

# Fundamentals of Computer Systems

## Combinational Logic

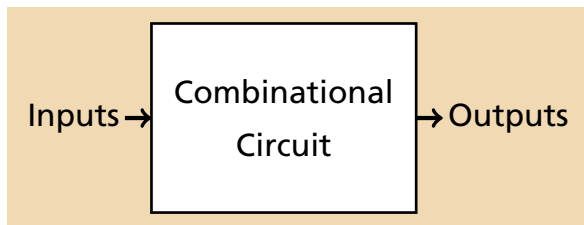
Stephen A. Edwards

Columbia University

Summer 2021

# Combinational Circuits

Combinational circuits are stateless.  
Their output is a function *only* of the current input.



## Basic Combinational Circuits

- Encoders and Decoders

- Priority Encoders

- Multiplexers

- Shifters

## Circuit Timing

- Critical Paths and Shortest Paths

- Glitches

## Arithmetic Circuits

- Half and Full Adders

- An Adder/Subtractor

- Overflow

- Carry Lookahead Adder

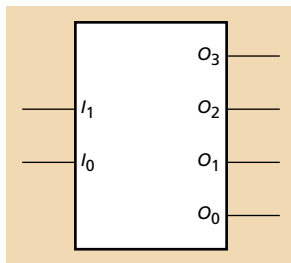


## Encoders and Decoders

## Overview: Decoder

A decoder takes a  $n$ -bit input and produces  $2^n$  single-bit outputs.

The binary input determines which output will be 1, all others 0. This is *one-hot encoding*.



# Decoders

---

<b>2-to-4</b>	
<b>in</b>	<b>out</b>
00	0001
01	0010
10	0100
11	1000

---

# Decoders

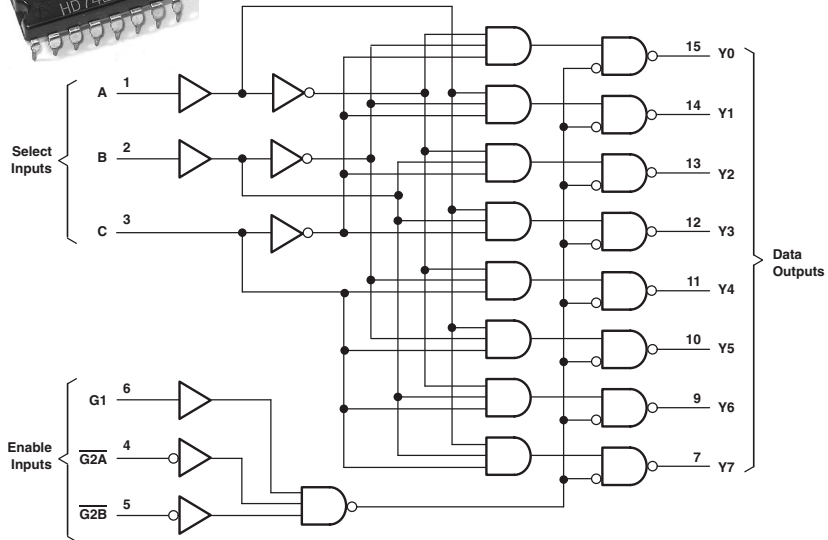
<b>2-to-4</b>		<b>3-to-8 decoder</b>	
<b>in</b>	<b>out</b>	<b>in</b>	<b>out</b>
00	0001	000	00000001
01	0010	001	00000010
10	0100	010	00000100
11	1000	011	00001000
		100	00010000
		101	00100000
		110	01000000
		111	10000000

# Decoders

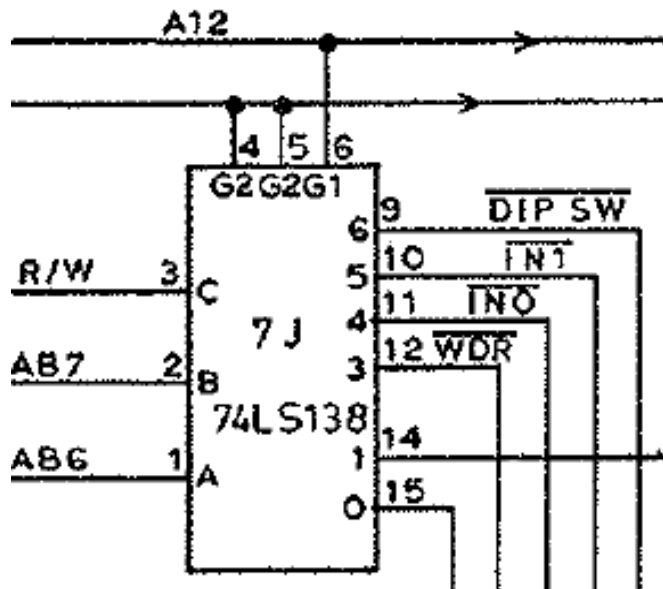
2-to-4		3-to-8 decoder		4-to-16 decoder	
in	out	in	out	in	out
00	0001	000	00000001	0000	0000000000000001
01	0010	001	00000010	0001	0000000000000010
10	0100	010	00000100	0010	0000000000000100
11	1000	011	00001000	0011	0000000000001000
		100	00010000	0100	0000000000010000
		101	00100000	0101	0000000000100000
		110	01000000	0110	0000000001000000
		111	10000000	0111	0000000010000000
				1000	0000000100000000
				1001	0000001000000000
				1010	0000010000000000
				1011	0000100000000000
				1100	0001000000000000
				1101	0010000000000000
				1110	0100000000000000
				1111	1000000000000000



