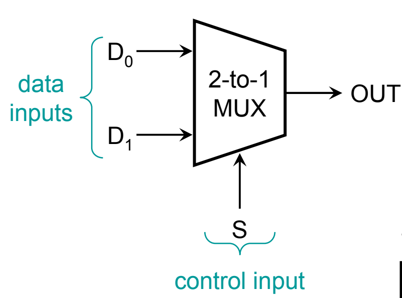
14:332:231
DIGITAL LOGIC DESIGN

Ivan Marsic, Rutgers University
Electrical & Computer Engineering
Fall 2013

Lecture #12: Multiplexers, Exclusive OR Gates, and Parity Circuits

Multiplexers (Data Selectors)

- A multiplexer (MUX for short) is a *digital switch*:
  - it passes (connects) one of its data inputs to the output
  - the data input selected is a function of a set of control inputs called *selection inputs*



S	OUT
0	D_0
1	D_1

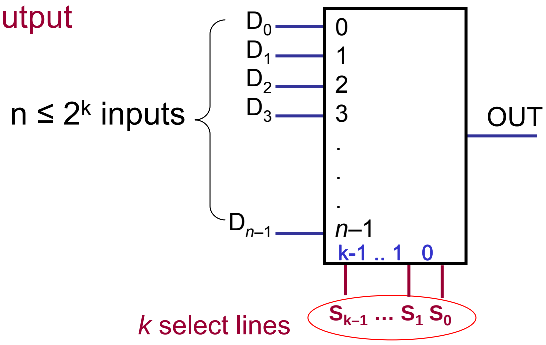
S	D_1	D_0	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Two alternative forms for a 2:1 MUX truth table

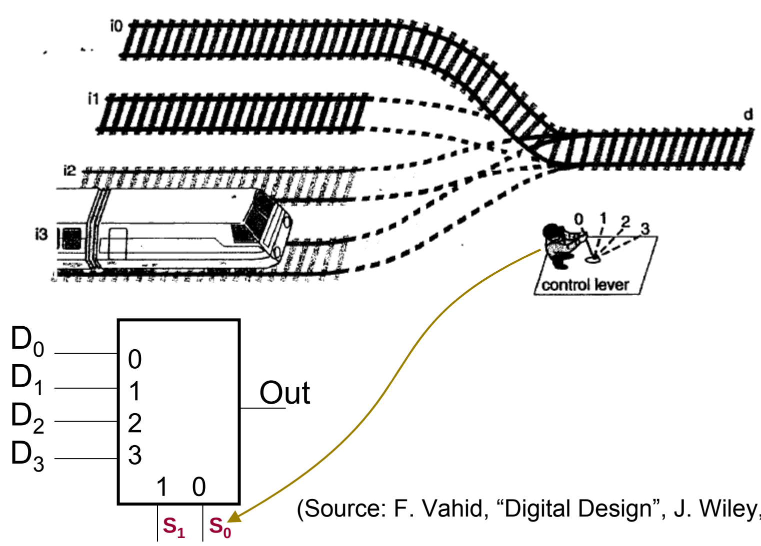
$OUT = S' \cdot D_0 + S \cdot D_1$

Multiplexers Do Selecting

- Selecting of data or information is a critical function in digital systems and computers
- Circuits that perform selecting have:
  - A set of *n* information inputs D_i from which the selection is made
  - A set of *k* control (select) lines for making the selection
  - A single output



Multiplexer Analogy



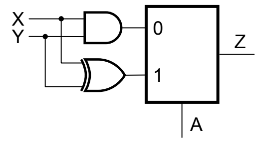
(Source: F. Vahid, “Digital Design”, J. Wiley, 2007)

Example Uses of Multiplexers

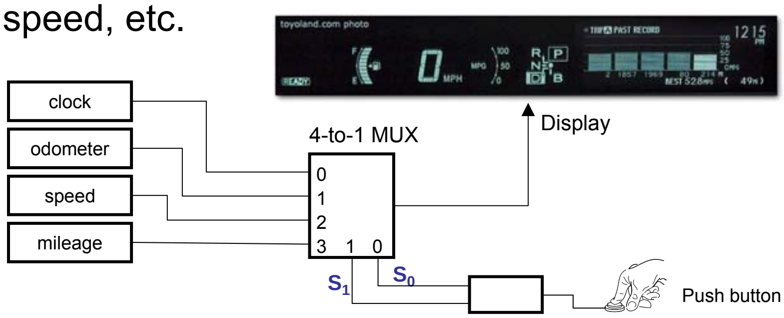
- In computers to select among signals
- To implement command:

  if A=0 then Z=X·Y

  else Z=X⊕Y



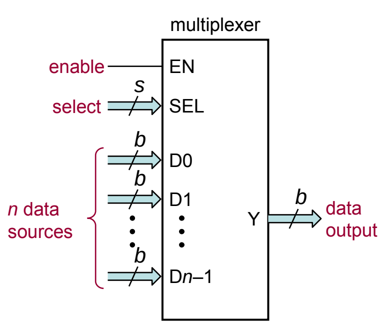
- Trip controller in a car to display mileage, time, speed, etc.

