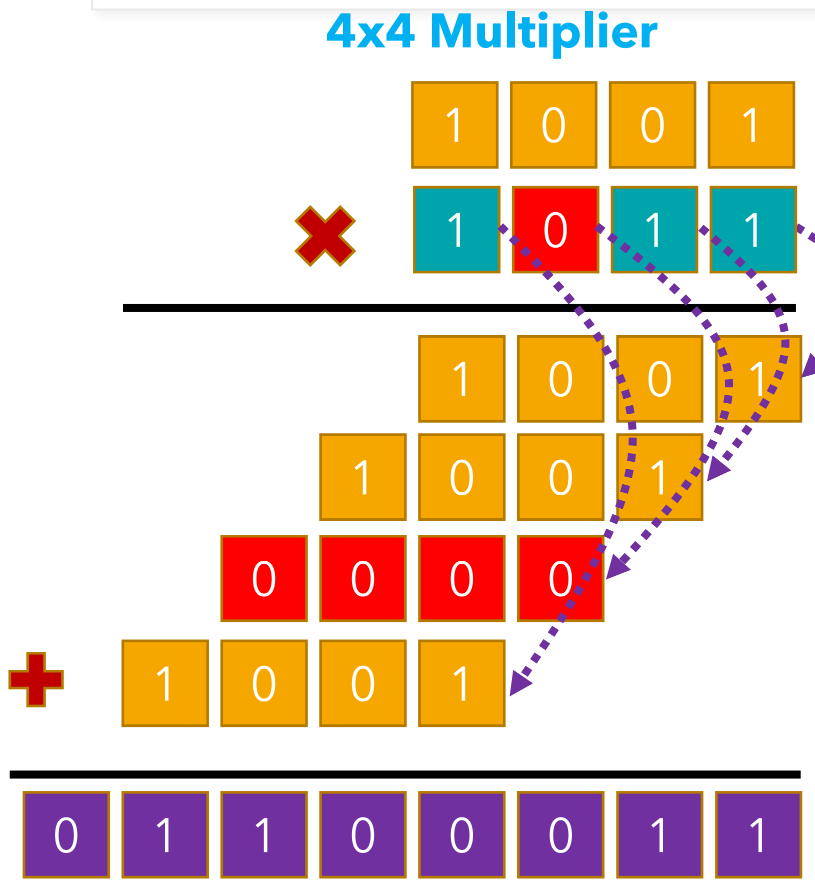
x	y	p
0	0	0
0	1	0
1	0	0
1	1	1

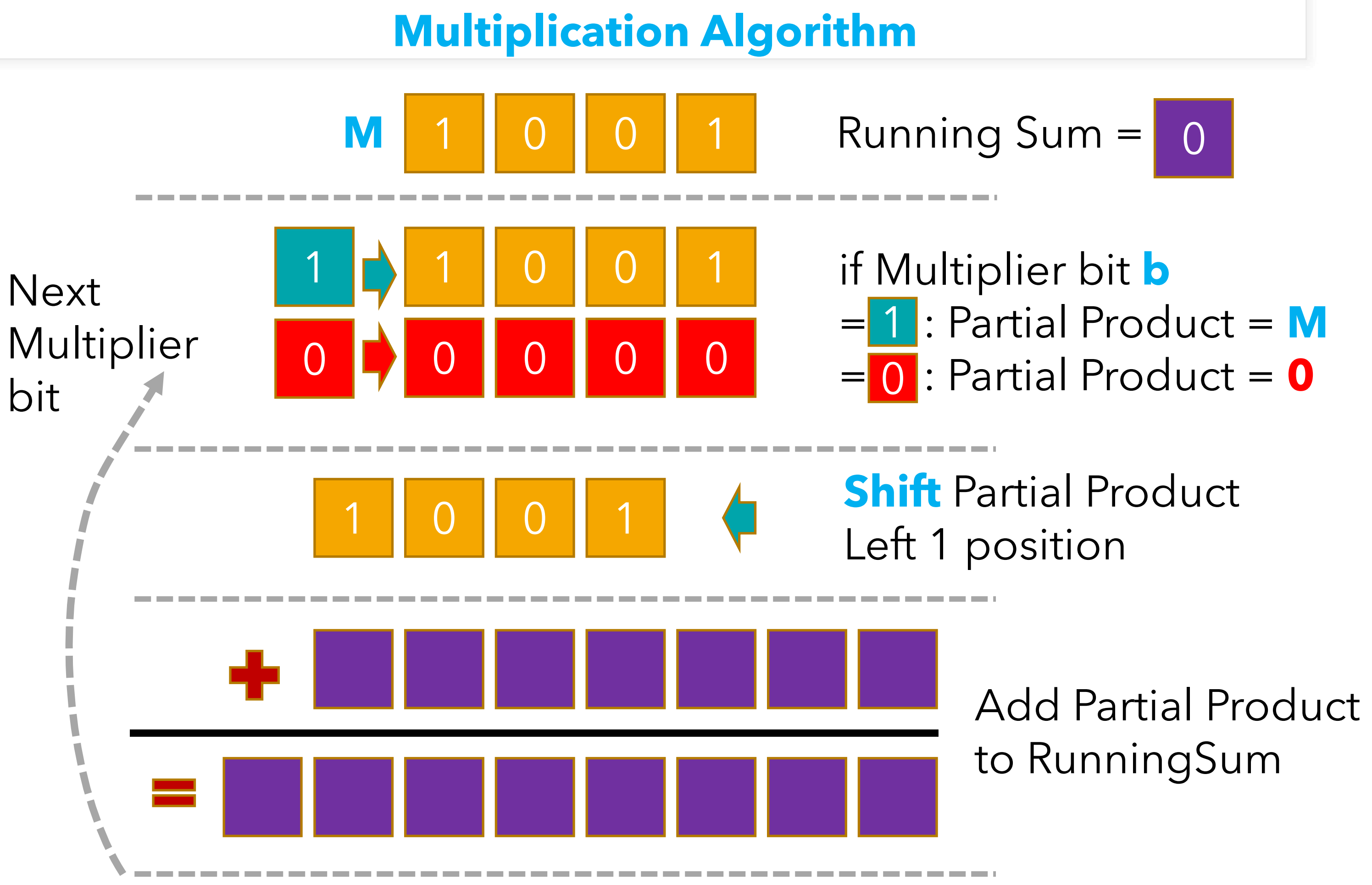


4x4 Multiplier

