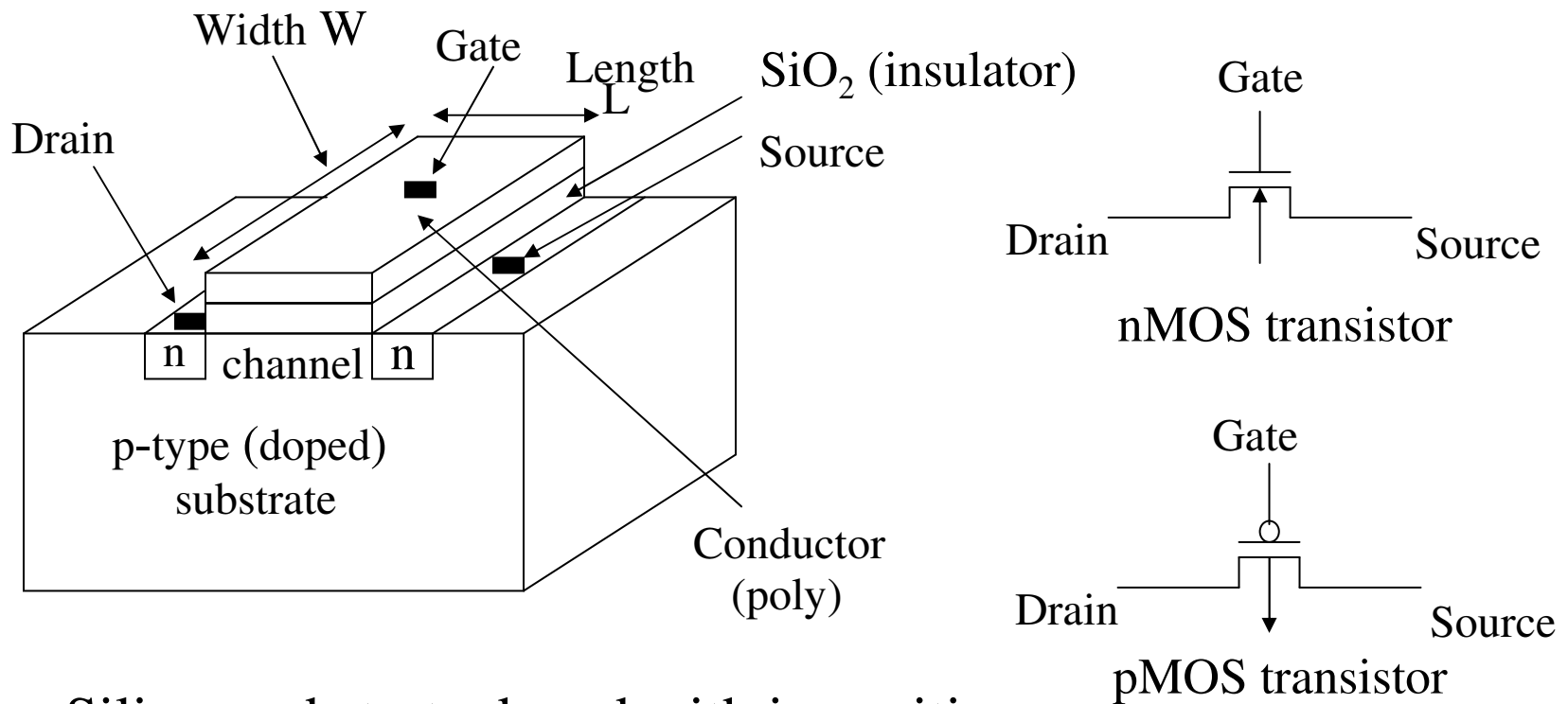


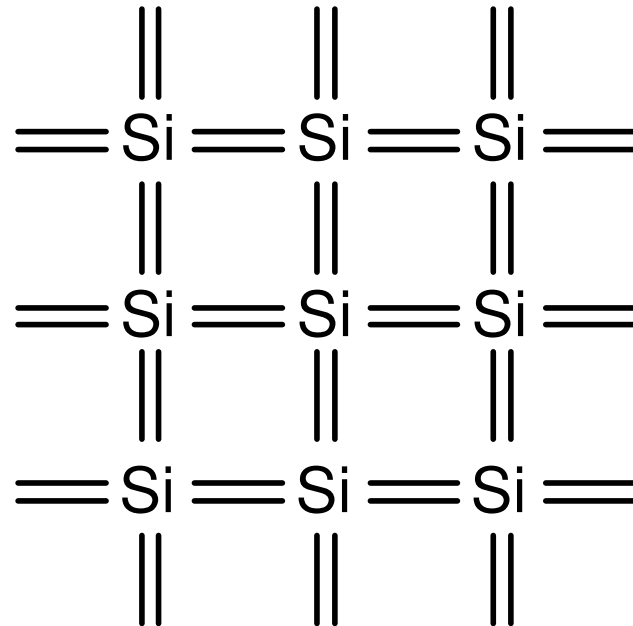
# MOS Transistors



- Silicon substrate doped with impurities
- Adding or cutting away insulating glass ( $\text{SiO}_2$ )
- Adding wires made of polycrystalline silicon (*polysilicon*, *poly*) or metal, insulated from the substrate by  $\text{SiO}_2$

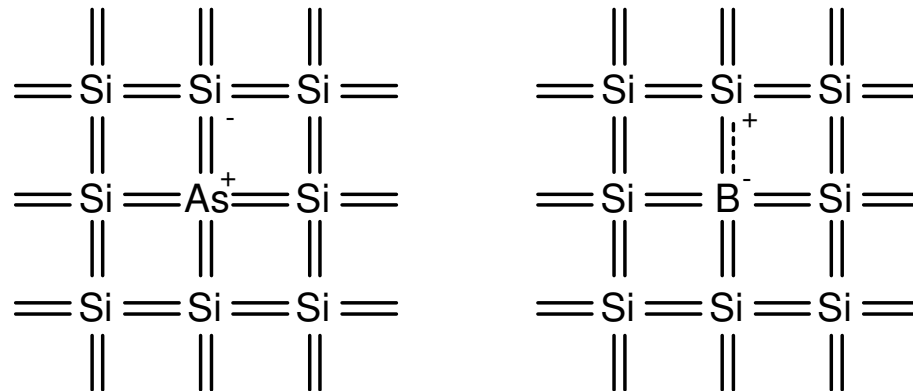
# Silicon Lattice

- Transistors are built on a silicon substrate
- Silicon is a Group IV material
- Forms crystal lattice with bonds to four neighbors



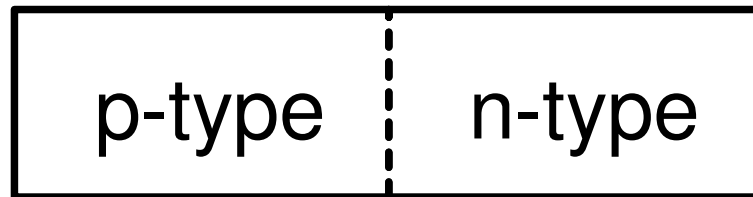
# Dopants

- Silicon is a semiconductor
- Pure silicon has no free carriers and conducts poorly
- Adding dopants increases the conductivity
- Group V: extra electron (n-type)
- Group III: missing electron, called hole (p-type)



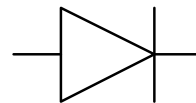
# p-n Junctions

- A junction between p-type and n-type semiconductor forms a diode.
- Current flows only in one direction