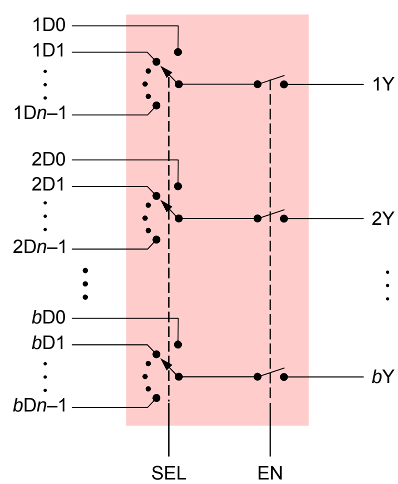


5 of 31

Multiplexer Structure

- Switch circuit equivalent
- Multiplexer is unidirectional





6 of 31

Example: 4-to-1-line Multiplexer

- Expression for OUT:

$$OUT = \underbrace{S_1'\cdot S_0'}_{M_0}\cdot D_0 + \underbrace{S_1'\cdot S_0}_{M_1}\cdot D_1 + \underbrace{S_1\cdot S_0'}_{M_2}\cdot D_2 + \underbrace{S_1\cdot S_0}_{M_3}\cdot D_3$$

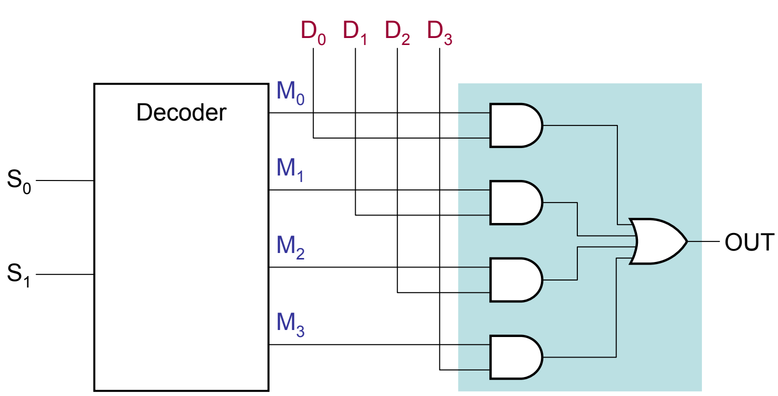
or: $OUT = \sum_{j=0}^{2^k-1} M_j\cdot D_j$

S_1	S_0	OUT
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- Circuit implementation: Sum-of-Products
  - 4 AND gates (4 product terms)
  - 2-to-4 line decoder (to generate the minterms)

Example: 4-to-1-line Multiplexer

- 2-to-2^2-line decoder
- $2^2 \times 2$ AND-OR



General Multiplexer Equation

- A general logic equation for a multiplexer output:

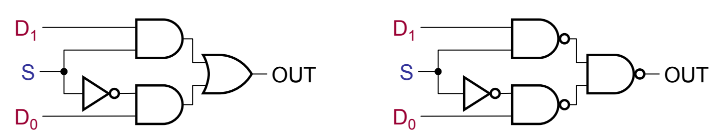
$$iY = \sum_{j=0}^{n-1} EN \cdot M_j \cdot iDj$$

- Logical sum of product terms
- Variable iY is a particular output bit (1 ≤ i ≤ b)
- Mj is a minterm *j* of the *s* select inputs

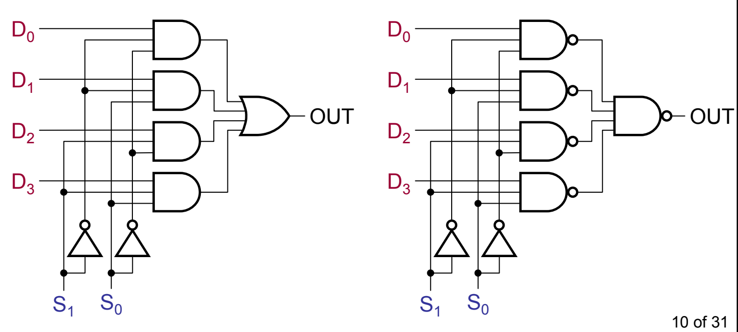
Gate Level Implementation of MUXs

positive logic: negative logic:

- 2:1 MUX



- 4:1 MUX



Multiplexer Standard Packaging

IC has limited number of pins (16)

$n \cdot b + b + s + 1 \leq 16 - 2$

in out SEL EN

n data inputs
b bits per input
s select inputs

$(n+1) \cdot b + \lceil \log_2 n \rceil \leq 13$

b	n	s			
1	8	3	(12)	8 input 1 bit	74x151
2	4	2	(12)	dual 4 input 2 bit	74x153
4	2	1	(13)	quad 2 input 4 bit	74x157

11 of 31

Truth table of 74x151

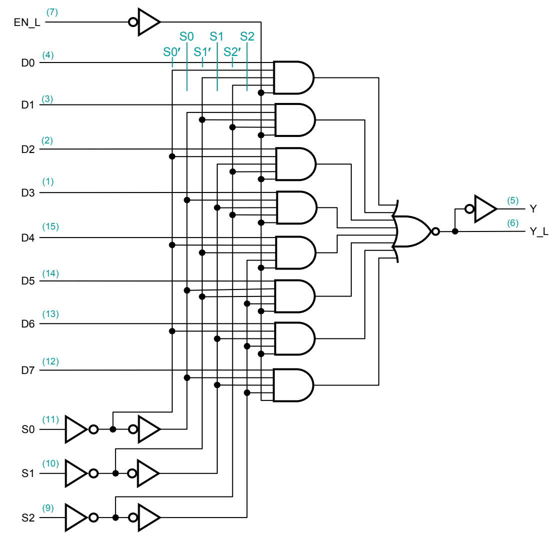
- Truth table for 74x151 8-input, 1-bit multiplexer
- Only “control” inputs are listed under “Inputs”
- Outputs specified as 0” or “1”, or a simple logic function of “data” inputs (e.g., D0 or D0′)

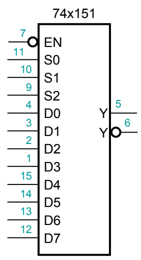
Inputs				*Outputs*	
EN_L	**S2**	**S1**	**S0**	**Y**	**Y_L**
1	x	x	x	0	1
0	0	0	0	D0	D0′
0	0	0	1	D1	D1′
0	0	1	0	D2	D2′
0	0	1	1	D3	D3′
0	1	0	0	D4	D4′
0	1	0	1	D5	D5′
0	1	1	0	D6	D6′
0	1	1	1	D7	D7′

12 of 31

74x151 8-input 1-bit Multiplexer

- 74x151
  logic diagram
  and
  logic symbol

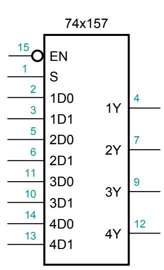


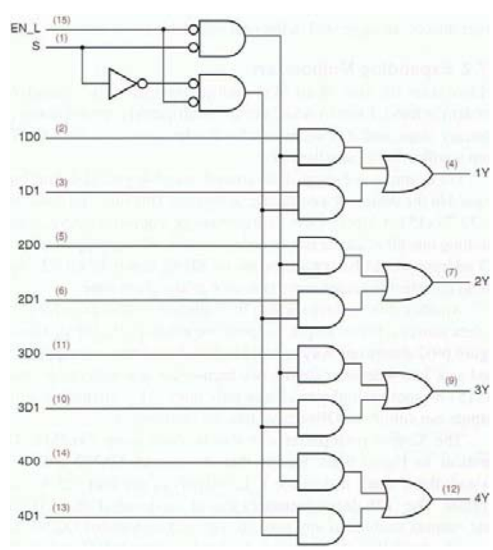


13 of 31

74x157 2-input 4-bit Multiplexer

- 74x157
  selects
  between two
  4-bit inputs

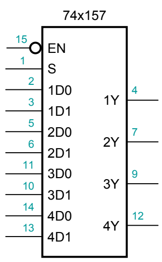




14 of 31

74x157 2-input 4-bit Multiplexer

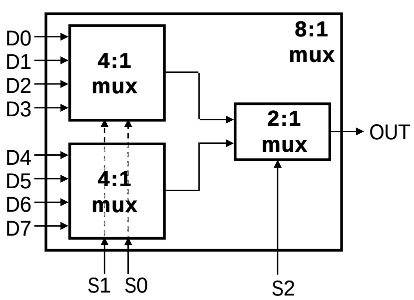
- 74x157
  truth table:



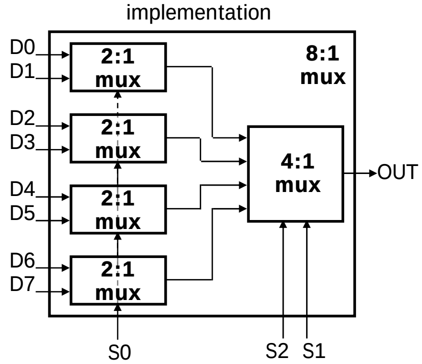
Inputs		*Outputs*			
EN_L	S	1Y	2Y	3Y	4Y
1	x	0	0	0	0
0	0	1D0	2D0	3D0	4D0
0	1	1D1	2D1	3D1	4D1

Cascading/Expanding Multiplexers

- Large multiplexers can be made by cascading smaller ones



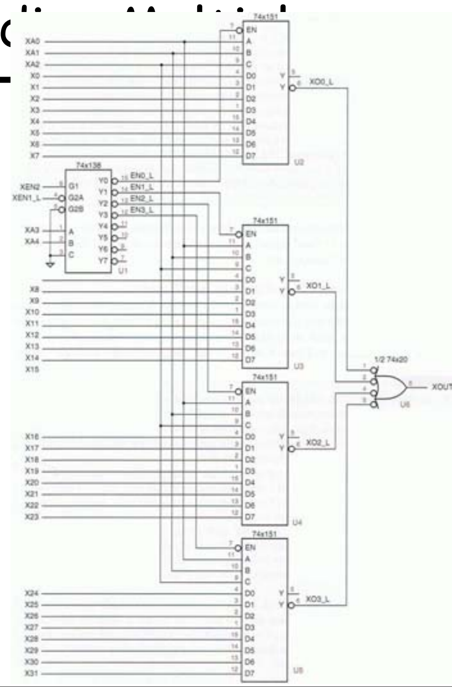
alternative implementation



Control signals **S1** and **S0** simultaneously choose one of D0, D1, D2, D3 and one of D4, D5, D6, D7

Control signal **S2** chooses which of the upper or lower mux's output to gate to OUT

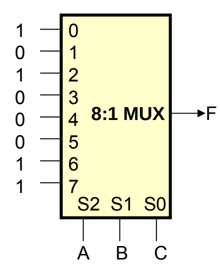
Cascading/Expanc



- Combining four 74x151s to make a 32-to-1 multiplexer
- 74x138 3-to-8 decoder used as 2-to-4 decoder for two high-order bits to enable one of 74x151s

Multiplexers as General-purpose Logic

- A 2^n:1 multiplexer can implement any function of *n* variables
  - with the variables used as control inputs and
  - the data inputs tied to 0 or 1



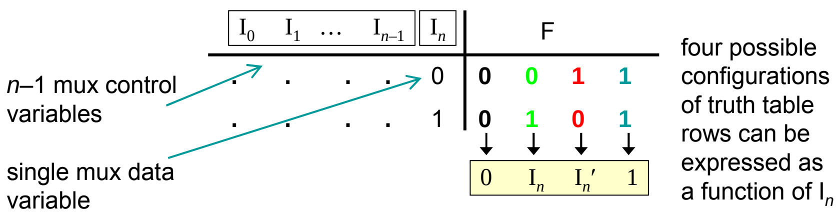
- Example:

F(A,B,C) = $M_0 + M_2 + M_6 + M_7$
= A'·B'·C' + A'·B·C' + A·B·C' + A·B·C
= A'·B'·C'·(1) + A'·B'·C·(0) + A'·B·C'·(1) + A'·B·C·(0) + A·B'·C'·(0) + A·B'·C·(0) + A·B·C'·(1) + A·B·C·(1)

OUT = A'·B'·C'·I0 + A'·B'·C·I1 + A'·B·C'·I2 + A'·B·C·I3 + A·B'·C'·I4 + A·B'·C·I5 + A·B·C'·I6 + A·B·C·I7

Multiplexers as General-purpose Logic

- *Generalization*:
  data inputs can also be tied to variables not just 0's an 1's

