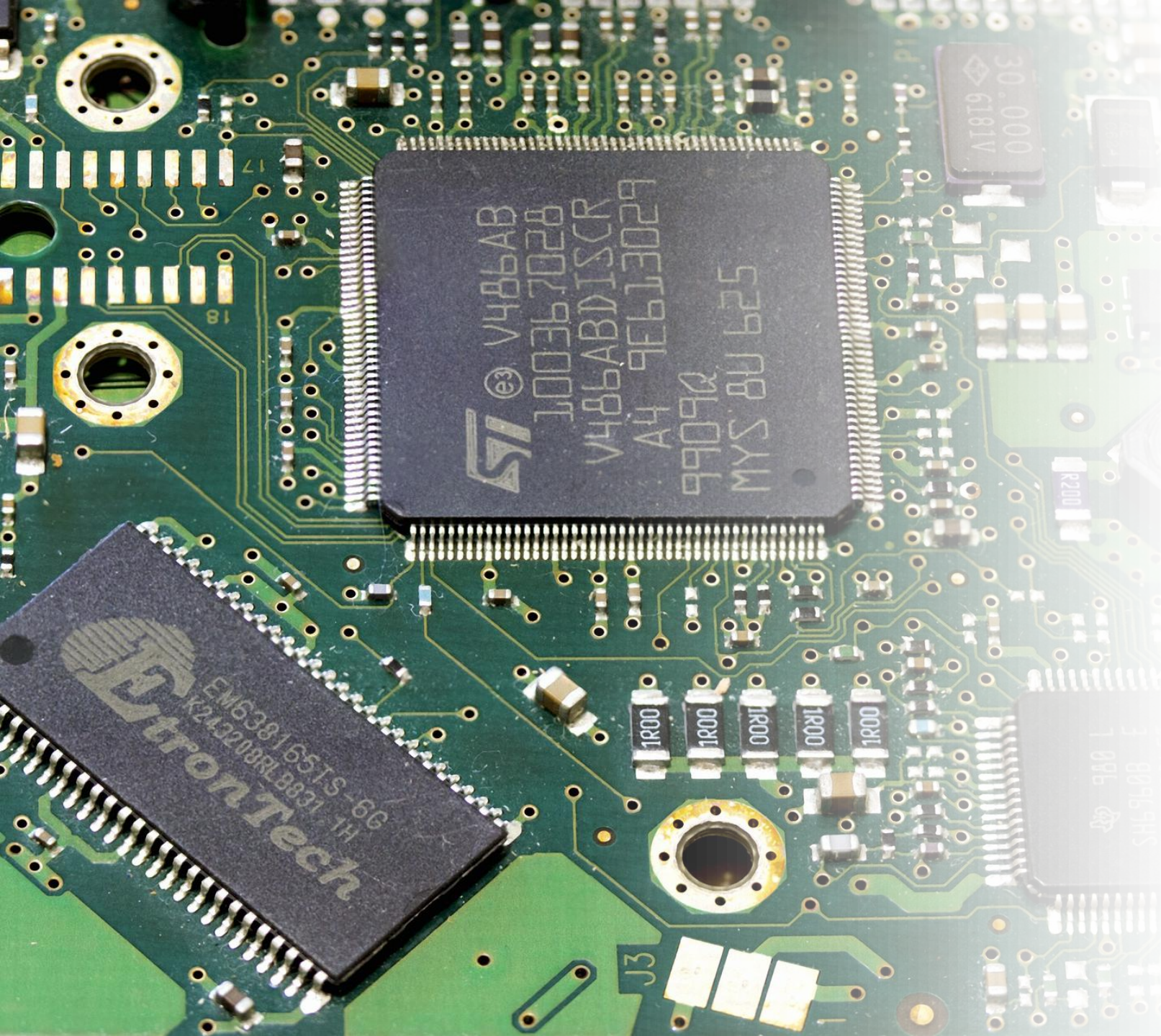




Digital Logic and System Design

5. Combinational Logic

COL215, I Semester 2024-2025

Venue: LHC 408

'E' Slot: Tue, Wed, Fri 10:00-11:00

Instructor: Preeti Ranjan Panda

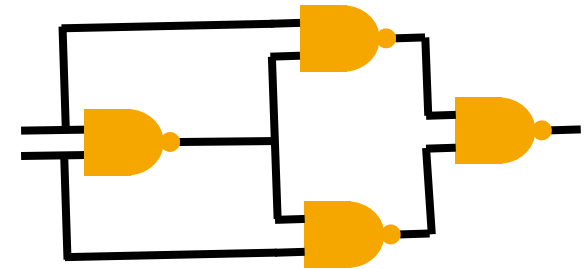
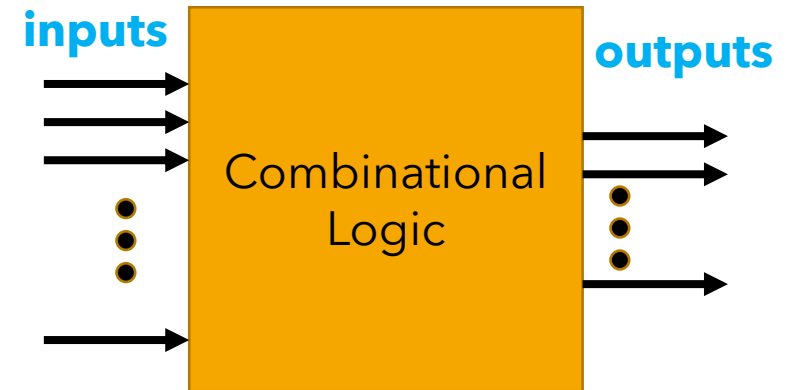
panda@cse.iitd.ac.in

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?

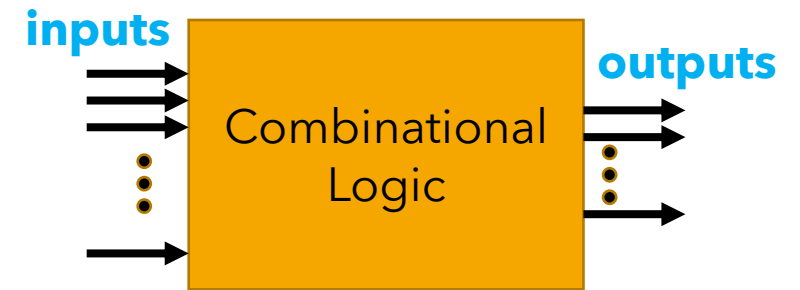


Example combinational circuit

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

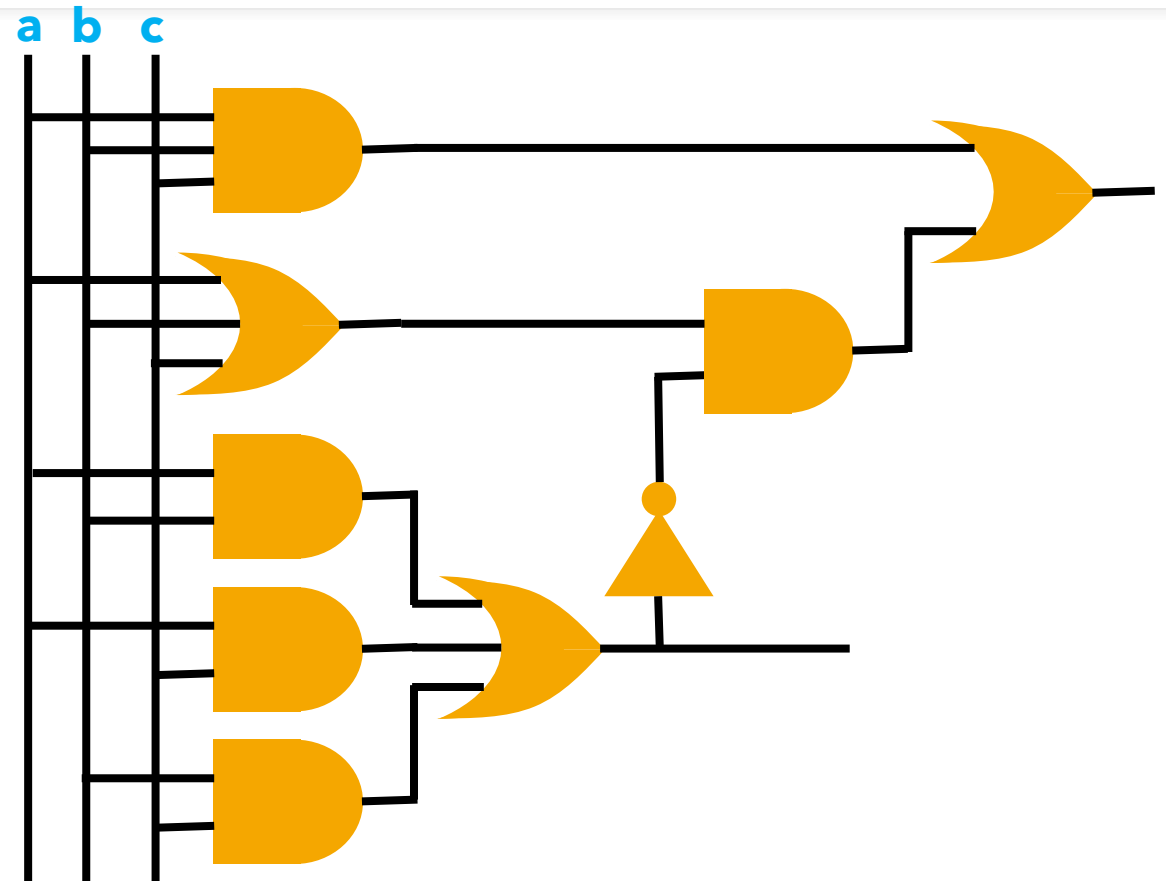


Tasks with Combinational Logic Circuits

- 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

a b c

000
001
010
011
100
101
110
111

Outputs

x y z

000
001
011
010
110
111
101
100

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

x (a, b, c)

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

y (a, b, c)

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

z (a, b, c)

Example: Boolean Simplification

Inputs

a b c

000

001

010

011

100

101

110

111

Outputs

x y z

000

001

011

010

110

111

101

100

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

$x = a$

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

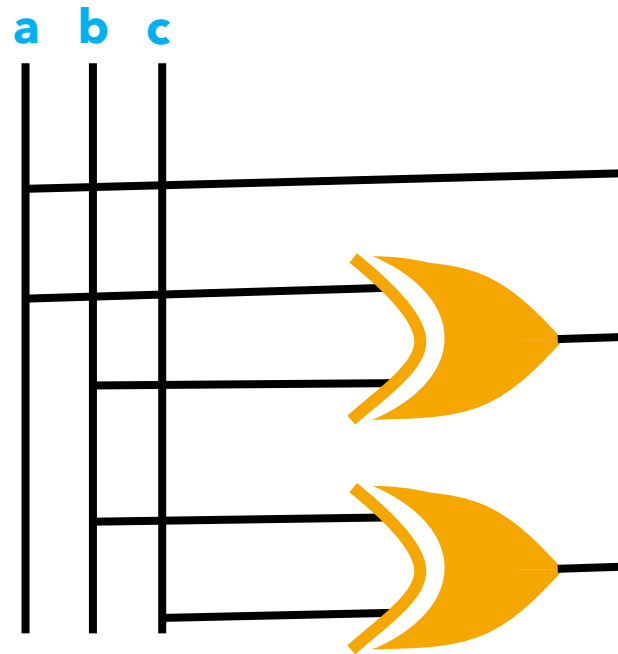
$y = a'b + ab'$

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

$z = b'c + bc'$

Gate Implementation

Binary Code (a, b, c)



Gray Code (x, y, z)

$$x = a$$

$$y = a'b + ab'$$

$$z = b'c + bc'$$

Designing a 1-bit Adder

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

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

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

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

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

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

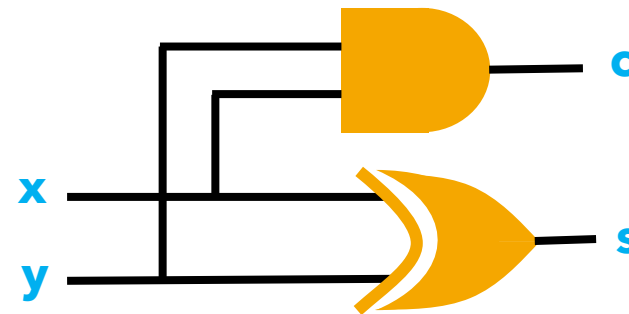
Adder: Simplification and Implementation

- 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 ($\mathbf{s}_{3-0}$), carry ($\mathbf{c}$)
- Truth Table?
- **Composing larger designs out of smaller ones**

$\mathbf{x}_{3-0}$		0	1	1	0	
$\mathbf{y}_{3-0}$	+	1	0	1	1	
	=	1	0	0	0	1
			└──────────┘			
		$\mathbf{c}$	$\mathbf{s}_{3-0}$			

Identify repeating pattern

Diagram illustrating a 4-bit addition ($x_{3-0} + y_{3-0}$) with a repeating pattern highlighted in a yellow box:

$$\begin{array}{r} x_{3-0} \quad 0 \ 1 \ 1 \ 0 \\ y_{3-0} \quad + \quad 1 \ 0 \ 1 \ 1 \\ \hline = \quad 1 \ 0 \ 0 \ 0 \ 1 \end{array}$$

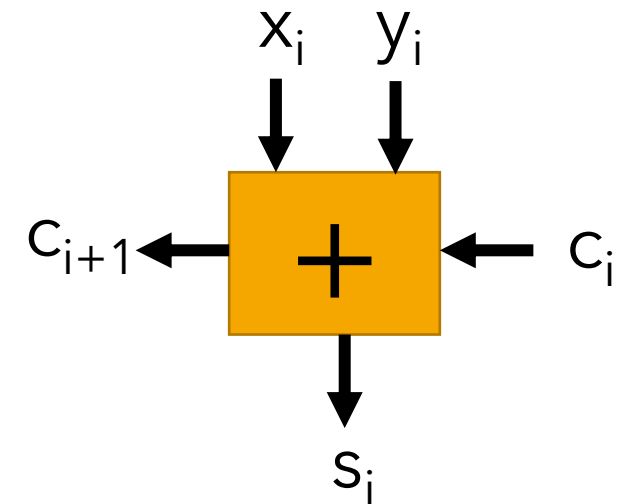
The result is shown as c (carry) and s_{3-0} (sum bits). The repeating pattern is indicated by a yellow box around the third bit position, with dashed red arrows showing the carry propagation from the third bit to the fourth bit.