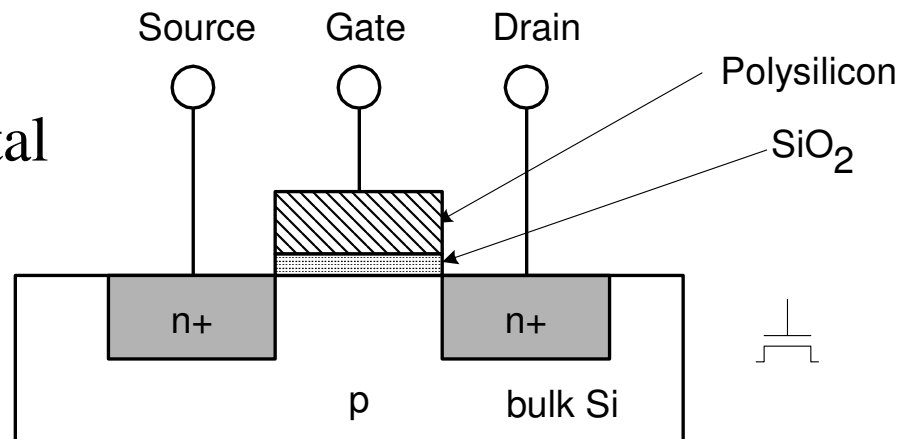
anode

cathode



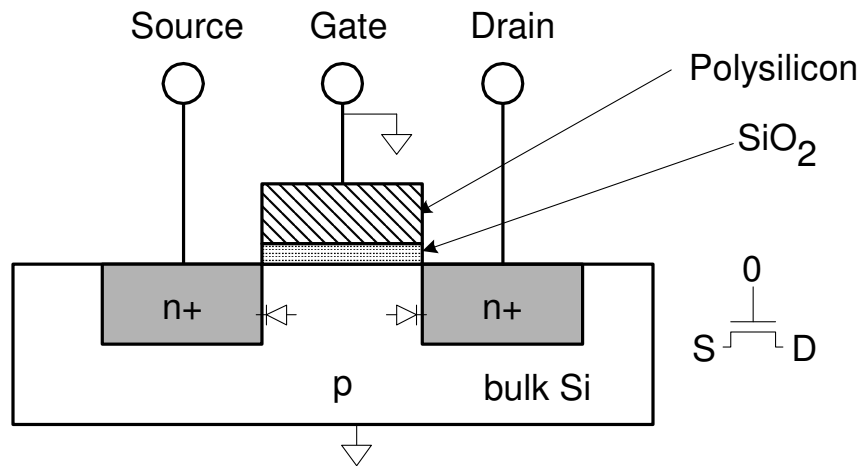
# nMOS Transistor

- Four terminals: gate, source, drain, body
- Gate – oxide – body stack looks like a capacitor
  - Gate and body are conductors
  - $\text{SiO}_2$  (oxide) is a very good insulator
  - Called metal – oxide – semiconductor (MOS) capacitor
  - Even though gate is no longer made of metal



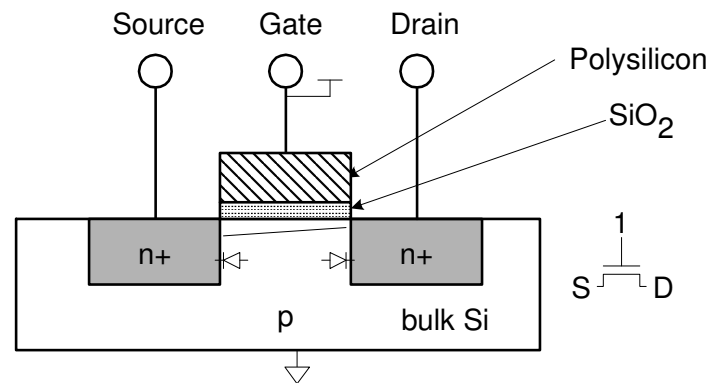
# nMOS Operation

- Body is commonly tied to ground (0 V)
- When the gate is at a low voltage:
  - P-type body is at low voltage
  - Source-body and drain-body diodes are OFF
  - No current flows, transistor is OFF