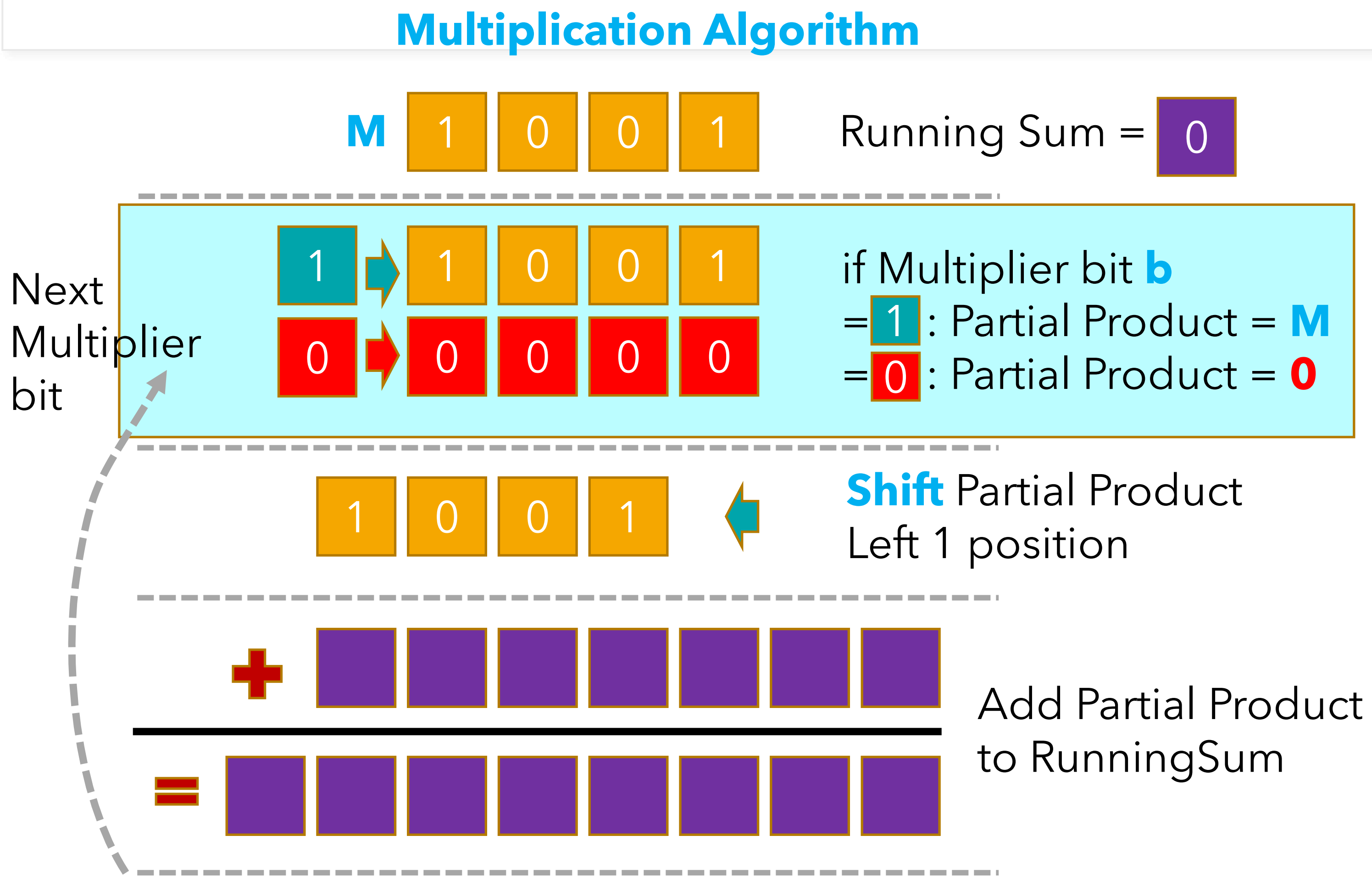


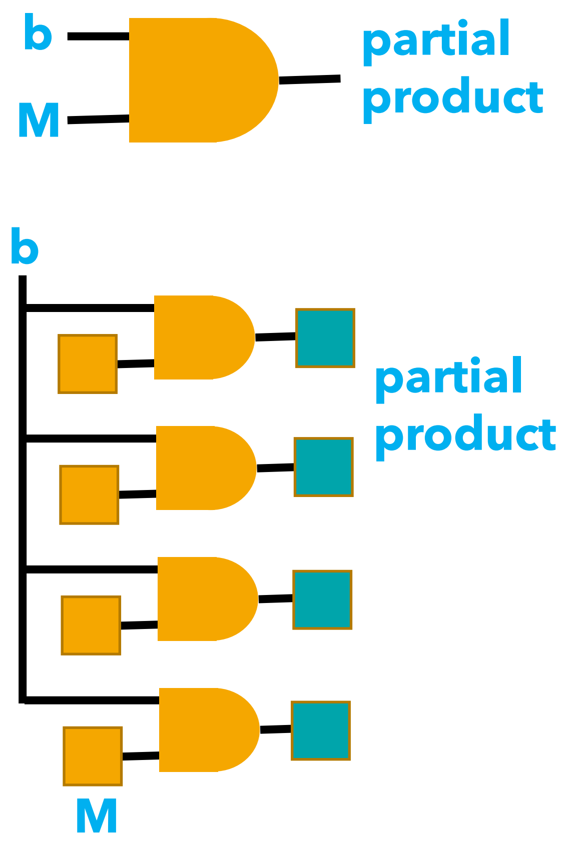
Multiplication Algorithm



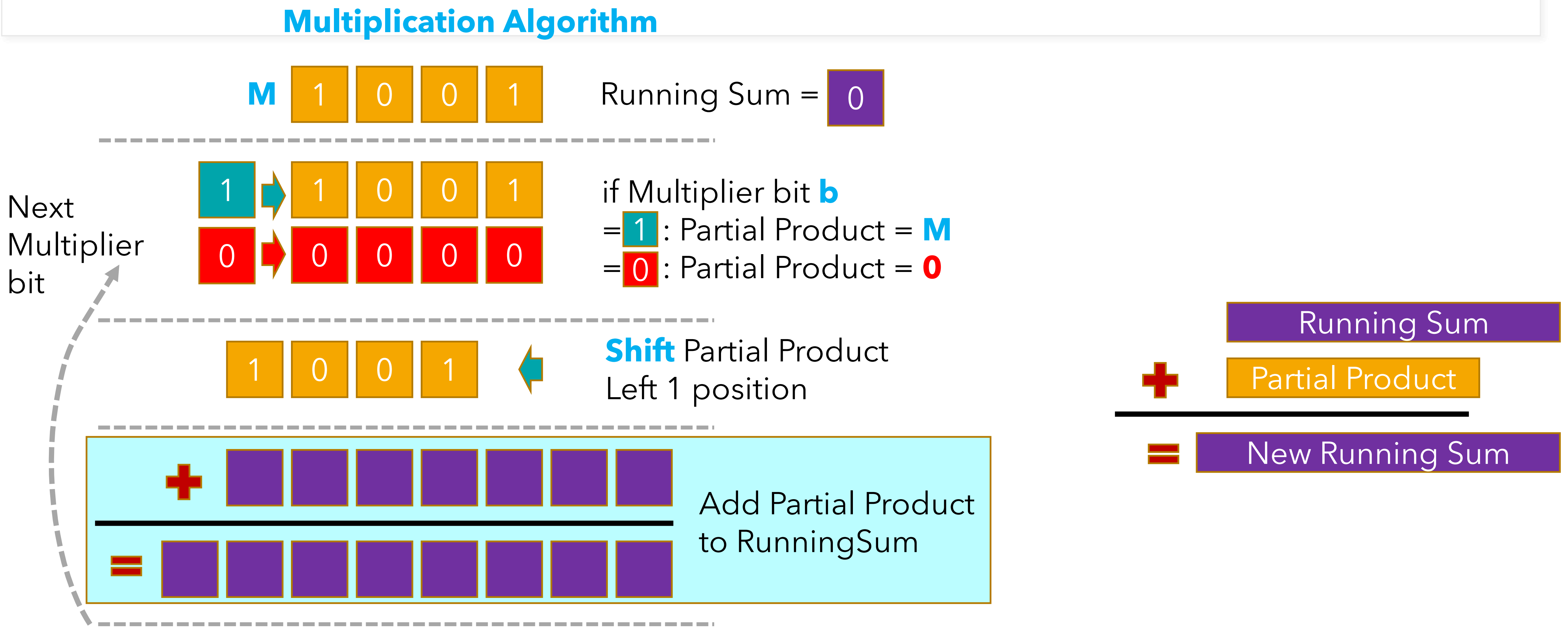

Multiplication Algorithm
M 1 0 0 1
Running Sum = 0
1 → 1 0 0 1
0 → 0 0 0 0
if Multiplier bit b
= 1 : Partial Product = M
= 0 : Partial Product = 0
Next Multiplier bit
1 0 0 1
Shift Partial Product Left 1 position
Add Partial Product to RunningSum