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^o , s

a b c	c^o	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

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

Carry Out:

$$c^o = ab + bc + ca$$

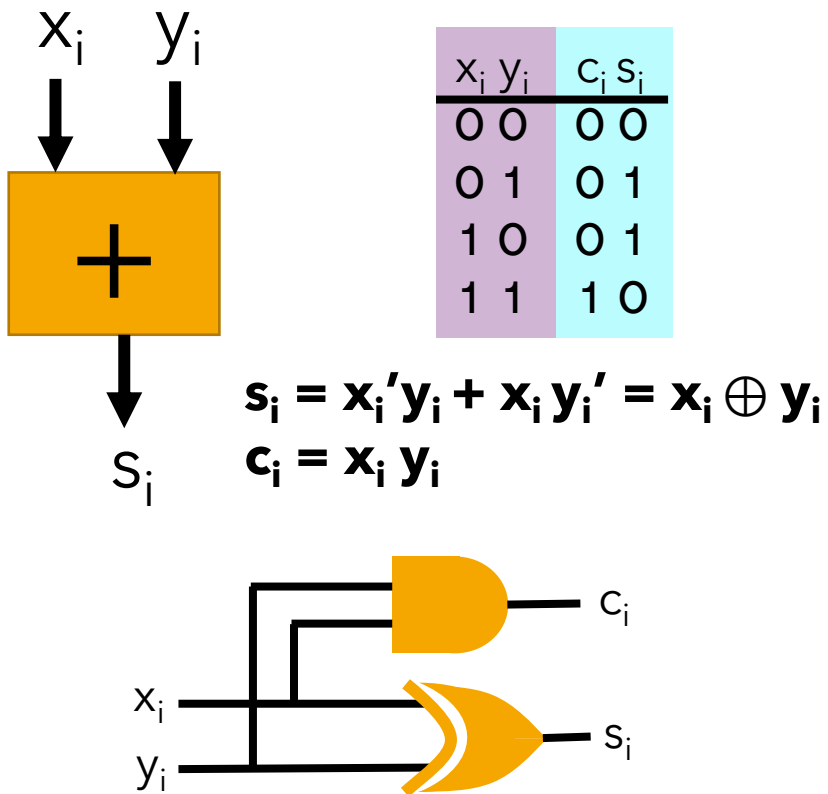
bc \ a	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Sum:

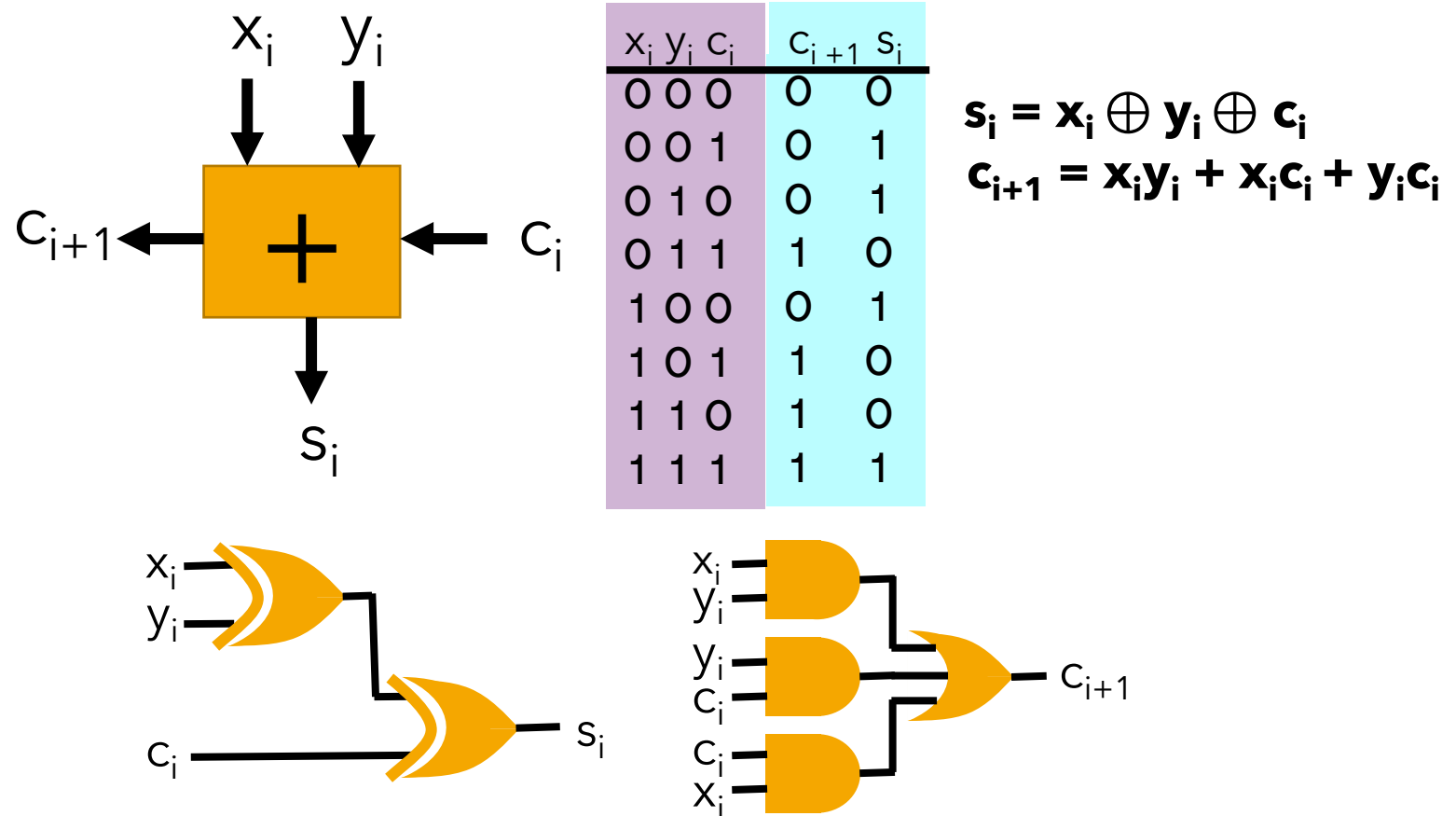
$$\begin{aligned} s &= ab'c' + a'b'c + a'bc' + abc \\ &= a(bc + b'c') + a'(b'c + bc') \\ &= a \oplus b \oplus c \end{aligned}$$

Half Adder vs. Full Adder

Half Adder



Full Adder



Ripple Carry Adder (RCA)

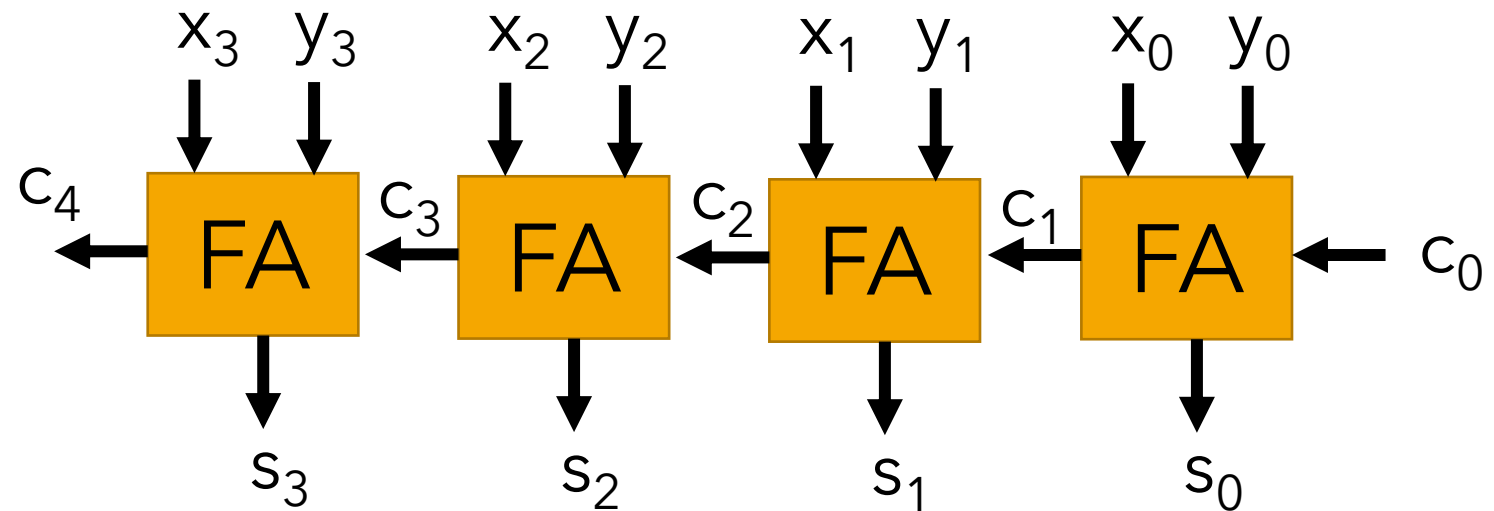
At each bit position i :

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?
 - $\text{carry_out} = 1$ irrespective of carry_in values
- **Carry Propagation**: When do we **propagate** an input carry to the output irrespective of input values?
 - $\text{carry} = \text{carry_in}$ 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**

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

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$G_i = x_i y_i$$

$$P_i = x_i \oplus y_i$$



$$s_i = P_i \oplus c_i$$

$$c_{i+1} = G_i + P_i c_i \quad (\text{verify})$$

Carry Lookahead Logic

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

$$c_1 = G_0 + P_0 c_0$$

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

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

$$\begin{aligned} c_4 &= G_3 + P_3 c_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0) \\ &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0 \end{aligned}$$

- **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)

$$\begin{aligned} G_i &= x_i y_i & s_i &= P_i \oplus c_i \\ P_i &= x_i \oplus y_i & c_{i+1} &= G_i + P_i c_i \end{aligned}$$

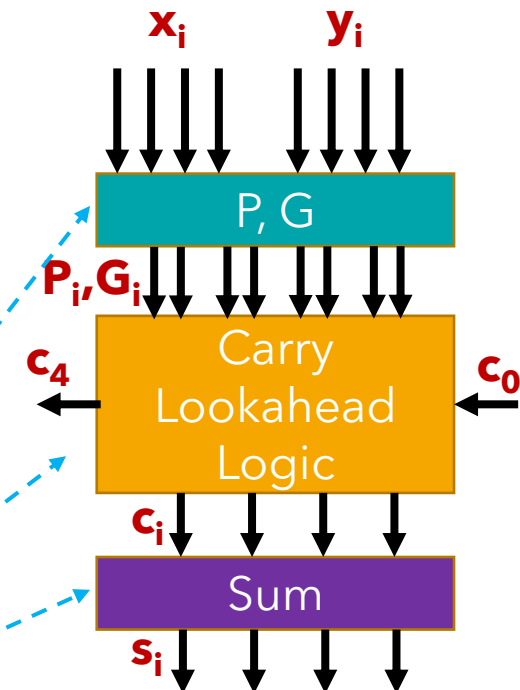
$$c_1 = G_0 + P_0 c_0$$

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

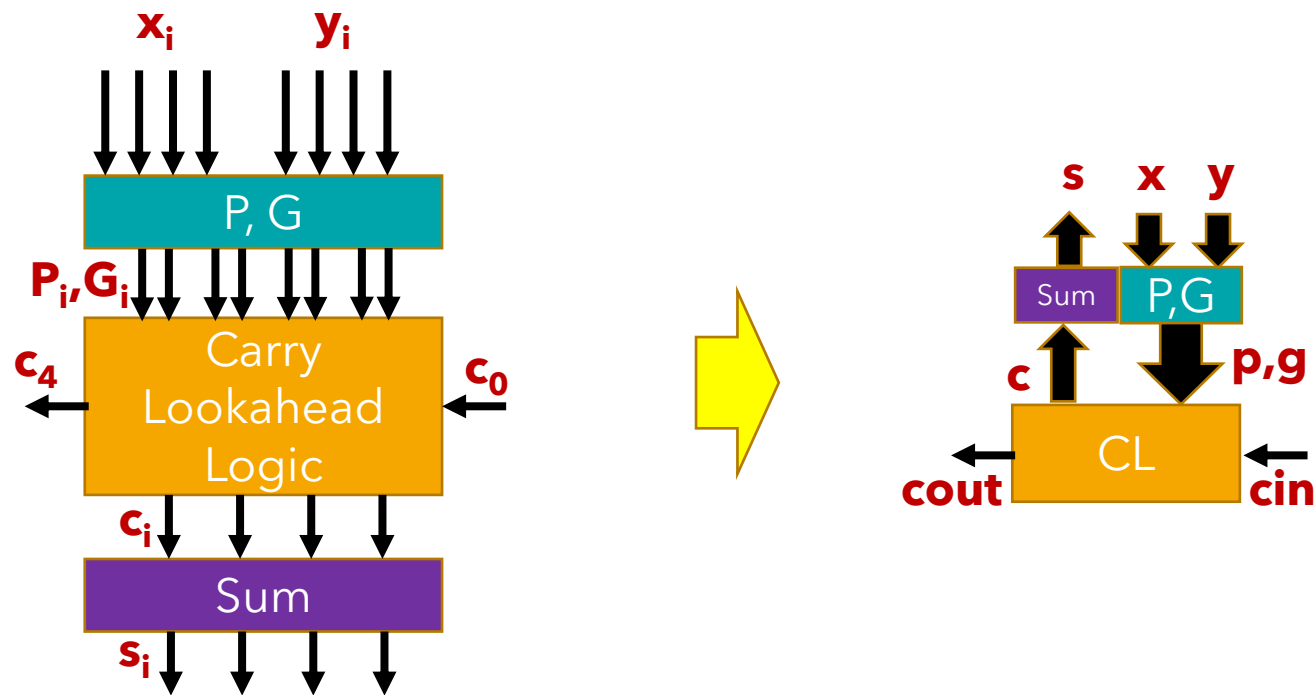
$$c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