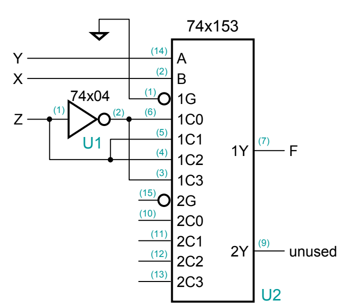


Multiplexers as Function Generators

- We can generate the four possible Z values (0, 1, Z, Z′) and realize the function with half the values
- Realizing $F = \sum_{X,Y,Z}(0,3,5,6)$ with a 4-input multiplexer:

$I_0\ I_1\ \dots\ I_{n-1}$ | I_n

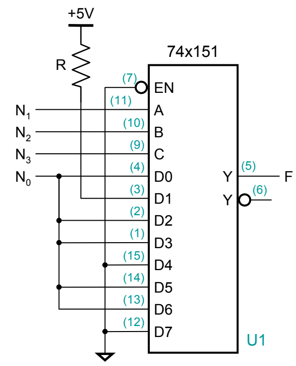
Row	X	Y	Z	F	
0	0	0	0	**1**	Z′
1	0	0	1	0	
2	0	1	0	0	Z
3	0	1	1	**1**	
4	1	0	0	0	Z
5	1	0	1	**1**	
6	1	1	0	**1**	Z′
7	1	1	1	0	



4-variable Function using 8-input Multiplexer

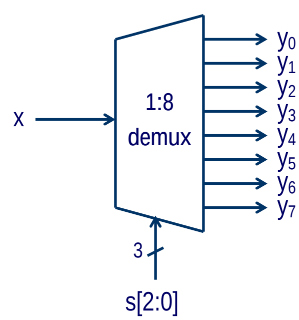
- Realizing $F = \sum_{N_0,N_1,N_2,N_3}(1,2,3,5,7,11,13)$ with an 8-input multiplexer:

Row	N_3	N_2	N_1	N_0	F	
0	0	0	0	0	0	N_0
1	0	0	0	1	1	
2	0	0	1	0	1	1
3	0	0	1	1	1	
4	0	1	0	0	0	N_0
5	0	1	0	1	1	
6	0	1	1	0	0	N_0
7	0	1	1	1	1	
8	1	0	0	0	0	0
9	1	0	0	1	0	
10	1	0	1	0	0	N_0
11	1	0	1	1	1	
12	1	1	0	0	0	N_0
13	1	1	0	1	1	
14	1	1	1	0	0	0
15	1	1	1	1	0	



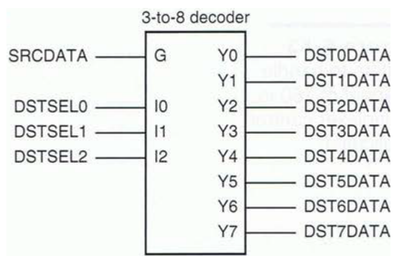
Demultiplexers

- Route a single input to one of many outputs, as a function of a set of control inputs



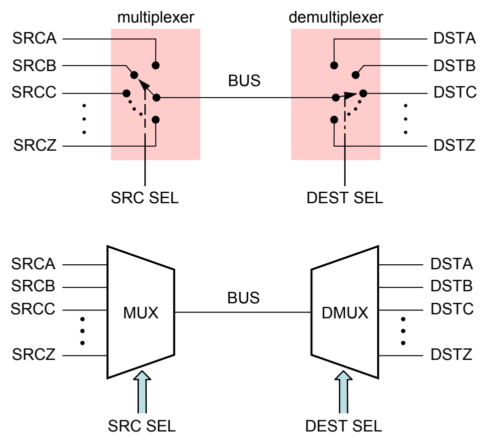
Demultiplexers

- Demultiplexer function is just the inverse of multiplexer's
- A *binary decoder* with enable input can be used as a demultiplexer



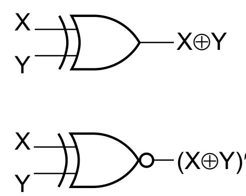
Demultiplexer Analogy

- A MUX driving a bus and a demultiplexer receiving the bus



Exclusive-OR Gates

- **Exclusive-OR (XOR)** gate is a 2-input gate whose output is "1" if exactly one of its input is "1"
  - or: XOR gate produces a "1" output if its inputs are *different*
- **Exclusive-NOR (XNOR)** or **Equivalence** just opposite—produces output "1" if its inputs are the same
- XOR: $X \oplus Y = X' \cdot Y + X \cdot Y'$
- XNOR: $(X \oplus Y)' = X \cdot Y + X' \cdot Y'$