Multiplier Logic

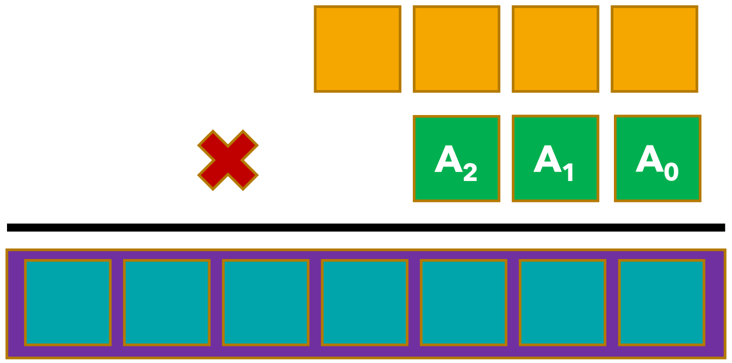

Multiplication Algorithm
M 1 0 0 1
Running Sum = 0
1 1 0 0 1
0 0 0 0 0
if Multiplier bit b
= 1 : Partial Product = M
= 0 : Partial Product = 0
Next Multiplier bit
1 0 0 1
Shift Partial Product Left 1 position
+
=
Add Partial Product to RunningSum

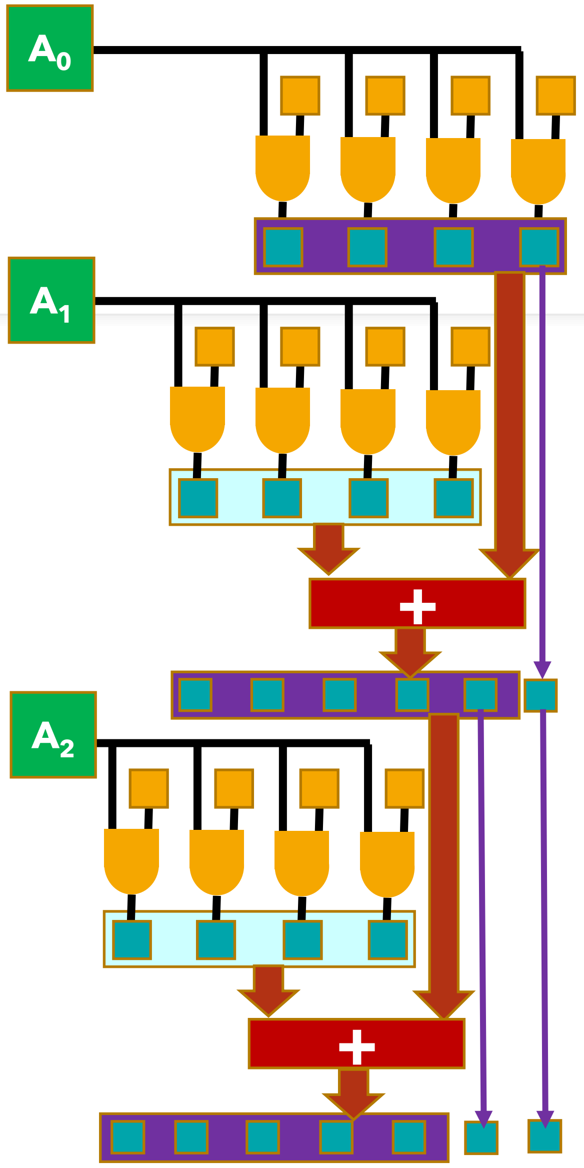


Multiplier Logic



4x3 Multiplier





Magnitude Comparator Logic

A = B

$x_3x_2x_1x_0$

$A = A_3A_2A_1A_0$