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

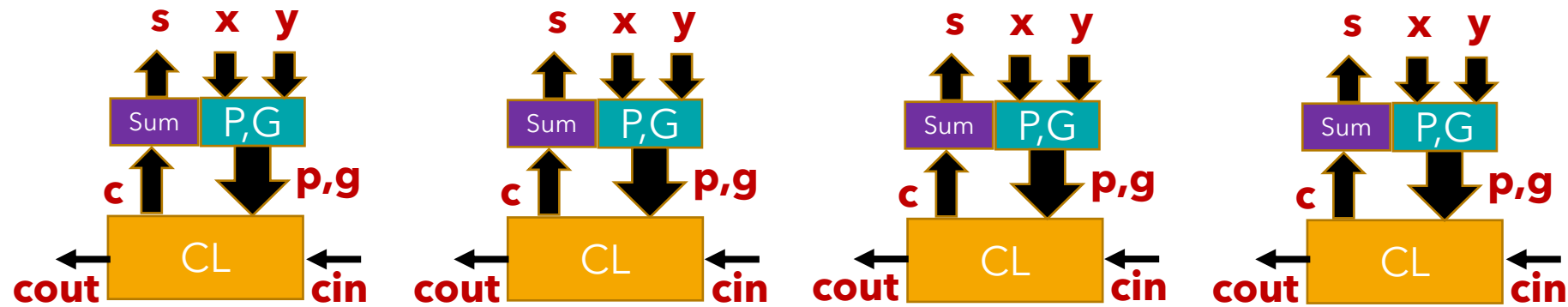
- **1 logic level to generate all P_i and G_i**
- **2 logic levels to generate c_4 from c_0**
 - Approx: 5 i/p gate has same delay as 2 i/p gate
- **1 logic level to generate all sums s_i**
- 4-bit Adder delay: **1+2+1 = 4 levels**



4-bit CLA: Simplified Diagram



16-bit Adder from 4-bit CLA

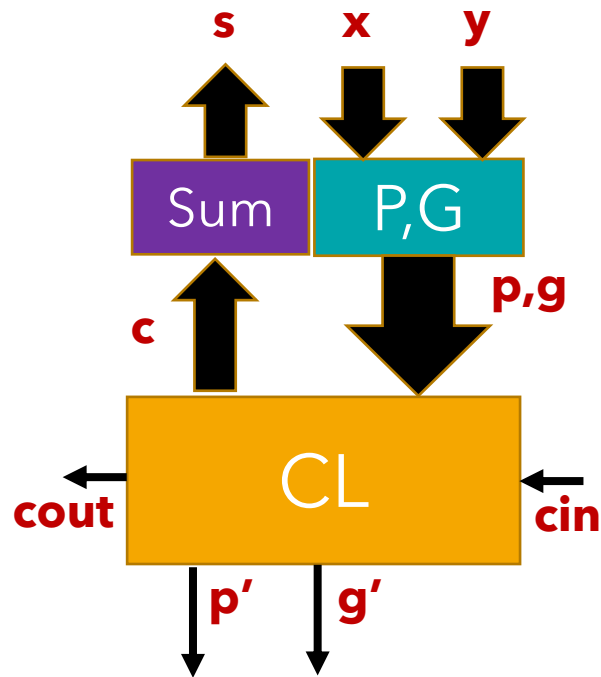


$$g_i = x_i y_i$$
$$p_i = x_i \oplus y_i$$

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

How do we extend the structure?