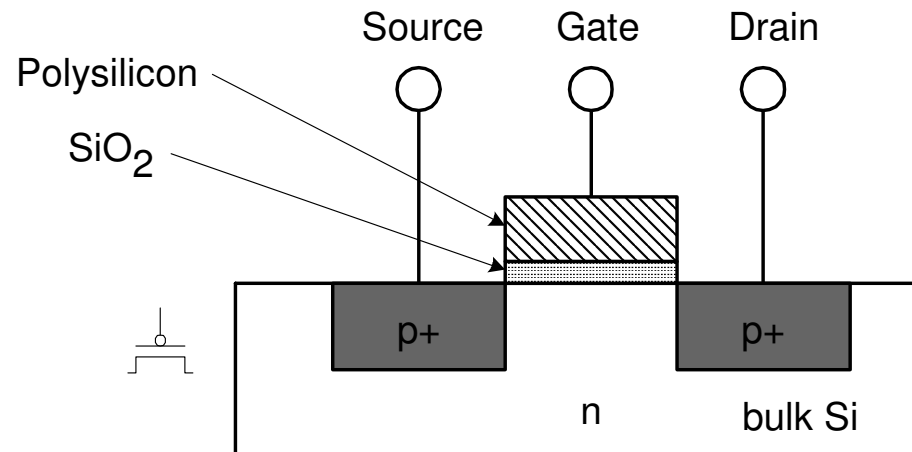
# nMOS Operation Cont.

- When the gate is at a high voltage:
  - Positive charge on gate of MOS capacitor
  - Negative charge attracted to body
  - Inverts a channel under gate to n-type
  - Now current can flow through n-type silicon from source through channel to drain, transistor is ON



# pMOS Transistor

- Similar, but doping and voltages reversed
  - Body tied to high voltage ( $V_{DD}$ )
  - Gate low: transistor ON
  - Gate high: transistor OFF
  - Bubble indicates inverted behavior