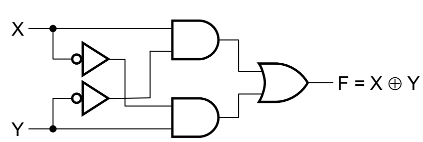


Exclusive-OR Gates

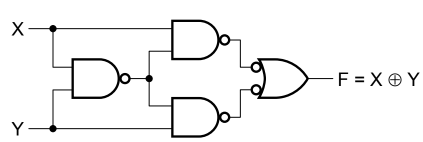
- Truth table and gate-level implementation

X	Y	X ⊕ Y (XOR)	(X ⊕ Y)′ (XNOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

AND-OR implementation:



three-level NAND implementation:



$(X \cdot Y)' \cdot (X+Y) = (X'+Y') \cdot (X+Y) = X' \cdot Y + Y' \cdot X$

XOR versus XNOR

$(X \oplus Y)' = (X' \cdot Y + X \cdot Y')' = (X + Y') \cdot (X' + Y)$
$= X \cdot X' + X' \cdot Y' + X \cdot Y + Y' \cdot Y$
$= X \cdot Y + X' \cdot Y'$

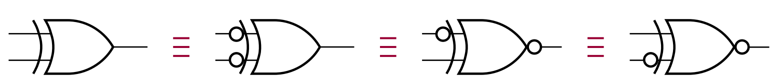
$X \rightarrow X'$ out $\rightarrow$ out′ … the circuit XOR = XOR
$(X' \cdot Y' + X \cdot Y)' = X' \cdot Y + X \cdot Y'$

$X \rightarrow X'$ out $\rightarrow$ out′ … the circuit XNOR = XNOR
$X' \cdot Y' + X \cdot Y$

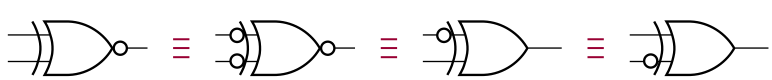
$X \oplus 1 = X \cdot 0 + X' \cdot 1 = X'$

Equivalent Symbols for XOR/XNOR

- XOR gates



- XNOR gates

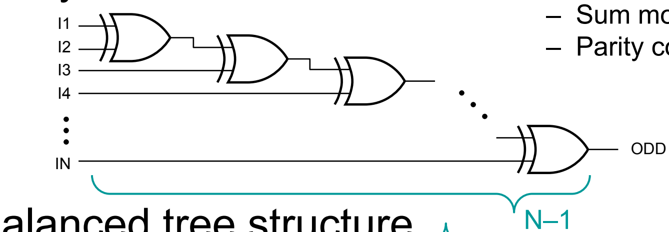


- Simple rule:
  - Any two signals (inputs or output) of an XOR or XNOR gate may be complemented w/o changing the resulting logic function

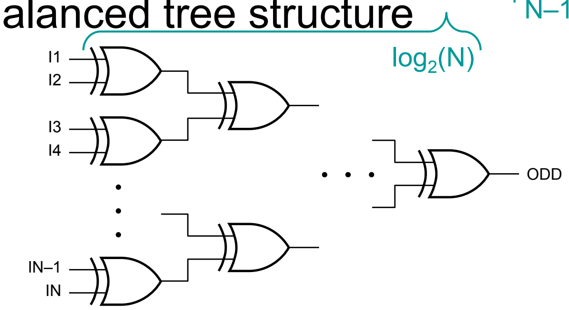
$X \oplus 0 = X$ $X \oplus 1 = X'$

Cascading XOR gates

- Daisy-chain connection
  - Sum modulo 2
  - Parity computation



- Balanced tree structure

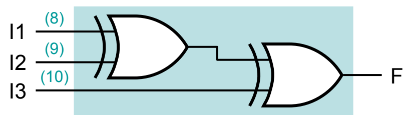


Cascading XOR gates

- The tree structure is faster than daisy-chain connection because the gates depth of its treelike structure is $\log_2(N)$, which is much less than N–1 for a daisy-chain structure
- Both are called **odd-parity circuits** because its output is “1” if an *odd number of inputs* are “1”
  - Used to generate and check parity bits in computer systems
    - Detects any single-bit error
- **Even-parity circuit** has odd-parity circuit’s output inverted—its output is “1” if an *even number* of its inputs are “1”

Example Parity Computation

F = I1 ⊕ I2 ⊕ I3 F = 1 ODD number of "1" in the input



I1	I2	I3	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

31 of 31