CL block-level carry propagate/generate

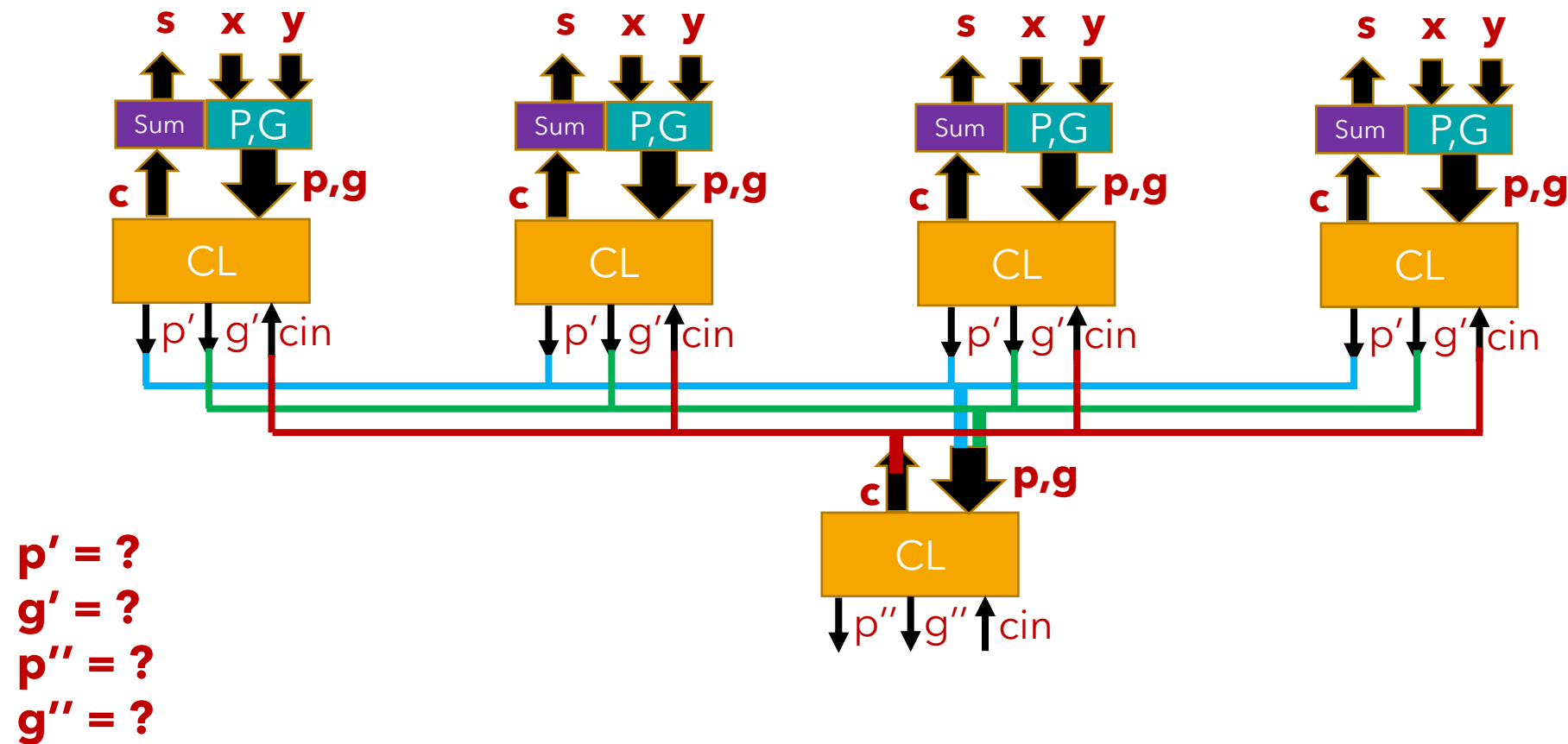


$$g_i = x_i y_i$$
$$p_i = x_i \oplus y_i$$

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

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

16-bit Adder from 4-bit CLA



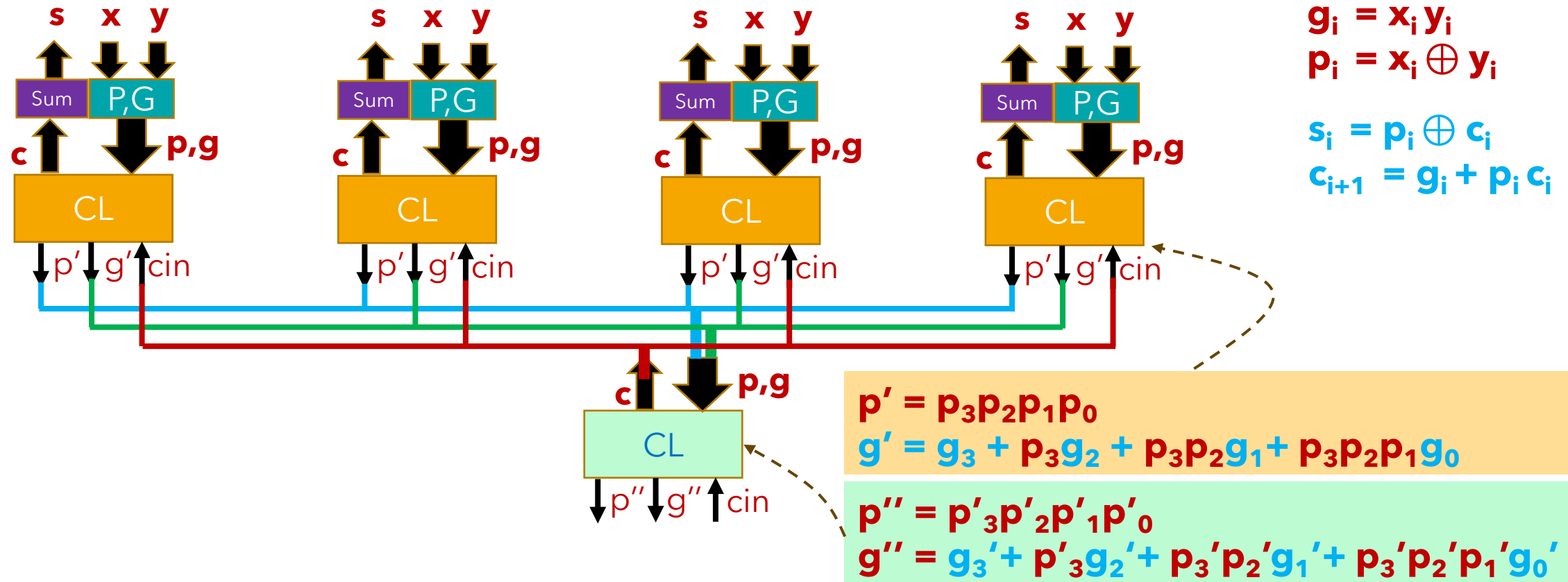
$$g_i = x_i y_i$$

$$p_i = x_i \oplus y_i$$

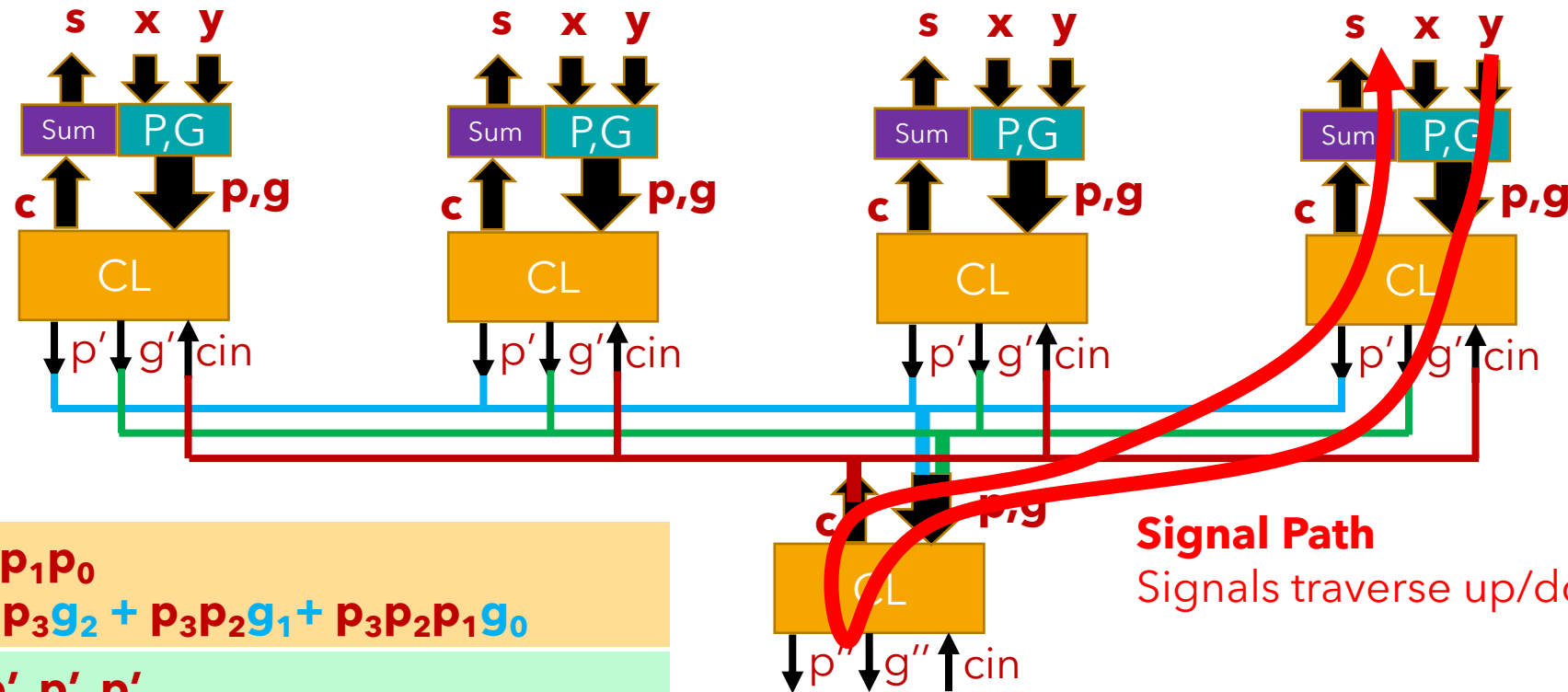
$$s_i = p_i \oplus c_i$$

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

16-bit Adder from 4-bit CLA



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



$$g_i = x_i y_i$$

$$p_i = x_i \oplus y_i$$

$$s_i = p_i \oplus c_i$$

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

$$p' = p_3 p_2 p_1 p_0$$

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

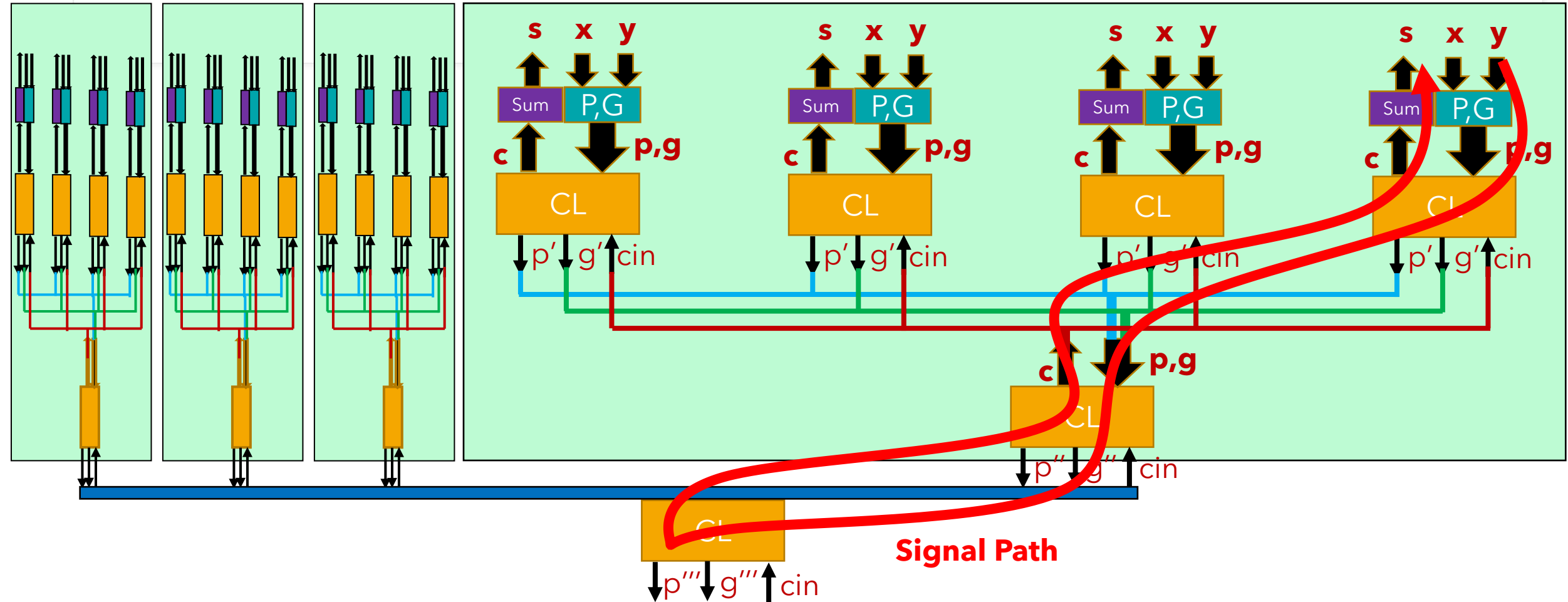
$$p'' = p'_3 p'_2 p'_1 p'_0$$

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

Signal Path

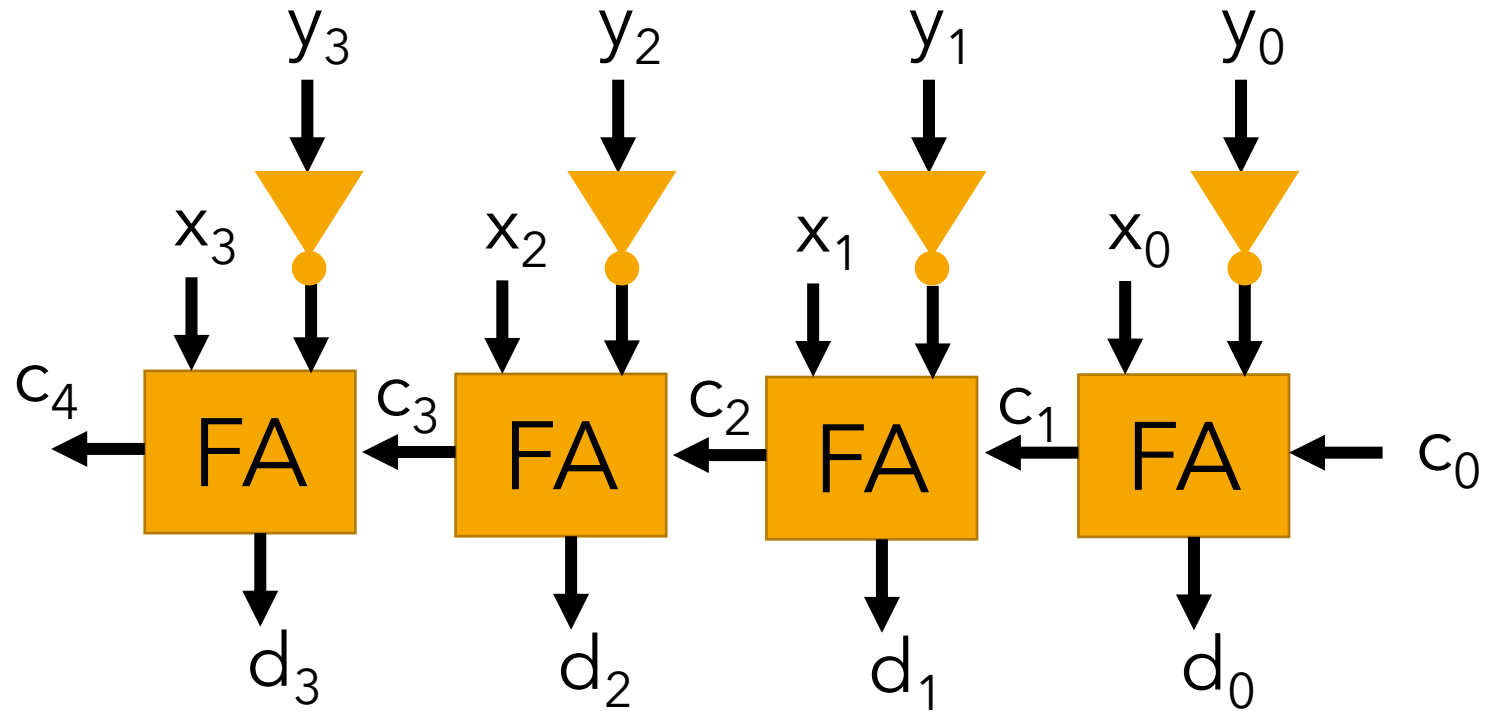
Signals traverse up/down, not left/right

64-bit Adder from 16-bit CLAs



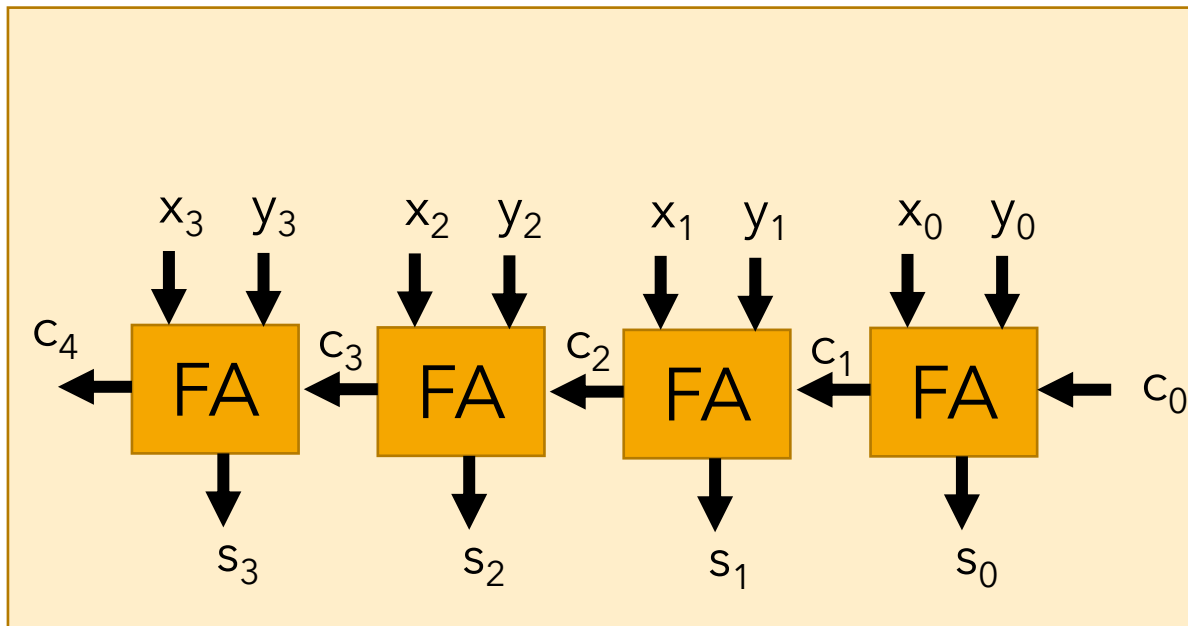
n-bit Subtraction

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

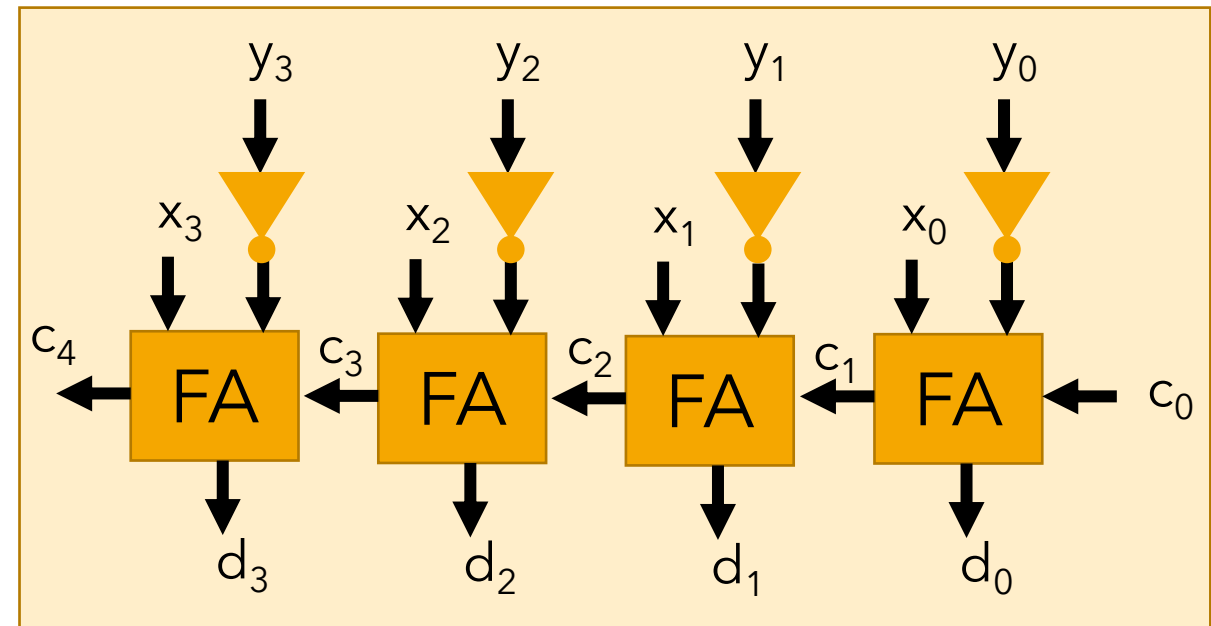


Programmable Adder/Subtractor

Adder

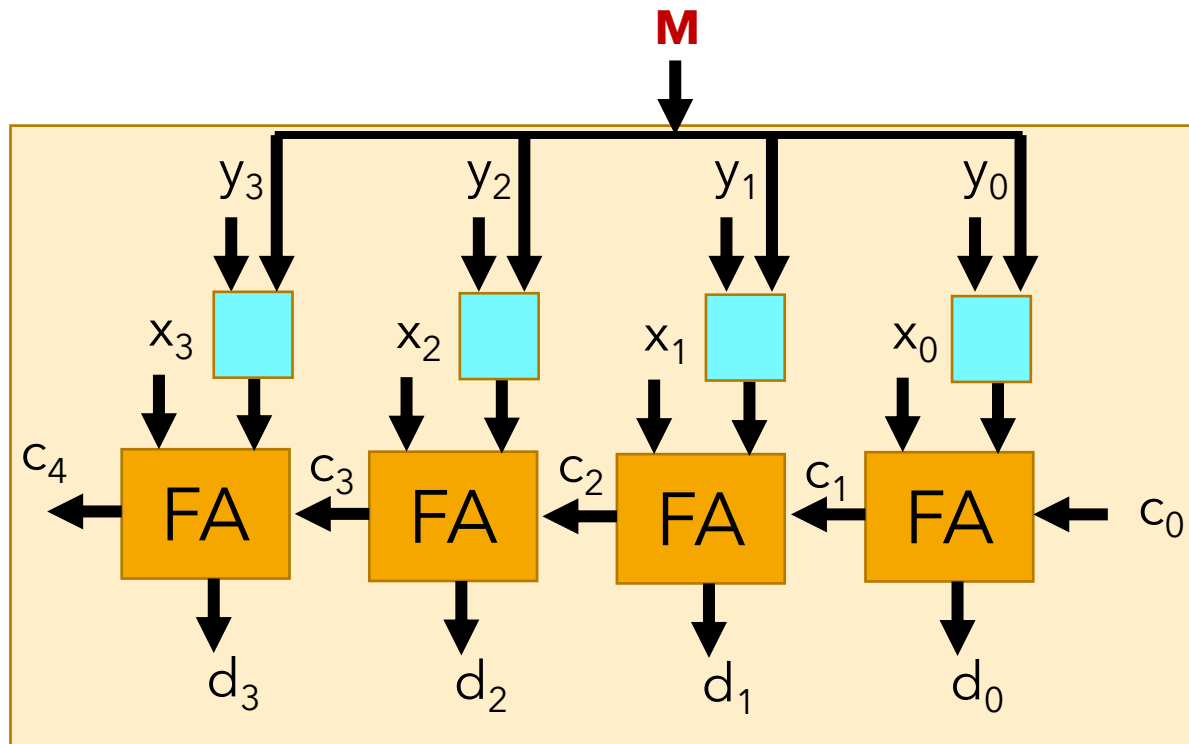


Subtractor

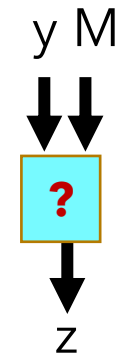


Very similar!
Can we combine into one structure?