# The 74138 3-to-8 Decoder

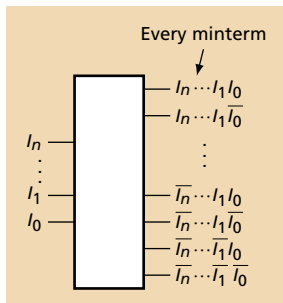


## A '138 Spotted in the Wild



Pac-Man (Midway, 1980)

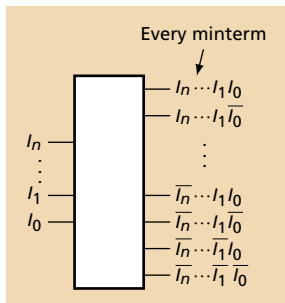
# General $n$ -bit Decoders



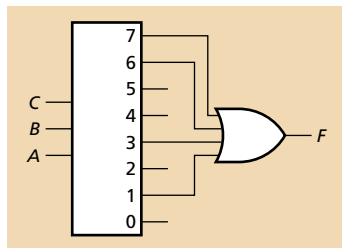
# General $n$ -bit Decoders

Implementing a function with a decoder:

E.g.,  $F = A\bar{C} + BC$



$C$	$B$	$A$	$F$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

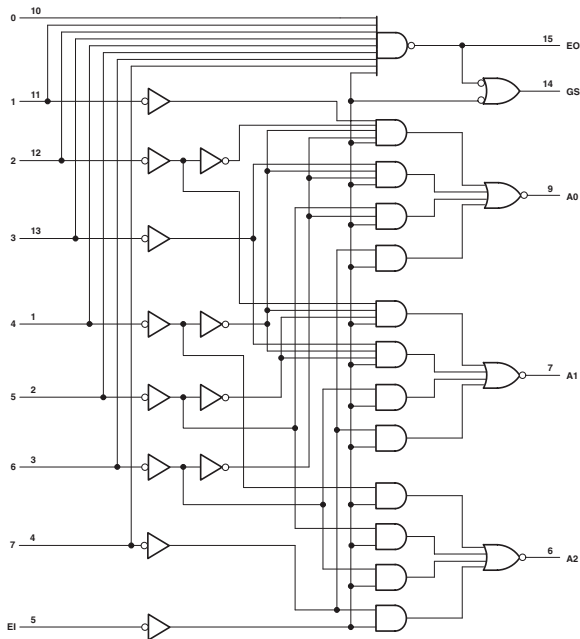


# The 74148 Priority Encoder

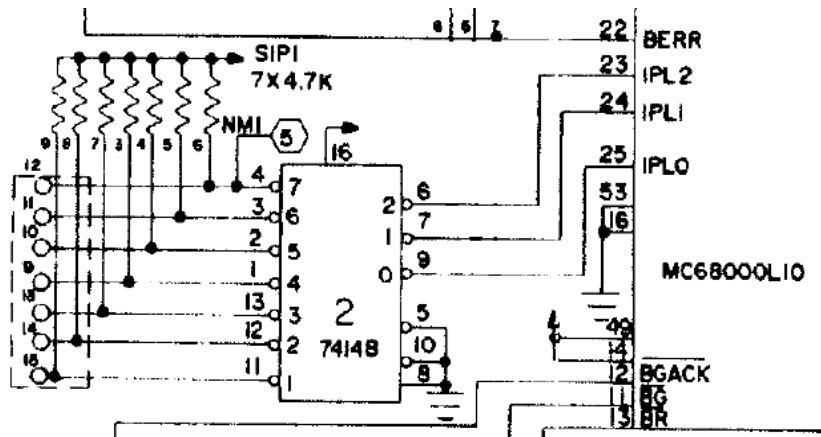
Input: 1-of- $2^n$

Output:  $n$ -bit  
binary number  
for *highest*  
*priority* input

	Inputs							Outputs		
E	0	1	2	3	4	5	6	7	10	GE
1	X	X	X	X	X	X	X	X	111	11
0	1	1	1	1	1	1	1	1	111	10
0	X	X	X	X	X	X	X	0	000	01
0	X	X	X	X	X	X	0	1	001	01
0	X	X	X	X	0	1	1		010	01
0	X	X	X	0	1	1	1		011	01
0	X	X	X	0	1	1	1	1	100	01
0	X	X	0	1	1	1	1	1	101	01
0	X	0	1	1	1	1	1	1	110	01
0	0	1	1	1	1	1	1	1	111	01



## A '148 Spotted in the Wild



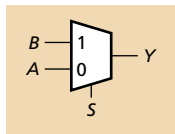
Users would connect wires to interrupt sources; pull-ups quiet unconnected interrupts

OB68K1A Single-board Computer (Omnibyte 1983)



Multiplexers

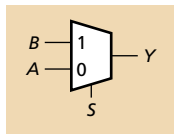
# The Two-Input Multiplexer



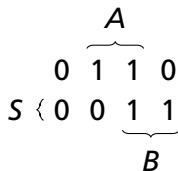
<i>S</i>	<i>B</i>	<i>A</i>	<i>Y</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