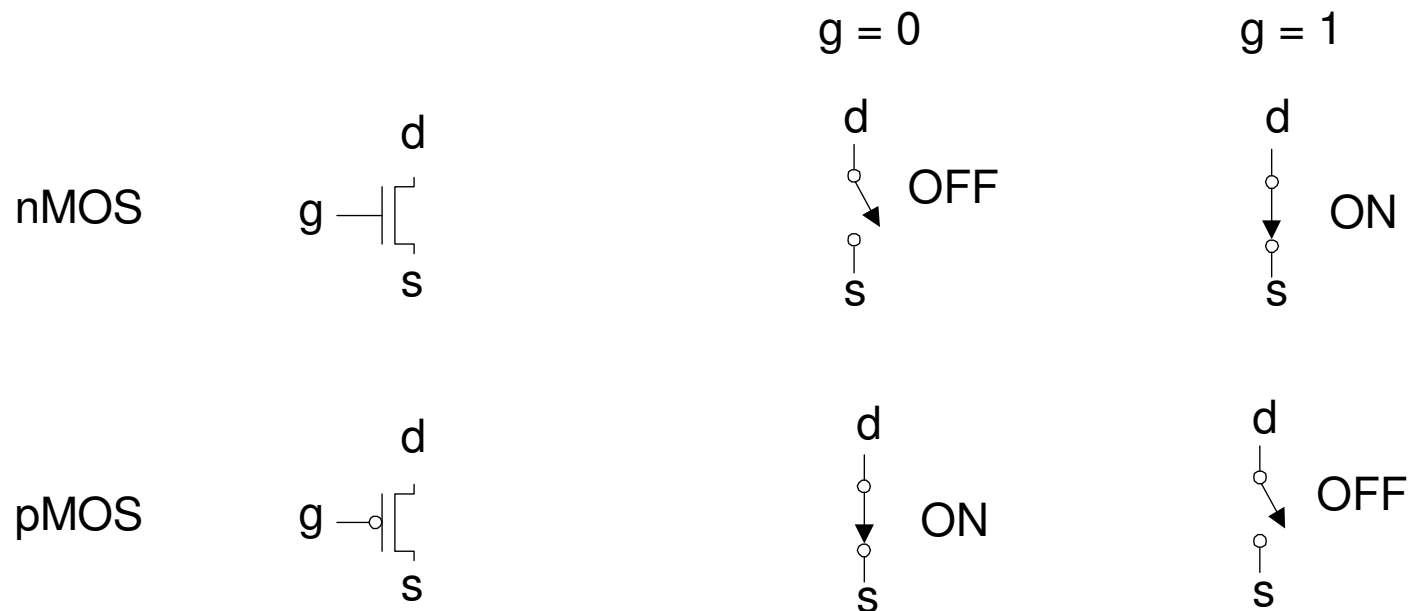


# Power Supply Voltage

- $\text{GND} = 0 \text{ V}$
- In 1980's,  $V_{\text{DD}} = 5\text{V}$
- $V_{\text{DD}}$  has decreased in modern processes
  - High  $V_{\text{DD}}$  would damage modern tiny transistors
  - Lower  $V_{\text{DD}}$  saves power
- $V_{\text{DD}} = 3.3, 2.5, 1.8, 1.5, 1.2, 1.0, \dots$

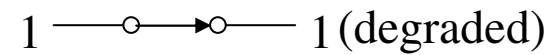
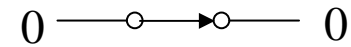
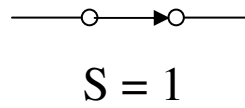
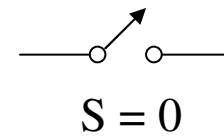
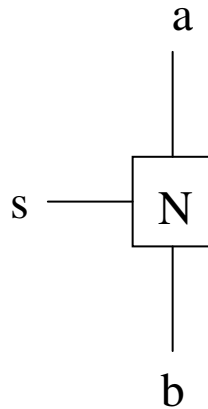
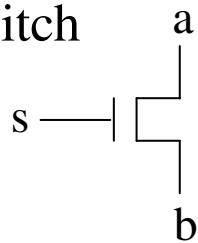
# Transistors as Switches

- We can view MOS transistors as electrically controlled switches
- Voltage at gate controls path from source to drain