Programmable Adder/Subtractor

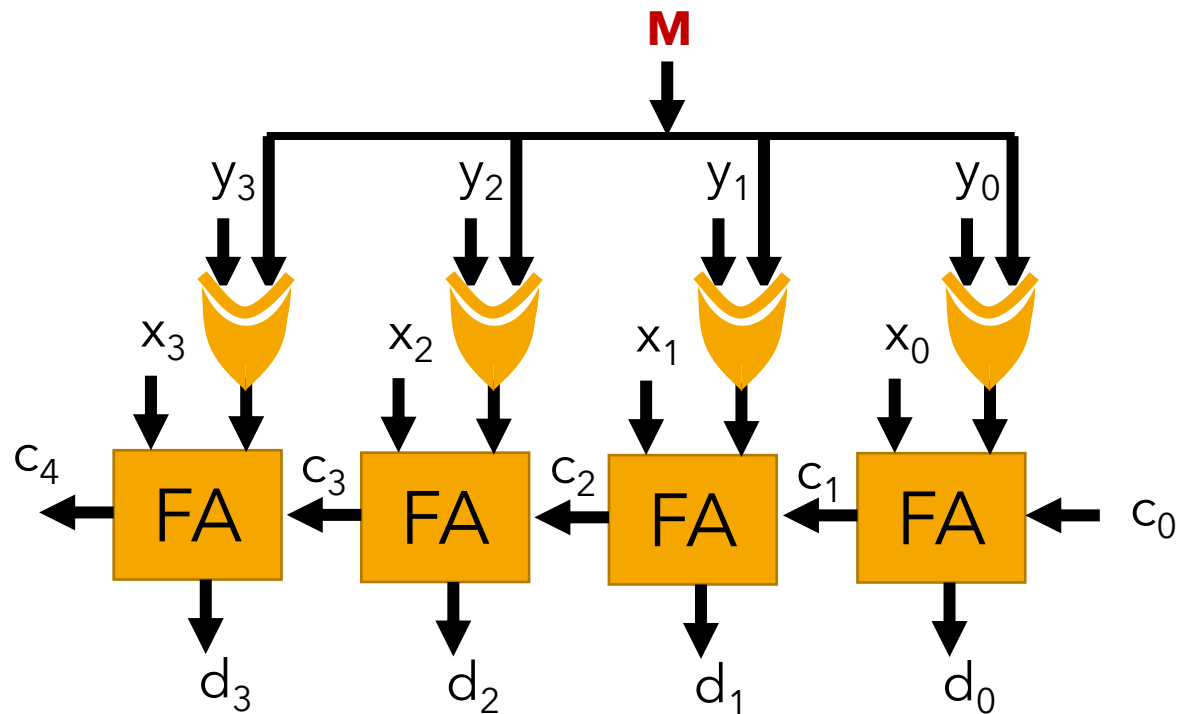


$M = 0$: Add
 $M = 1$: Subtract

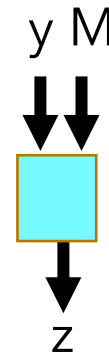


$M = 0: z = y$
 $M = 1: z = y'$
What function is $z(y, M)$?