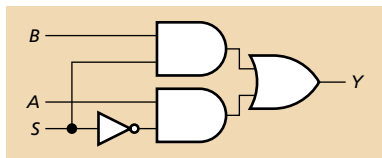
# The Two-Input Multiplexer



S	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

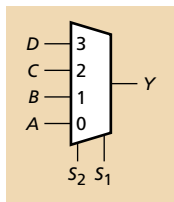


S	B	A	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

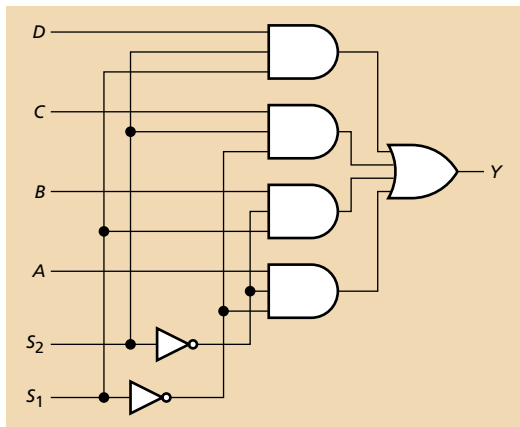


S	Y
0	A
1	B

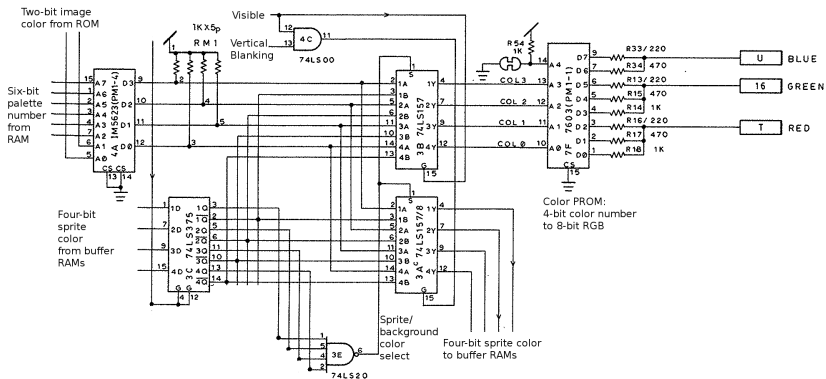
# The Four-Input Mux



$S_2$	$S_1$	Y
0	0	A
0	1	B
1	0	C
1	1	D



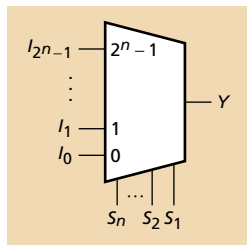
# Two-input Muxes in the Wild



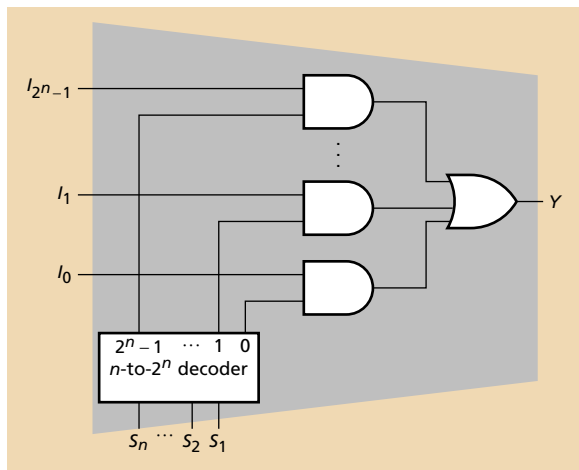
Quad 2-to-1 mux 3B selects color from a sprite or the background

Pac-Man (Midway, 1980)

# General $2^n$ -input muxes



$$\begin{aligned} Y &= I_0 \overline{S_n} \cdots \overline{S_2} \overline{S_1} + \\ & I_1 \overline{S_n} \cdots \overline{S_2} S_1 + \\ & I_2 \overline{S_n} \cdots S_2 \overline{S_1} + \\ & \vdots \\ & I_{2^n-2} S_n \cdots S_2 \overline{S_1} + \\ & I_{2^n-1} S_n \cdots S_2 S_1 \end{aligned}$$



## Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

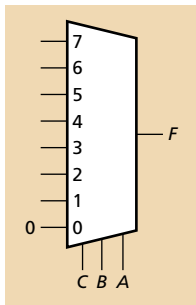
<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

# Using a Mux to Implement an Arbitrary Function

Apply each value in the truth table:

$$F = A\bar{C} + BC$$

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

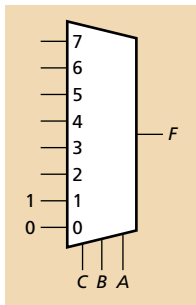


# Using a Mux to Implement an Arbitrary Function

Apply each value in the truth table:

$$F = A\bar{C} + BC$$

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

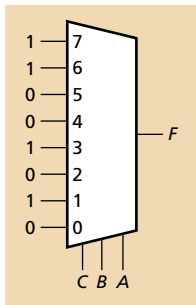


# Using a Mux to Implement an Arbitrary Function

Apply each value in the truth table:

$$F = A\bar{C} + BC$$

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1





## Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

## Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

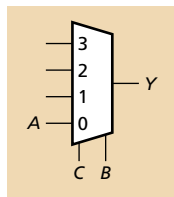
# Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

C	B	A	F
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

Can always remove a select and feed in 0, 1, S, or  $\bar{S}$ .

C	B	F
0	0	A
0	1	
1	0	
1	1	



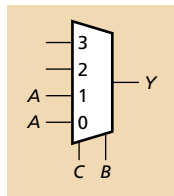
# Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

C	B	A	F
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

Can always remove a select and feed in 0, 1, S, or  $\bar{S}$ .

C	B	F
0	0	A
0	1	A
1	0	
1	1	



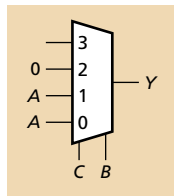
# Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

C	B	A	F
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

Can always remove a select and feed in 0, 1, S, or  $\bar{S}$ .

C	B	F
0	0	A
0	1	A
1	0	0
1	1	



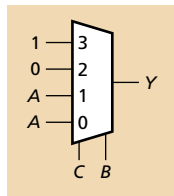
# Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

C	B	A	F
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

Can always remove a select and feed in 0, 1, S, or  $\bar{S}$ .