$B = B_3B_2B_1B_0$

$x_i = A_i'B_i' + A_iB_i$

A > B

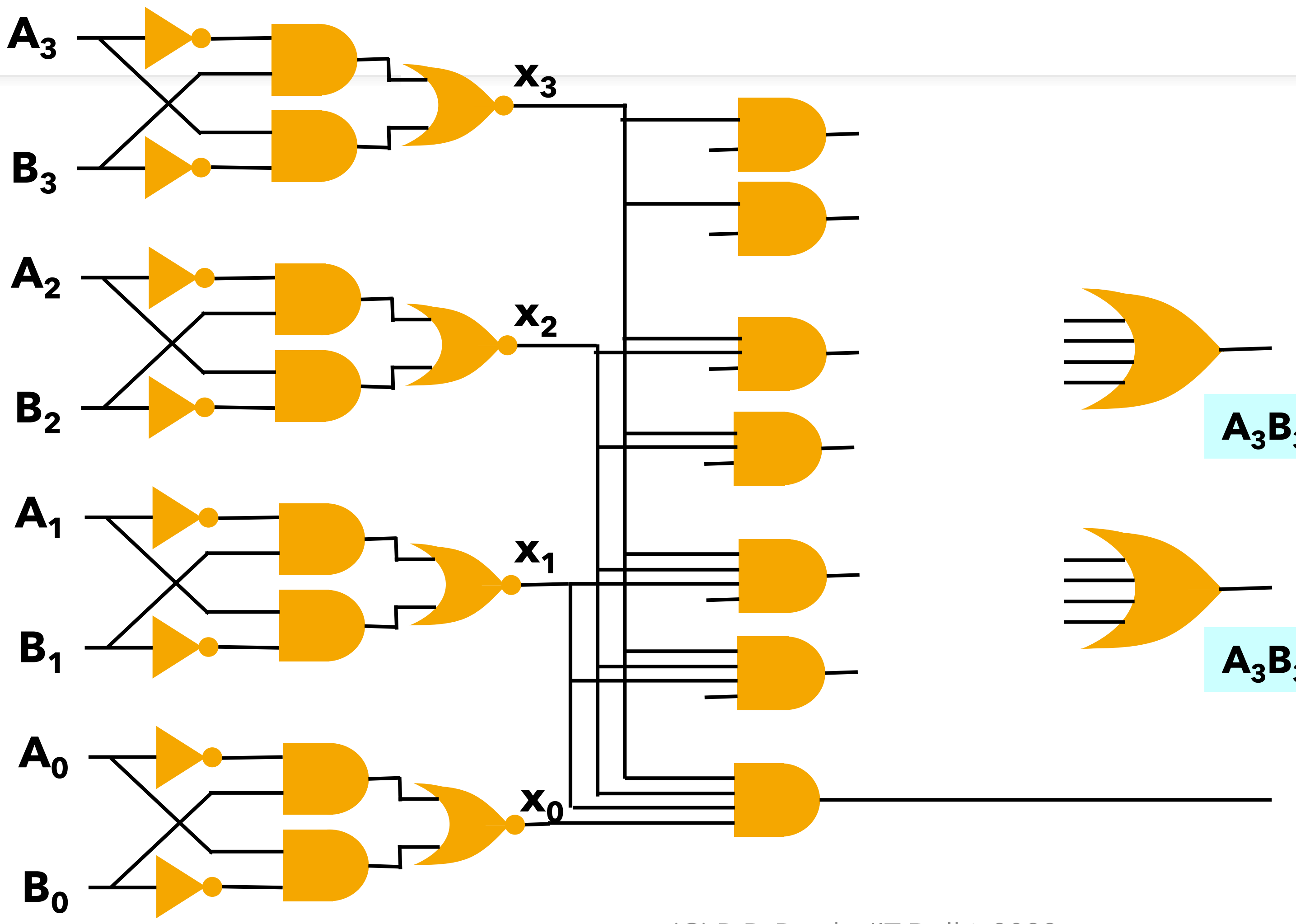
$A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

A < B

$A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

Similarity in expressions for the 3 comparisons

Magnitude Comparator Implementation



A > B

$A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

A < B

$A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

A = B

$x_3x_2x_1x_0$