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

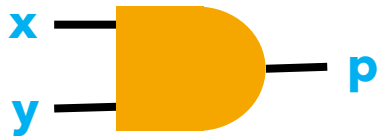
$$z = M \oplus y$$

Binary Multiplier

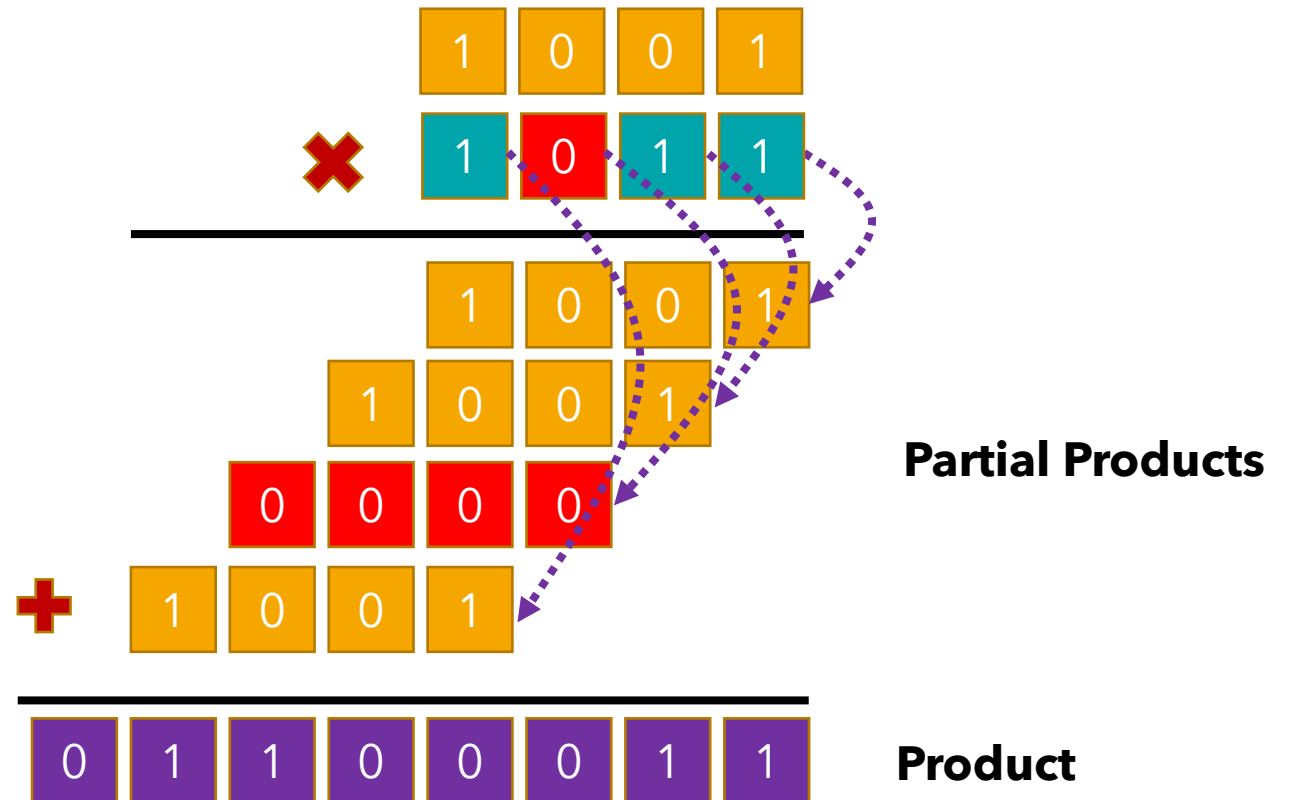
1x1 Multiplier

0	0	1	1
x 0	x 1	x 0	x 1
= 0	= 0	= 0	= 1

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

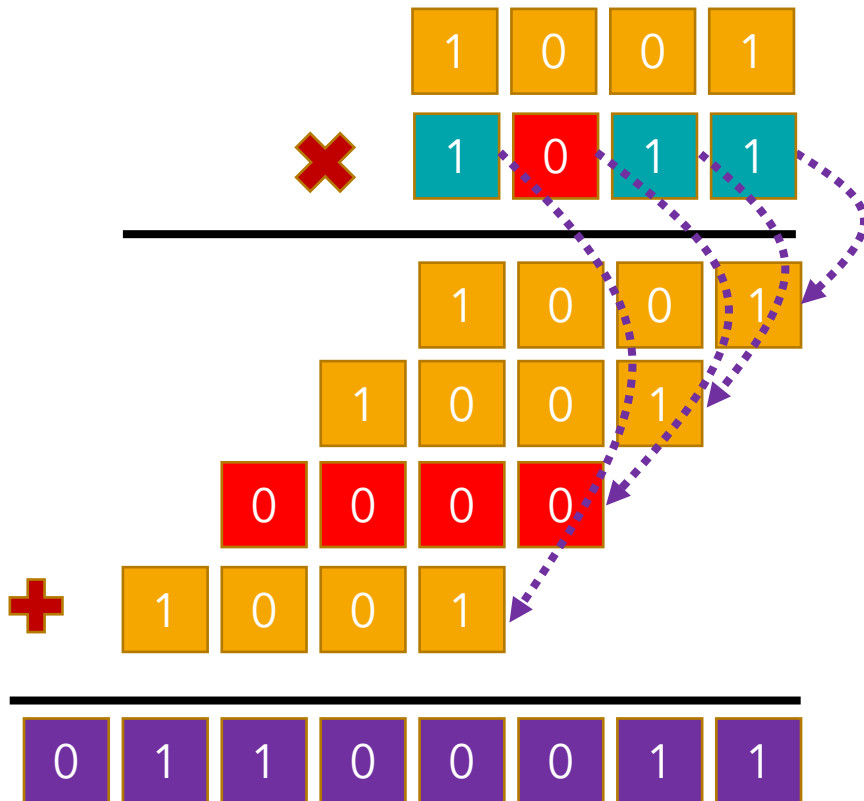


4x4 Multiplier

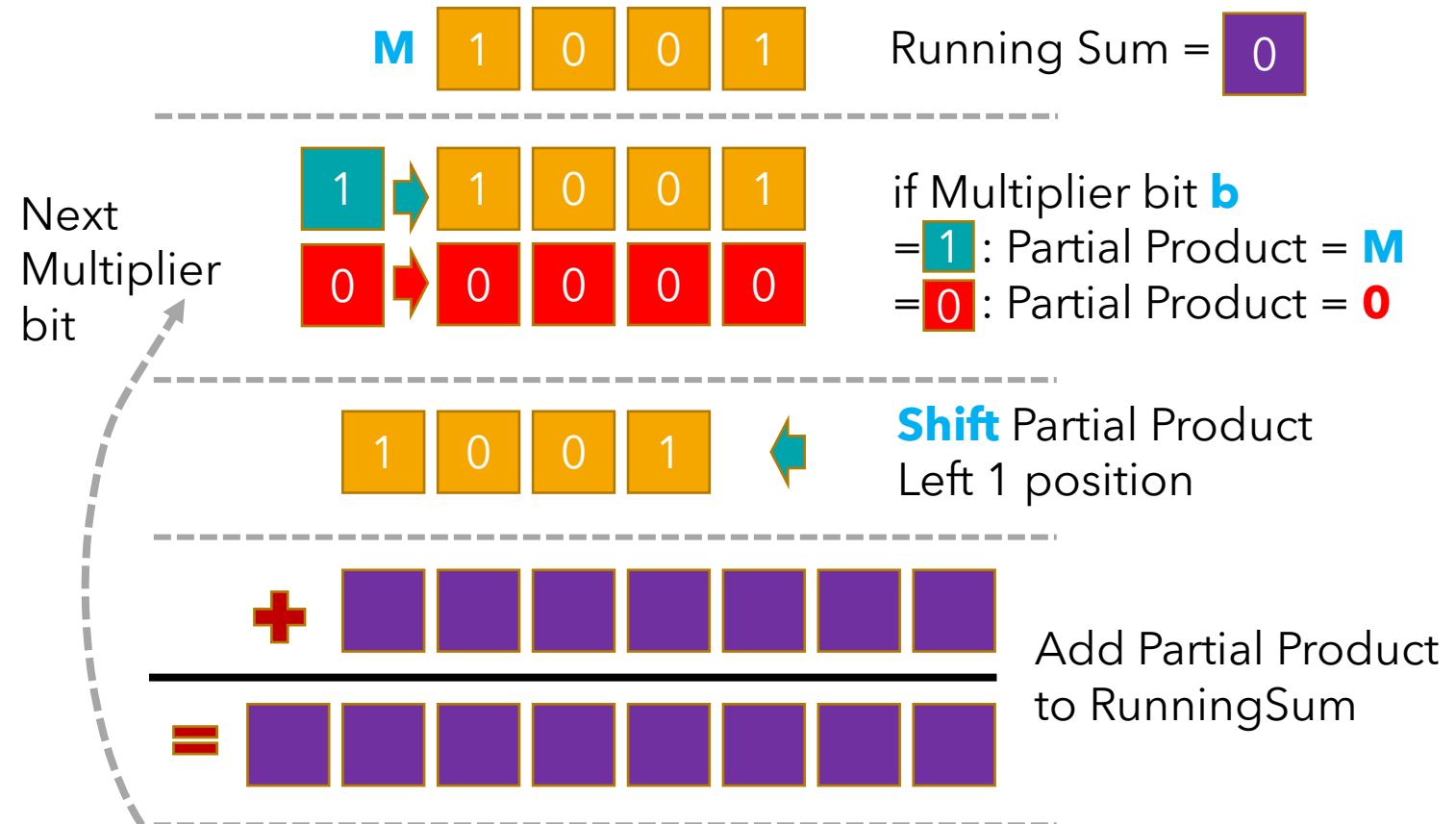


Multiplication Algorithm

4x4 Multiplier

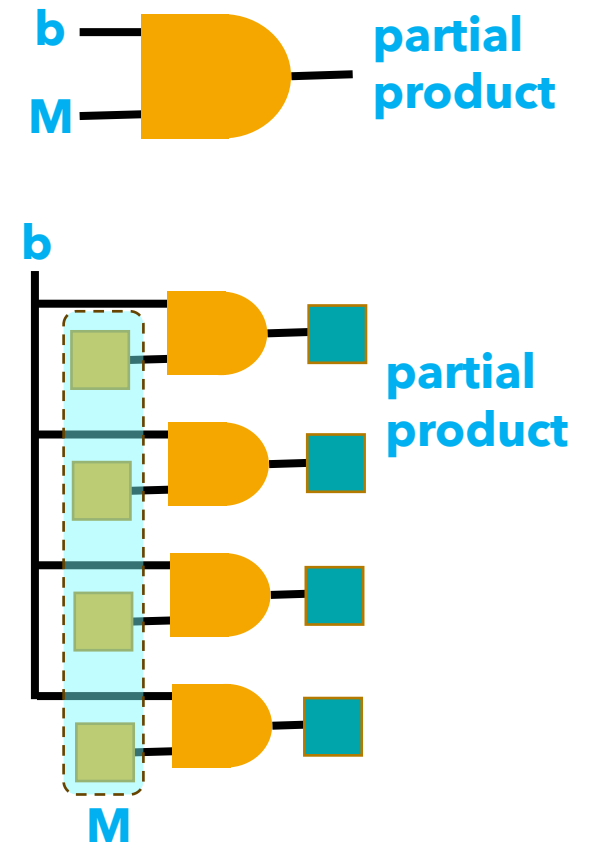
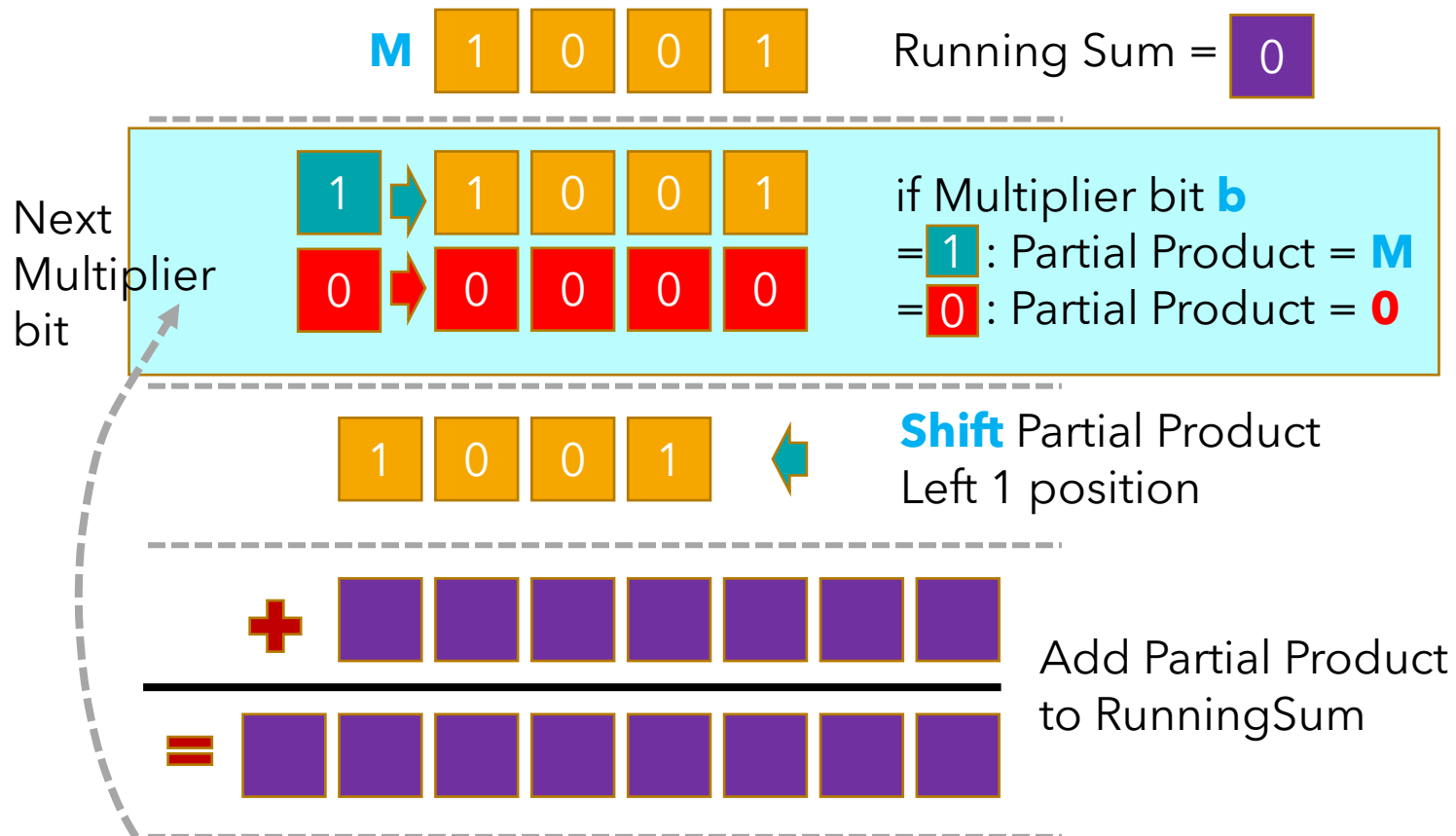


Multiplication Algorithm



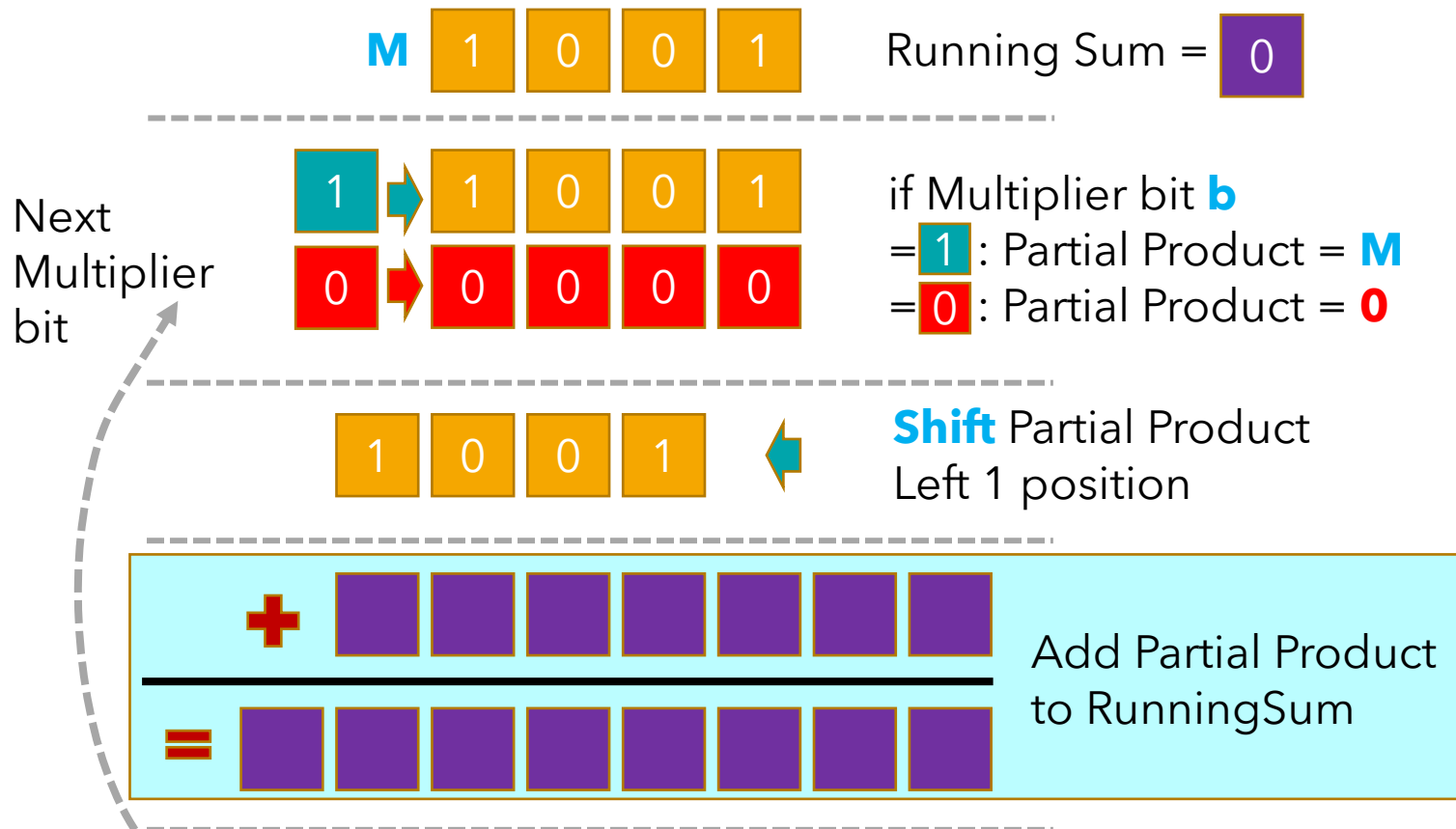
Multiplier Logic

Multiplication Algorithm

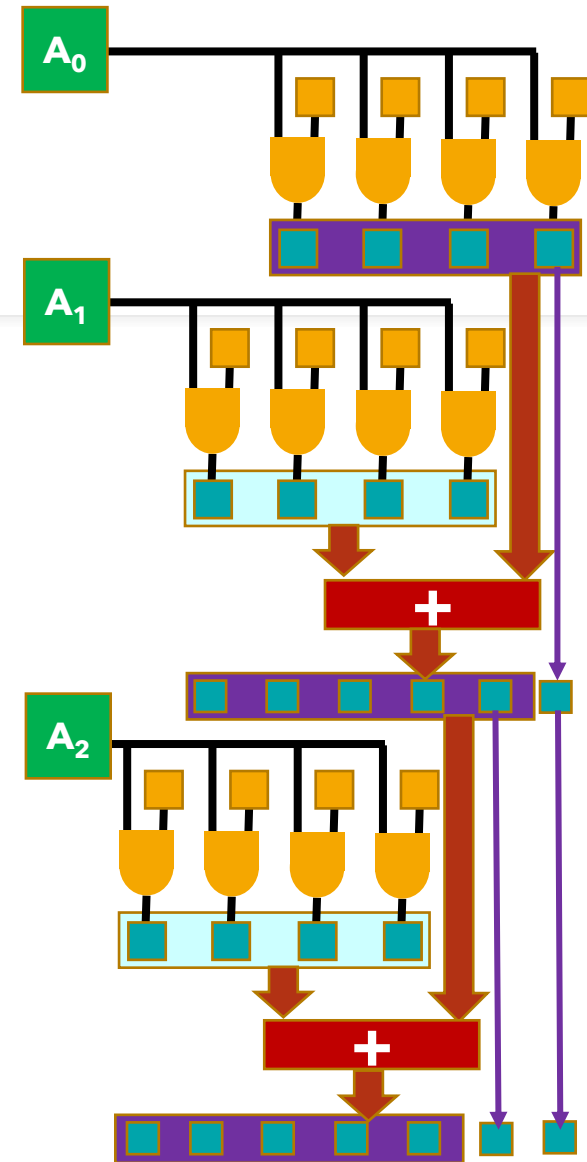
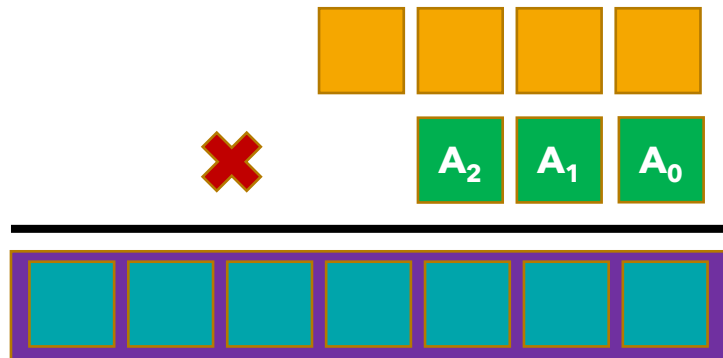


Multiplier Logic

Multiplication Algorithm



4x3 Multiplier



Magnitude Comparator Logic

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

$$x_i = A_i'B_i' + A_iB_i$$

$$A = B$$

$$x_3x_2x_1x_0$$

$$A > B$$

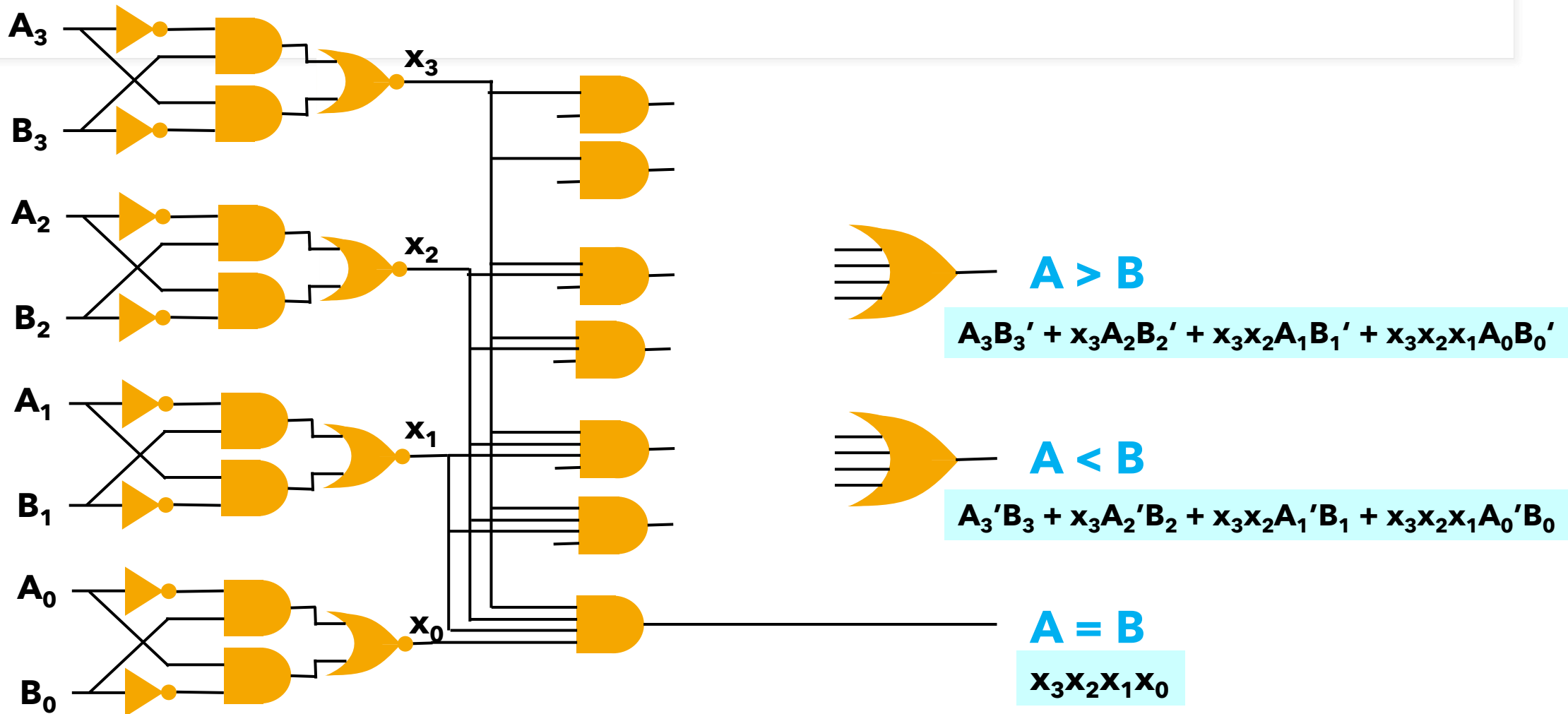
$$A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$A < B$$