C	B	F
0	0	A
0	1	A
1	0	0
1	1	1



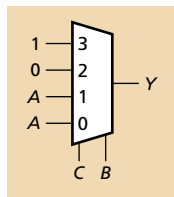
# Using a Mux to Implement an Arbitrary Function

$$F = A\bar{C} + BC$$

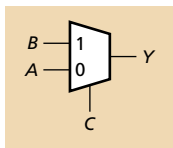
C	B	A	F
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

Can always remove a select and feed in 0, 1, S, or  $\bar{S}$ .

C	B	F
0	0	A
0	1	A
1	0	B
1	1	B



In this case, the function just happens to be a mux: (not always the case!)



## Using a Mux to Implement Another Function

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



# Using a Mux to Implement Another Function

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

# Using a Mux to Implement Another Function

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

# Using a Mux to Implement Another Function

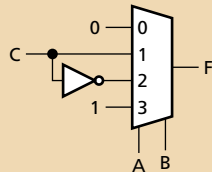
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	F
0	0	0
0	1	C
1	0	$\bar{C}$
1	1	1

# Using a Mux to Implement Another Function

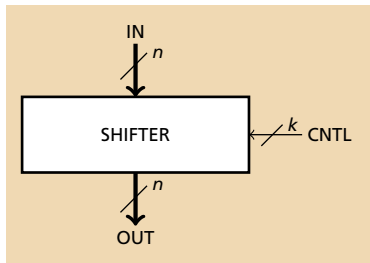
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	F
0	0	0
0	1	C
1	0	$\bar{C}$
1	1	1



# Shifters

A shifter shifts the inputs bits to the left or to the right.

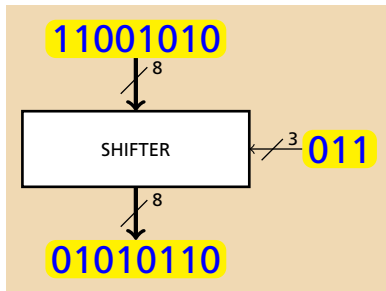


There are various types of shifters.

- ▶ Barrel: Selector bits indicate (in binary) how far to the left to shift the input.
- ▶ L/R with enable: Two control bits (upper enables, lower indicates direction).

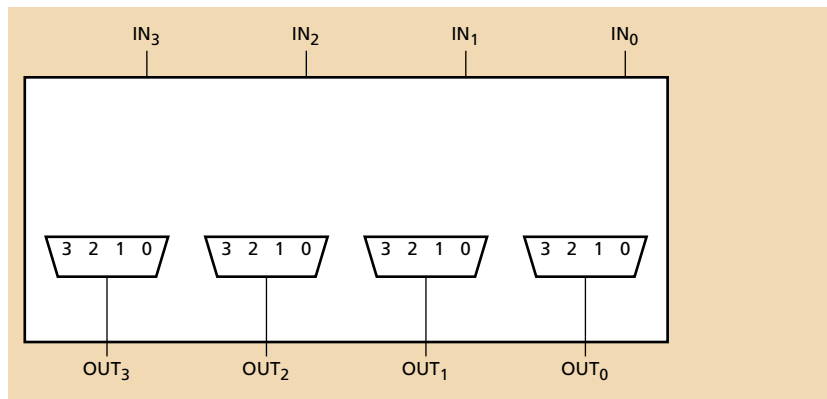
In either case, bits may “roll out” or “wrap around”

## Example: Barrel Shifter with Wraparound



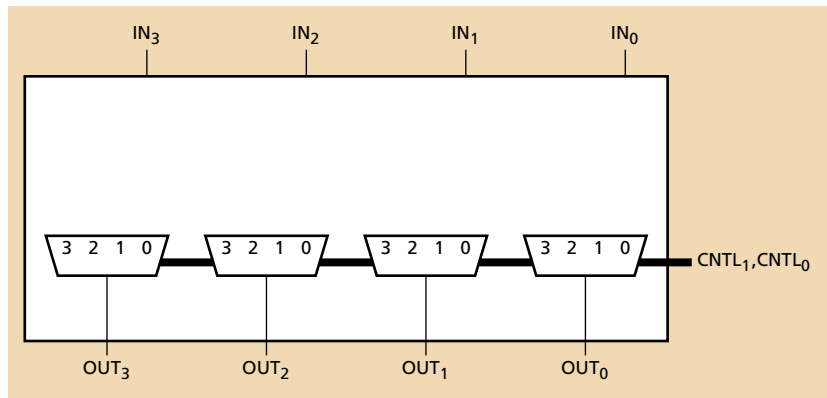
# A Barrel Shifter with Wraparound

Main idea: wire up all possible shift amounts and use muxes to select correct one.



# A Barrel Shifter with Wraparound

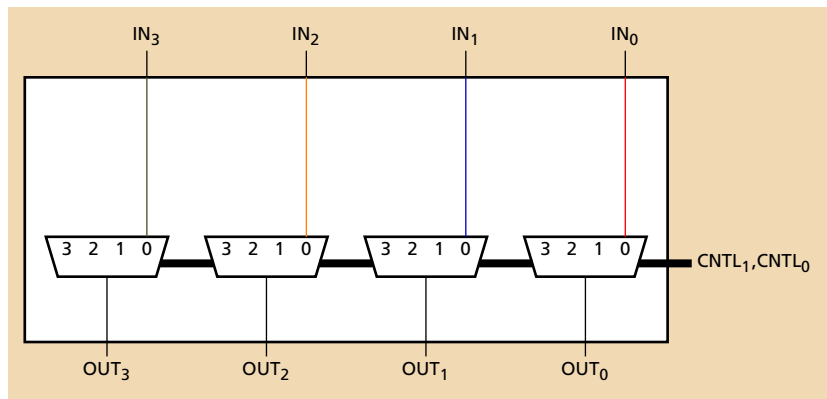
Main idea: wire up all possible shift amounts and use muxes to select correct one.