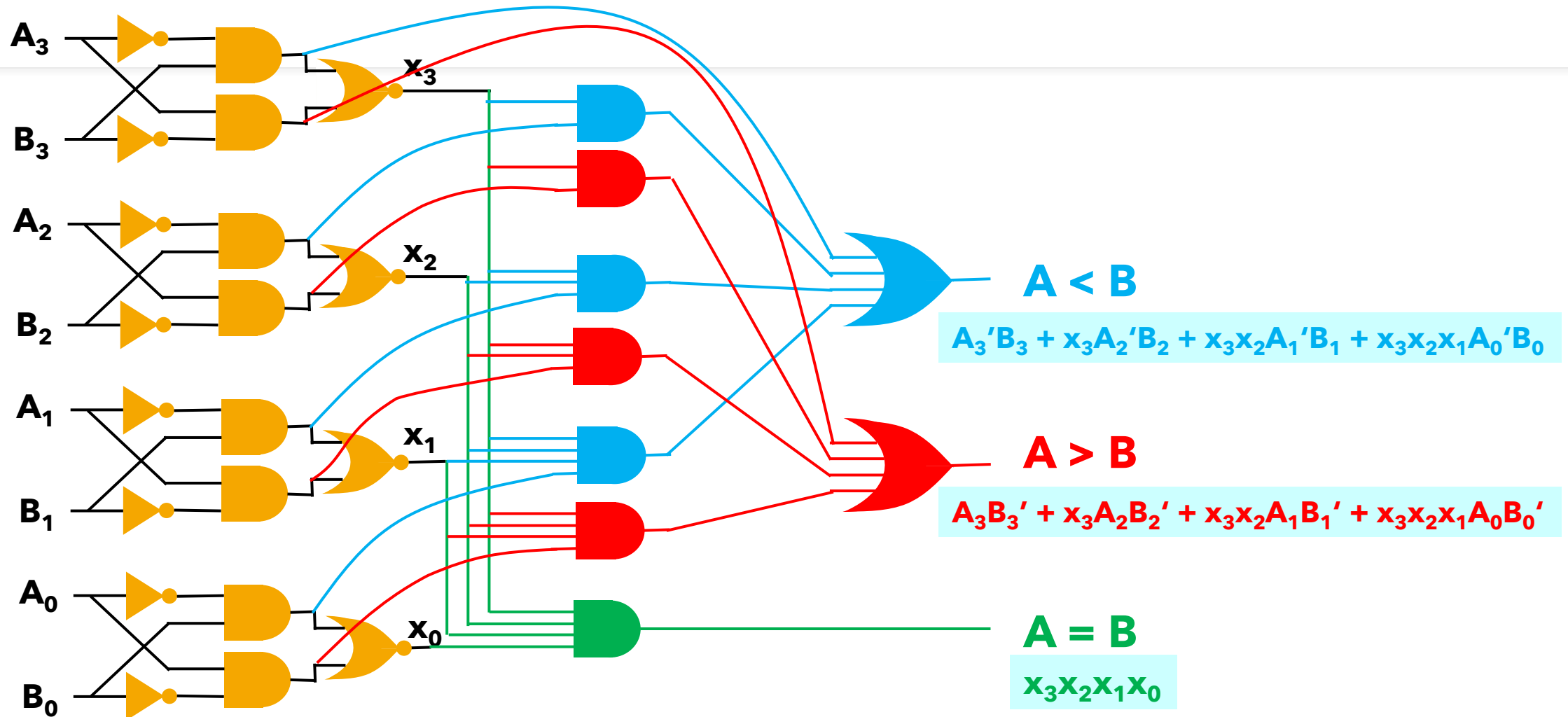
$$A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Similarity in expressions for the 3 comparisons

Magnitude Comparator Implementation



Magnitude Comparator Implementation

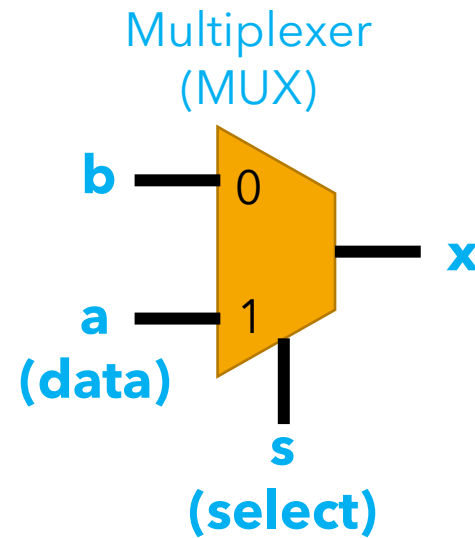


Multiplexer: Implementing Conditionals

- Selection Logic

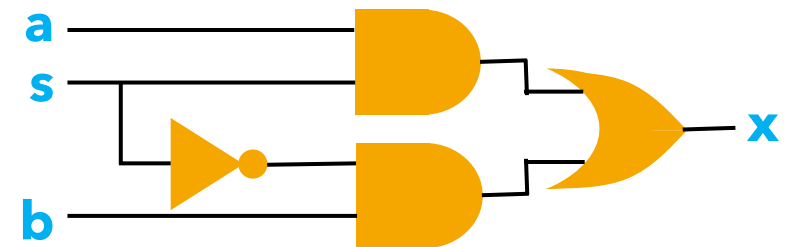
Function

```
if (s)
  x = a
else
  x = b
```



How do we
implement a MUX?

$$x = sa + s'b$$

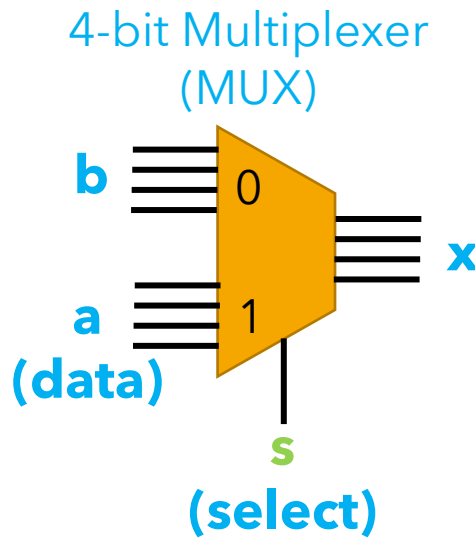


MUX with wider data

- Selection Logic

Function

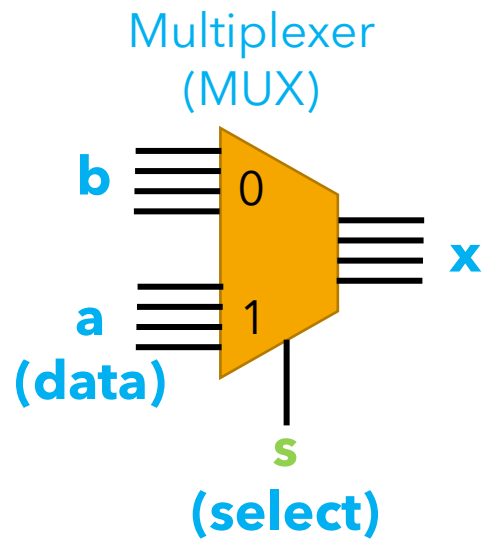
```
if (s)
  x [3:0] = a [3:0]
else
  x [3:0] = b [3:0]
```



How do we implement
a 4-bit MUX?

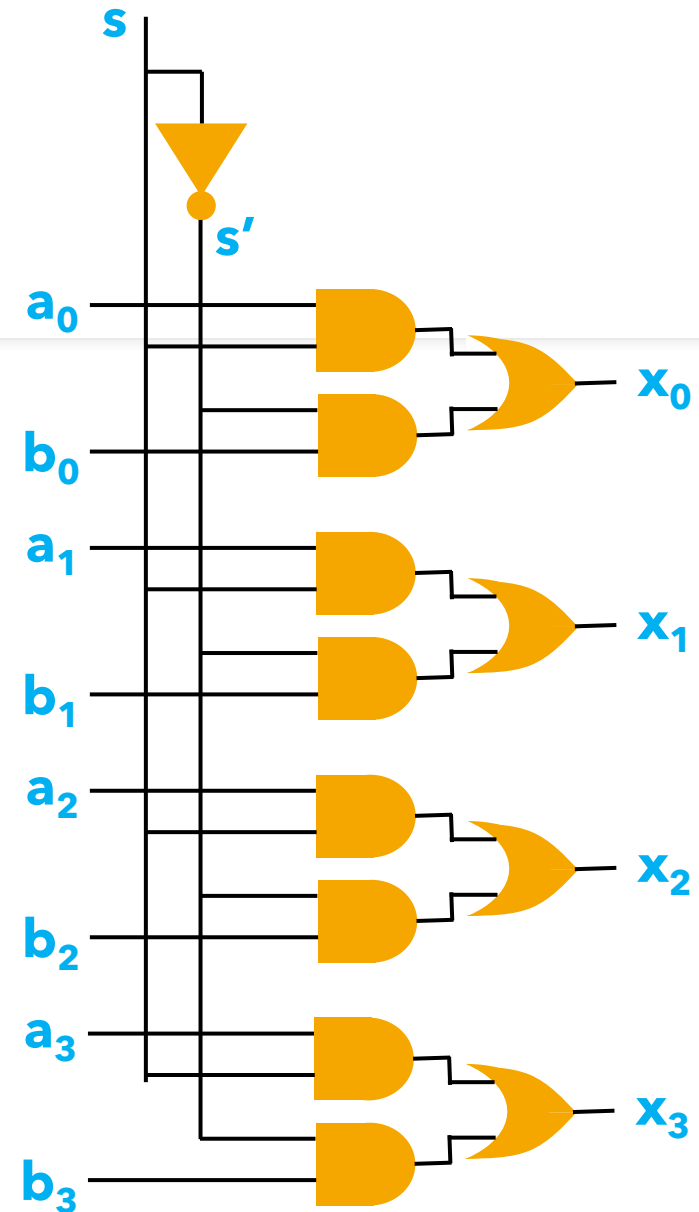
$$\begin{aligned}x_0 &= sa_0 + s'b_0 \\x_1 &= sa_1 + s'b_1 \\x_2 &= sa_2 + s'b_2 \\x_3 &= sa_3 + s'b_3\end{aligned}$$

MUX with wider data



How do we implement
a 4-bit MUX?

$$\begin{aligned}x_0 &= sa_0 + s'b_0 \\x_1 &= sa_1 + s'b_1 \\x_2 &= sa_2 + s'b_2 \\x_3 &= sa_3 + s'b_3\end{aligned}$$



MUX with multiple data (wider **select**)

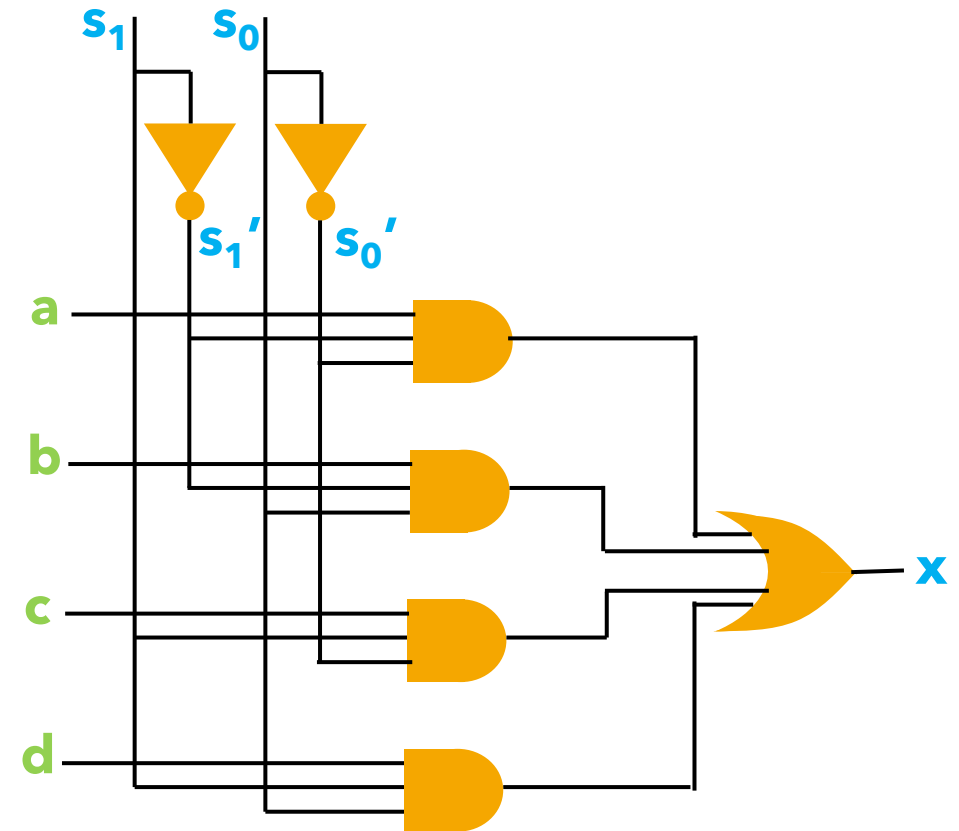
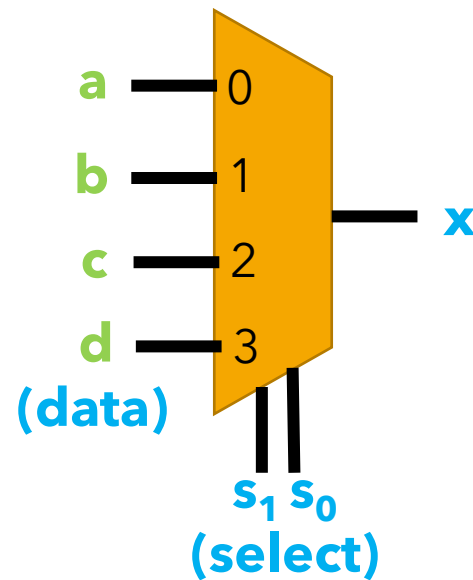
$$x = s_1's_0'a + s_1s_0'b + s_1's_0c + s_1s_0d$$