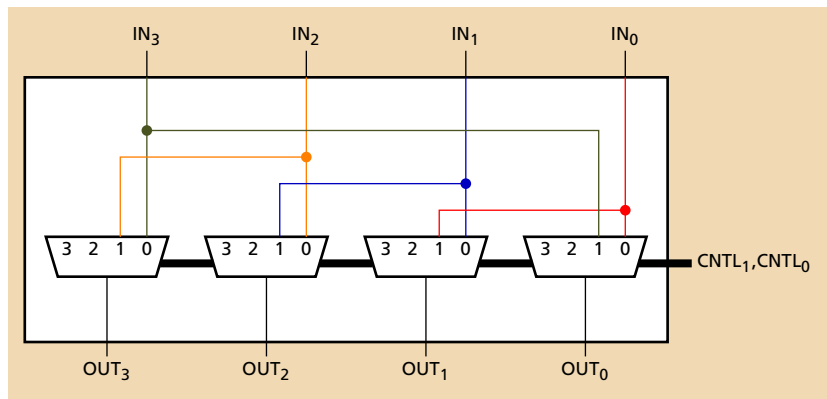
# A Barrel Shifter with Wraparound

Main idea: wire up all possible shift amounts and use muxes to select correct one.



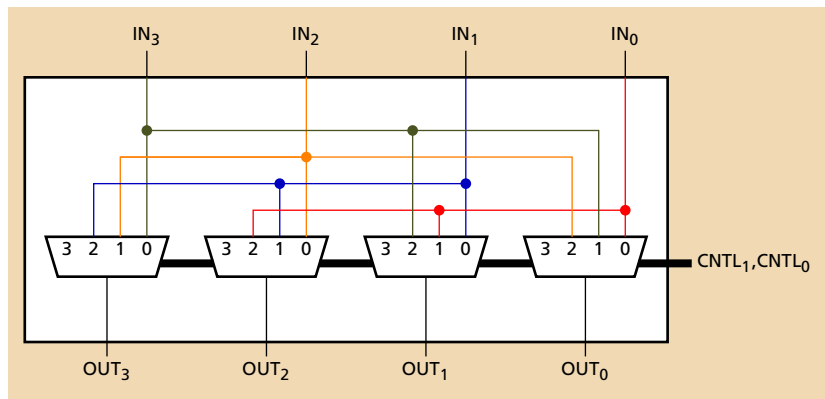
# A Barrel Shifter with Wraparound

Main idea: wire up all possible shift amounts and use muxes to select correct one.



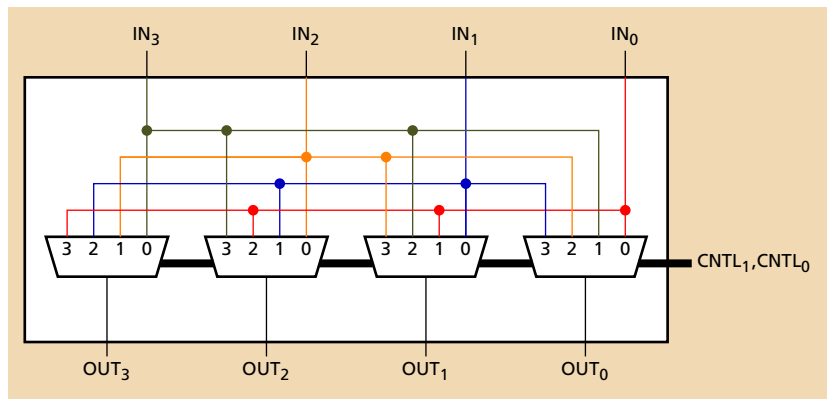
# A Barrel Shifter with Wraparound

Main idea: wire up all possible shift amounts and use muxes to select correct one.



# A Barrel Shifter with Wraparound

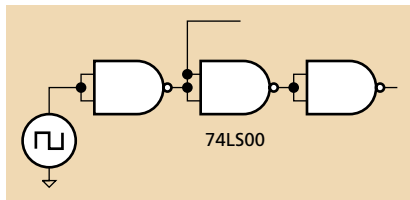
Main idea: wire up all possible shift amounts and use muxes to select correct one.





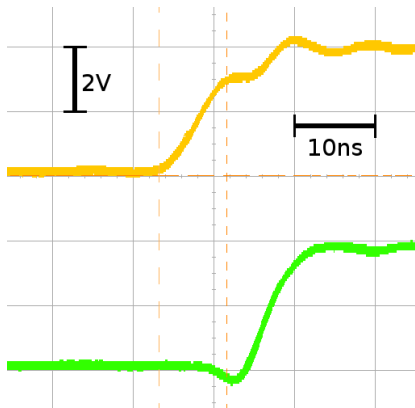
Timing

# Computation Always Takes Time

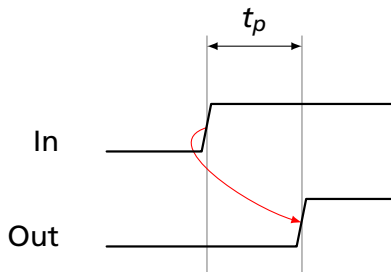


There is always a delay between inputs and outputs because

- ▶ Limited currents charging capacitance
- ▶ The speed of light

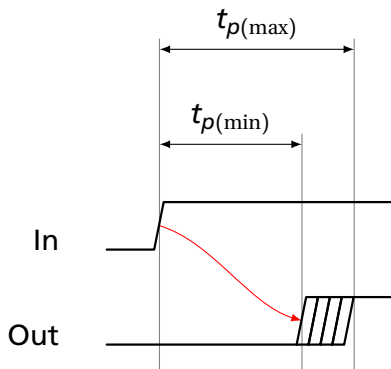


# The Simplest Timing Model



- ▶ Each gate has its own propagation delay  $t_p$ .
- ▶ When an input changes, any changing outputs do so after  $t_p$ .
- ▶ Wire delay is zero.

## A More Realistic Timing Model

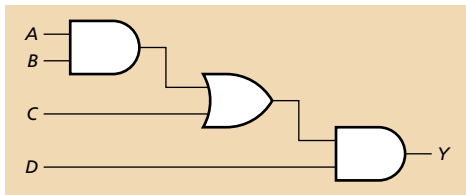


It is difficult to manufacture two gates with the same delay; better to treat delay as a range.

- ▶ Each gate has a minimum and maximum propagation delay  $t_{p(\min)}$  and  $t_{p(\max)}$ .
- ▶ Outputs may start changing after  $t_{p(\min)}$  and stabilize no later than  $t_{p(\max)}$ .

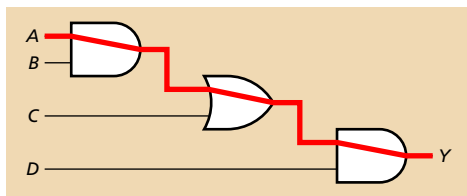


# Critical Paths and Shortest Paths



How slow can this be?

# Critical Paths and Shortest Paths

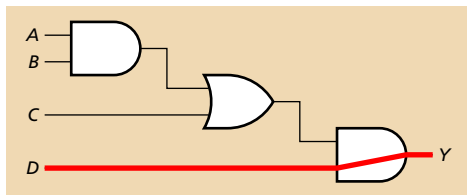


How slow can this be?

The **critical path** has the longest possible delay.

$$t_{p(\max)} = t_{p(\max, \text{AND})} + t_{p(\max, \text{OR})} + t_{p(\max, \text{AND})}$$

# Critical Paths and Shortest Paths



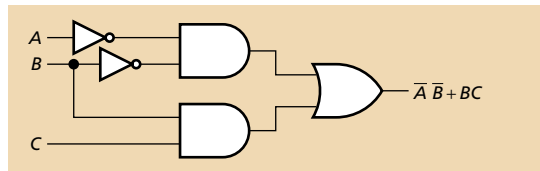
How fast can this be?

The **shortest path** has the least possible delay.

$$t_{p(\min)} = t_{p(\min, \text{AND})}$$

# Glitches

A glitch is when a single input change can cause multiple output changes.



	$B$			
	┌───┴───┐			
	1	0	0	0
$C$ {	1	1	1	0
		└───┬───┘		
			$A$	

An arrow points to the cell containing '1' in the second row and second column of the truth table.

$A$  \_\_\_\_\_

$C$  \_\_\_\_\_

$B$  \_\_\_\_\_

$\bar{B}$  \_\_\_\_\_

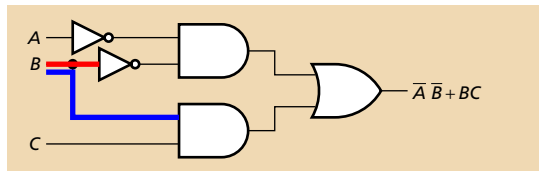
$\bar{A}\bar{B}$  \_\_\_\_\_

$BC$  \_\_\_\_\_

$\bar{A}\bar{B} + BC$  \_\_\_\_\_

# Glitches

A glitch is when a single input change can cause multiple output changes.



	$B$			
	1	0	0	0
$C$	1	1	1	0
	$A$			

$A$  \_\_\_\_\_

$C$  \_\_\_\_\_

$B$  \_\_\_\_\_

$\bar{B}$  \_\_\_\_\_

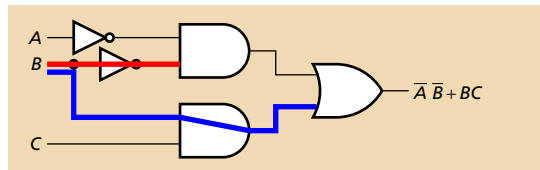
$\bar{A}\bar{B}$  \_\_\_\_\_

$BC$  \_\_\_\_\_

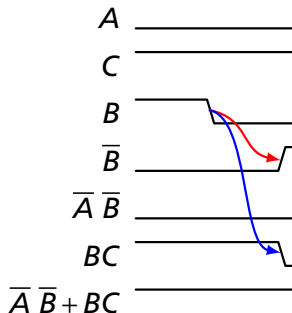
$\bar{A}\bar{B} + BC$  \_\_\_\_\_

# Glitches

A glitch is when a single input change can cause multiple output changes.

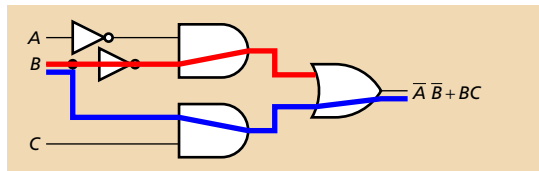


	$B$			
	1	0	0	0
$C$	1	1	1	0
	$A$			

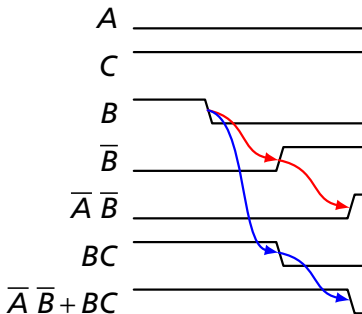


# Glitches

A glitch is when a single input change can cause multiple output changes.