Function (C++)

```
switch (s) {  
  case 0: x = a; break;  
  case 1: x = b; break;  
  case 2: x = c; break;  
  default: x = d; break;  
}
```

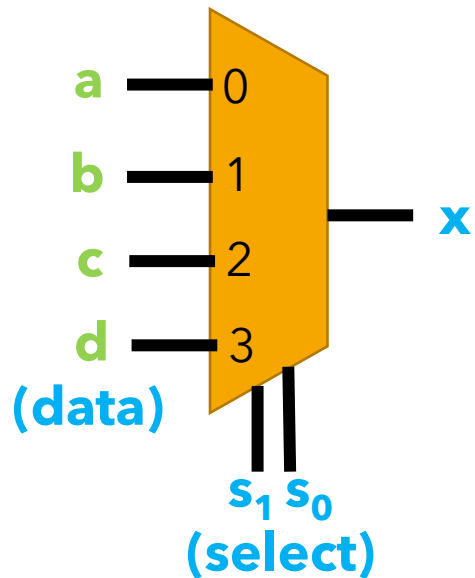
Select a, b, c, or d
depending on value of s

How do we implement
a **4-to-1 MUX**?



Implement ANY function with MUX

$$x = s_1's_0'a + s_1s_0'b + s_1's_0c + s_1s_0d$$



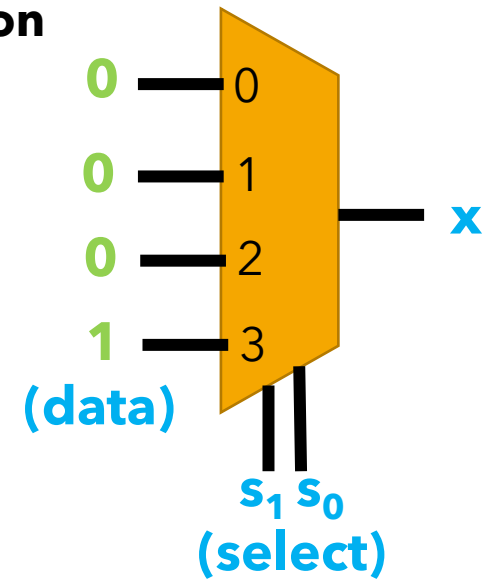
Can we implement
ANY function of
2 variables
with this structure?

$$x = s_1's_0'0 + s_1s_0'0 + s_1's_00 + s_1s_01$$

Example Function

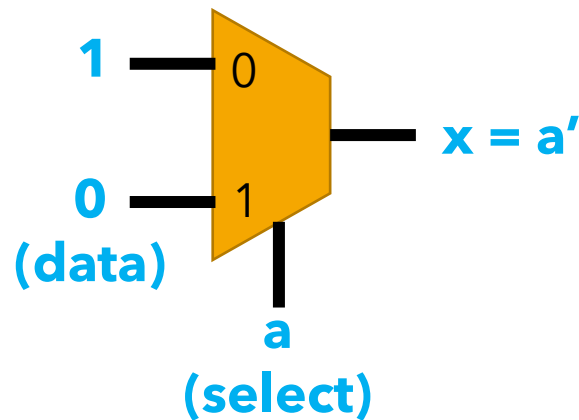
s_1s_0	x
0 0	0
0 1	0
1 0	0
1 1	1

Implement
with MUX



Implement ANY function with MUX

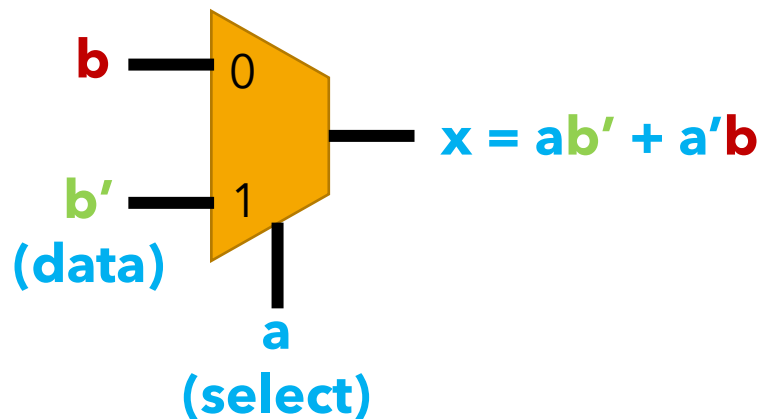
Can we implement
 $x = a'$
using MUX?



Implement with MUX:

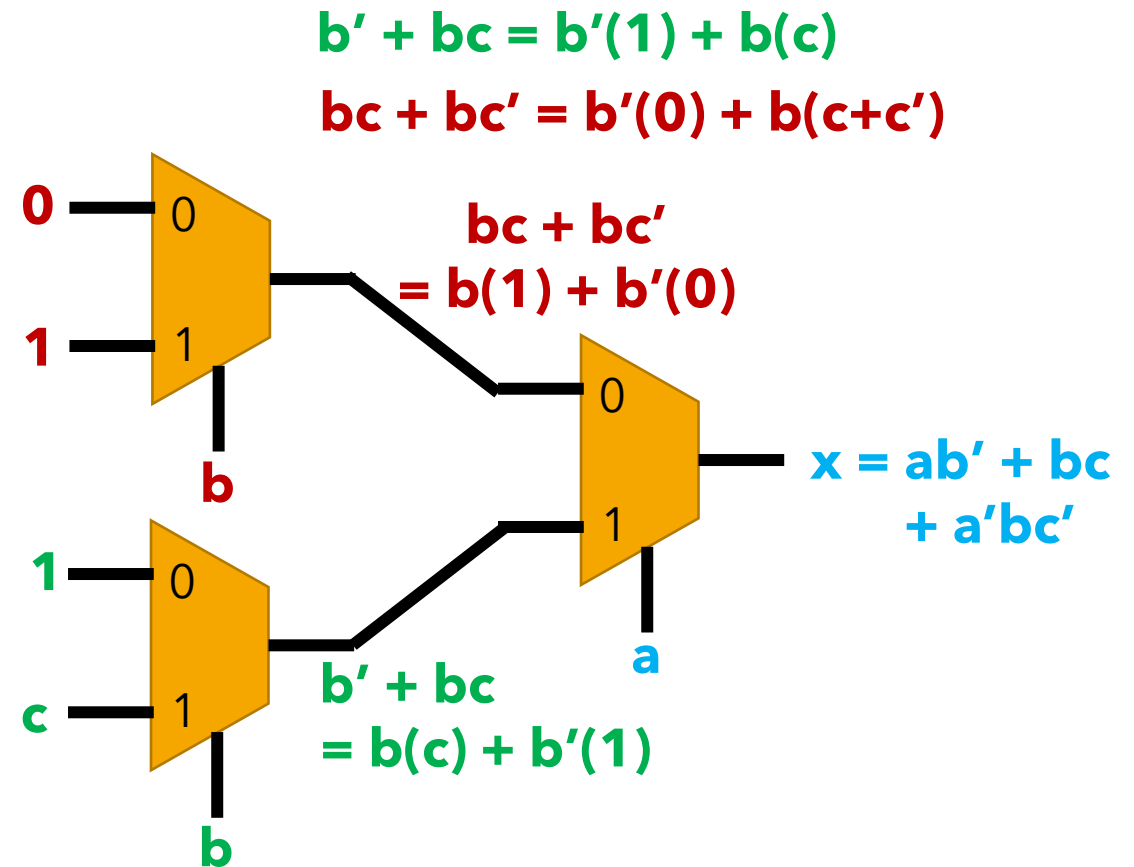
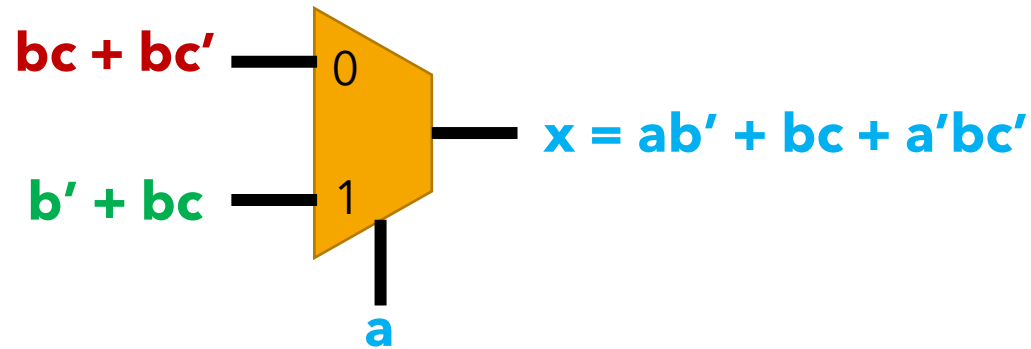
$$x = ab' + a'b$$
$$= a(b') + a'(b)$$

Any $f(a,b,c,...)$ can be written as:
 $a g(b,c,...) + a' h(b,c,...)$



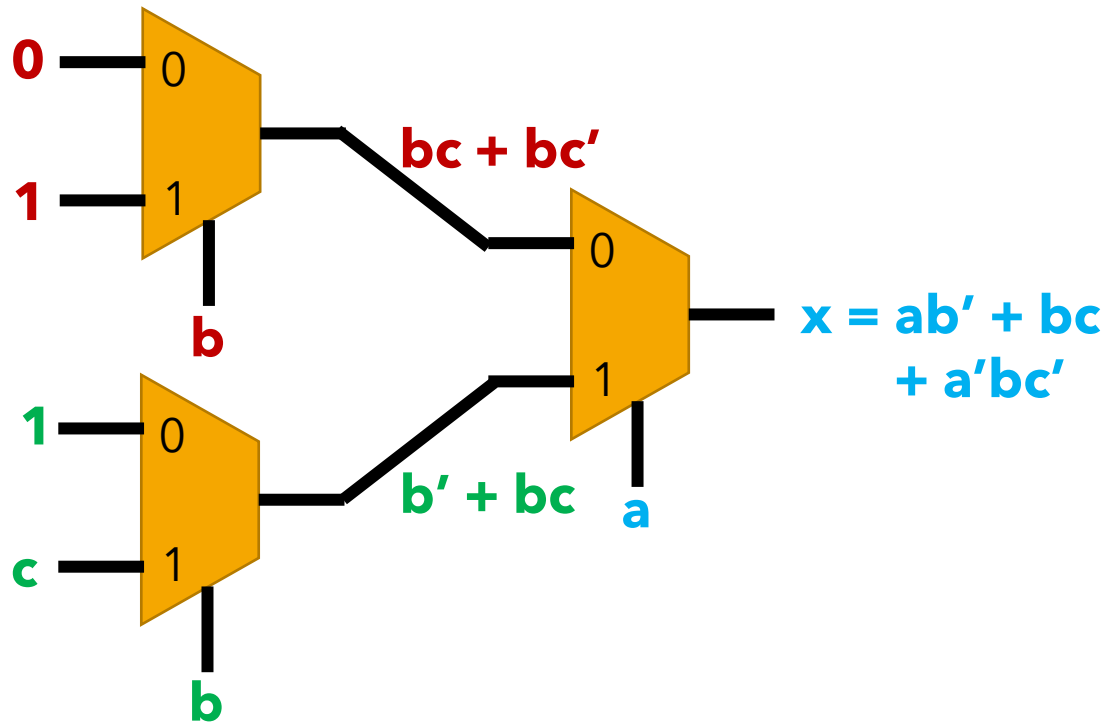
Implement ANY function with MUX

$$\begin{aligned}x &= ab' + bc + a'bc' \\&= ab' + (a + a')bc + a'bc' \\&= a(b' + bc) + a'(bc + bc')\end{aligned}$$

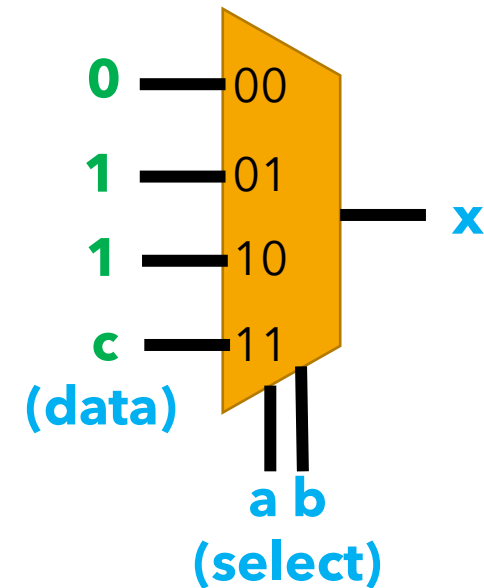


Implement with 4-to-1 MUX

$$\begin{aligned}x &= ab' + bc + a'bc' \\ &= a(b' + bc) + a'(bc + bc')\end{aligned}$$



$$\begin{aligned}x &= a(b' + bc) + a'(bc + bc') \\ &= a'b'0 + a'b(c + c') + ab'(1) + abc\end{aligned}$$



Equivalently, from Truth Table

a	b	c	x
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

$x = 0$

$x = c'$

$x = 1$

$x = c$

What function is **x** of **c**
for each **ab** value?