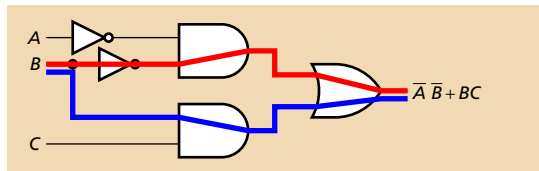


	<i>B</i>			
	1	0	0	0
<i>C</i>	1	1	1	0
	<i>A</i>			

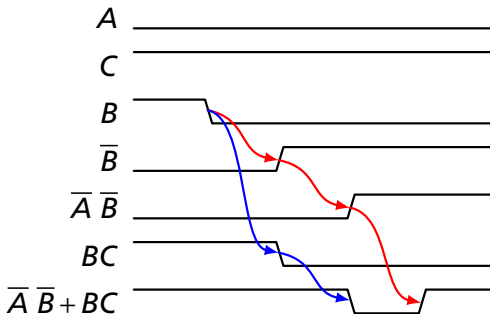


# Glitches

A glitch is when a single input change can cause multiple output changes.



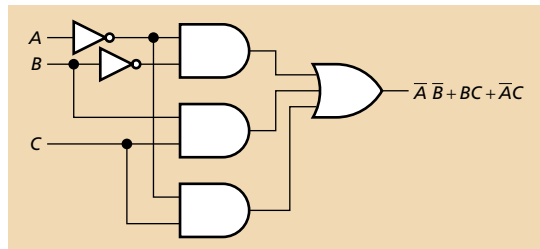
	$B$			
	1	0	0	0
$C$	1	1	1	0
	$A$			





# Glitches

A glitch is when a single input change can cause multiple output changes.



	$B$			
	┌───┴───┐			
	1	0	0	0
$C$ {	1	1	1	0
	└───┬───┘			
	$A$			

Adding such redundancy only works for single input changes; glitches may be unavoidable when multiple inputs change.



## Arithmetic Circuits

# Arithmetic: Addition

Adding two one-bit numbers:

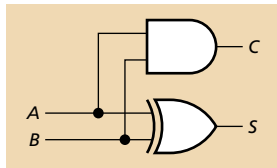
$A$  and  $B$

Produces a two-bit result:

$C$   $S$

(carry and sum)

$A$	$B$	$C$	$S$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Half Adder



Male Adder

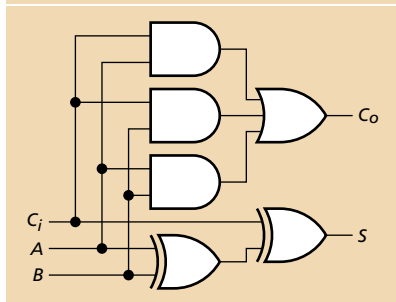
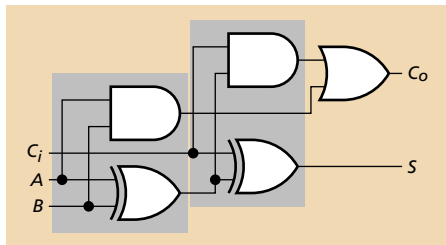
# Full Adder

In general, you  
need to add  
*three bits*:

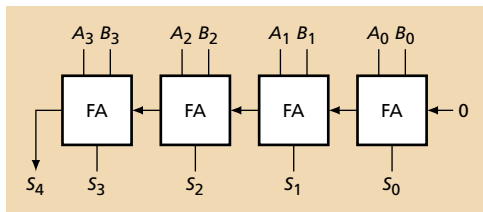
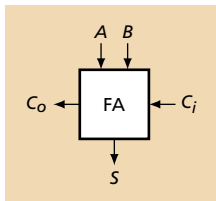
$$\begin{array}{r} 111000 \\ 111010 \\ + 11100 \\ \hline 1010110 \end{array}$$

$$\begin{array}{l} 0 + 0 = 00 \\ 0 + 1 + 0 = 01 \\ 0 + 0 + 1 = 01 \\ 0 + 1 + 1 = 10 \\ 1 + 1 + 1 = 11 \\ 1 + 1 + 0 = 10 \end{array}$$

$C_i A B$	$C_o S$
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

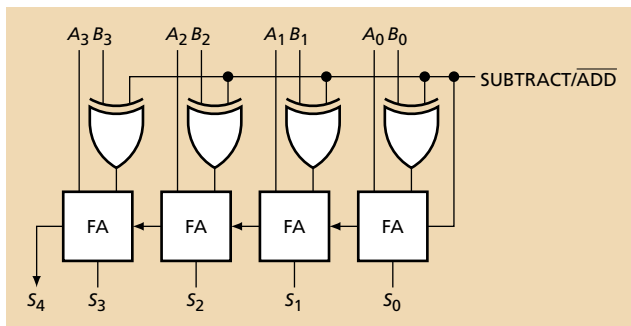


# A Four-Bit Ripple-Carry Adder



# A Two's Complement Adder/Subtractor

To subtract  $B$  from  $A$ , add  $A$  and  $-B$ .  
Neat trick: carry in takes care of the  $+1$  operation.



# Overflow in Two's-Complement Representation

When is the result too positive or too negative?

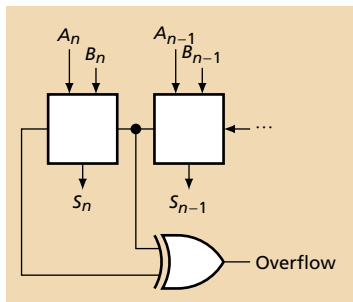
+	-2	-1	0	1
-2	$\begin{array}{r} 10 \\ 10 \\ +10 \\ \hline 00 \end{array}$			
-1	$\begin{array}{r} 10 \\ 10 \\ +11 \\ \hline 01 \end{array}$	$\begin{array}{r} 11 \\ 11 \\ +11 \\ \hline 10 \end{array}$		
0	$\begin{array}{r} 00 \\ 10 \\ +00 \\ \hline 10 \end{array}$	$\begin{array}{r} 00 \\ 11 \\ +00 \\ \hline 11 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +00 \\ \hline 00 \end{array}$	
1	$\begin{array}{r} 00 \\ 10 \\ +01 \\ \hline 11 \end{array}$	$\begin{array}{r} 11 \\ 11 \\ +01 \\ \hline 00 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +01 \\ \hline 01 \end{array}$	$\begin{array}{r} 01 \\ 01 \\ +01 \\ \hline 10 \end{array}$

# Overflow in Two's-Complement Representation

When is the result too positive or too negative?

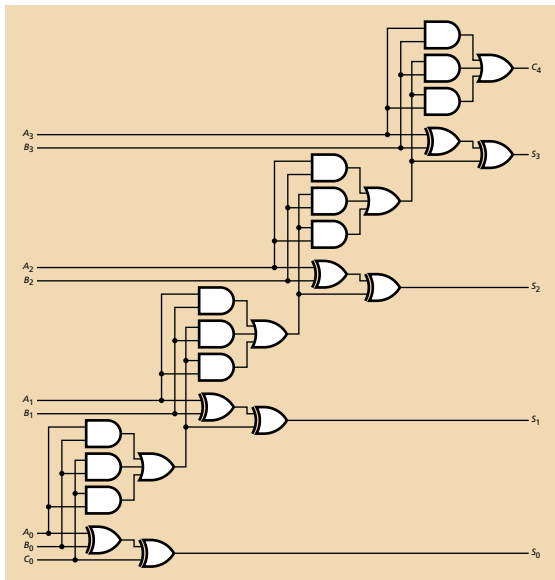
+	-2	-1	0	1
-2	$\begin{array}{r} 10 \\ 10 \\ +10 \\ \hline 00 \end{array} \times$			
-1	$\begin{array}{r} 10 \\ 10 \\ +11 \\ \hline 01 \end{array} \times$	$\begin{array}{r} 11 \\ 11 \\ +11 \\ \hline 10 \end{array}$		
0	$\begin{array}{r} 00 \\ 10 \\ +00 \\ \hline 10 \end{array}$	$\begin{array}{r} 00 \\ 11 \\ +00 \\ \hline 11 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +00 \\ \hline 00 \end{array}$	
1	$\begin{array}{r} 00 \\ 10 \\ +01 \\ \hline 11 \end{array}$	$\begin{array}{r} 11 \\ 11 \\ +01 \\ \hline 00 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +01 \\ \hline 01 \end{array}$	$\begin{array}{r} 01 \\ 01 \\ +01 \\ \hline 10 \end{array} \times$

The result does not fit when the top two carry bits differ.





## Ripple-Carry Adders are Slow



The *depth* of a circuit is the number of gates on a critical path.

This four-bit adder has a depth of 8.

$n$ -bit ripple-carry adders have a depth of  $2n$ .

# Carry Generate and Propagate

The carry chain is the slow part of an adder; carry-lookahead adders reduce its depth using the following trick:

	A				
	0	0	1	0	
C	{	0	1	1	1
			B		

For bit  $i$ ,

$$\begin{aligned}C_{i+1} &= A_i B_i + A_i C_i + B_i C_i \\ &= A_i B_i + C_i (A_i + B_i) \\ &= G_i + C_i P_i\end{aligned}$$

K-map for the carry-out function of a full adder

Generate  $G_i = A_i B_i$  sets carry-out regardless of carry-in.

Propagate  $P_i = A_i + B_i$  copies carry-in to carry-out.

# Carry Lookahead Adder

Expand the carry functions into sum-of-products form:

$$C_{i+1} = G_i + C_i P_i$$

$$C_1 = G_0 + C_0 P_0$$

$$C_2 = G_1 + C_1 P_1$$

$$= G_1 + (G_0 + C_0 P_0) P_1$$

$$= G_1 + G_0 P_1 + C_0 P_0 P_1$$

$$C_3 = G_2 + C_2 P_2$$

$$= G_2 + (G_1 + G_0 P_1 + C_0 P_0 P_1) P_2$$

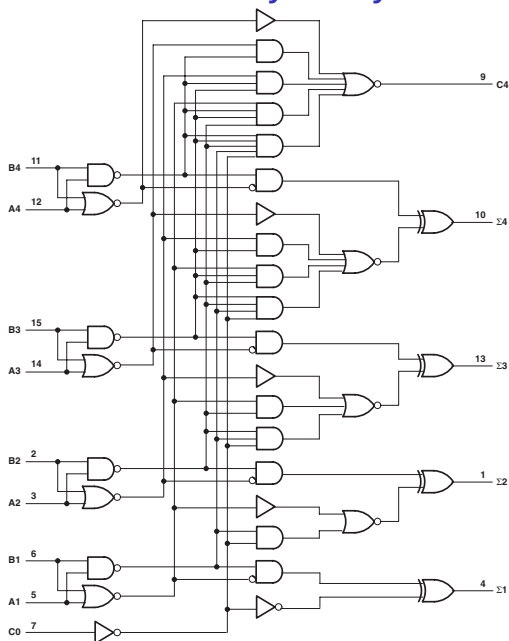
$$= G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2$$

$$C_4 = G_3 + C_3 P_3$$

$$= G_3 + (G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2) P_3$$

$$= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3$$

# The 74283 Binary Carry-Lookahead Adder



Carry out  $i$  has  $i + 1$  product terms, largest of which has  $i + 1$  literals.

If wide gates don't slow down, delay is independent of number of bits.

More realistic: if limited to two-input gates, depth is  $O(\log_2 n)$ .