# MOS Transistor Switches

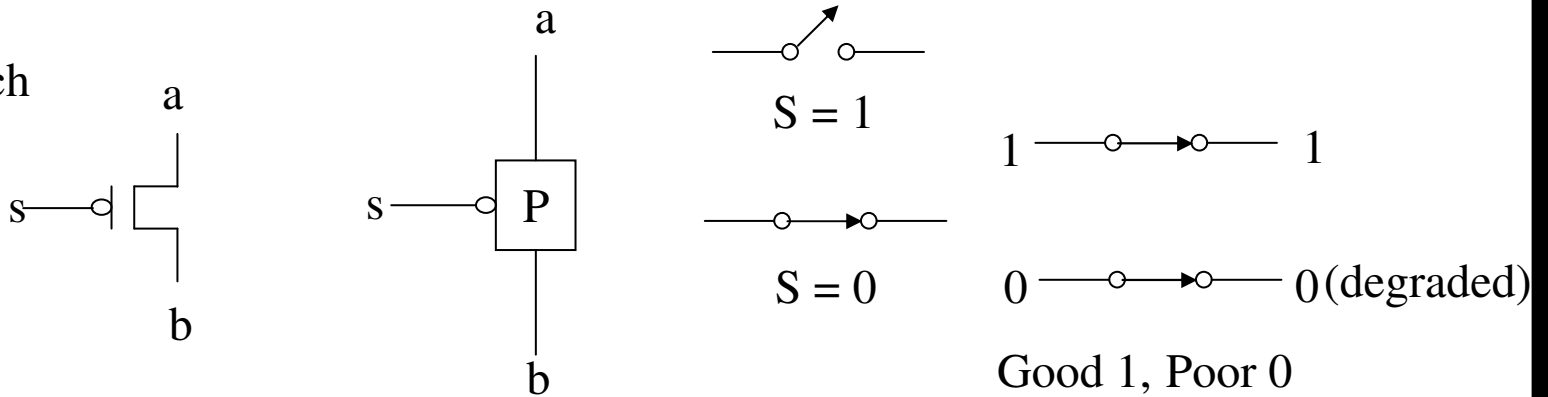
N-switch



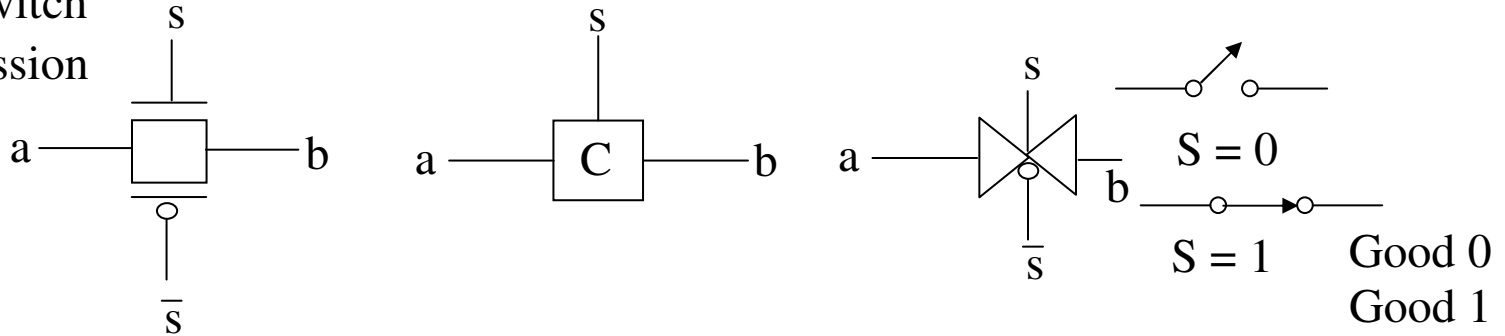
Good 0, Poor 1

# MOS Transistor Switches

P-switch



CMOS switch  
(Transmission gate)

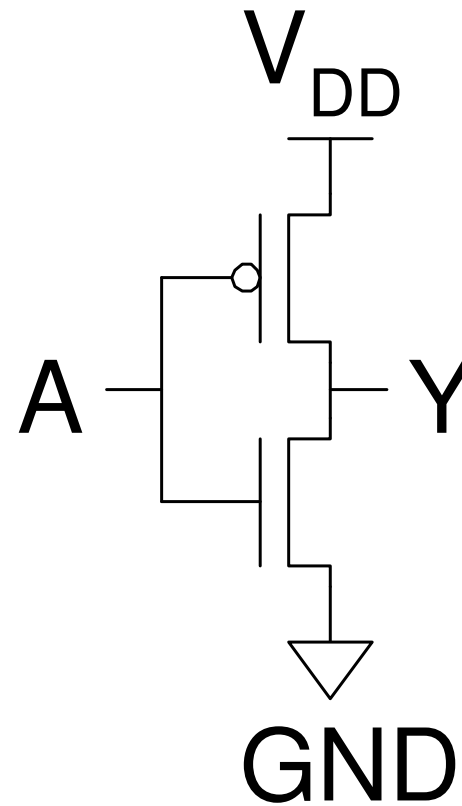
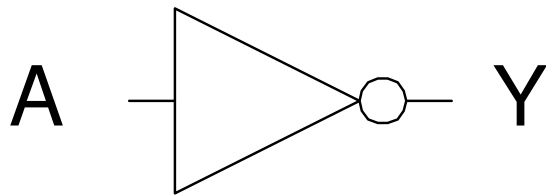


# Signal Strength

- *Strength* of signal
  - How close it approximates ideal voltage source
- $V_{DD}$  and GND rails are strongest 1 and 0
- nMOS pass strong 0
  - But degraded or weak 1
- pMOS pass strong 1
  - But degraded or weak 0
- Thus nMOS are best for pull-down network

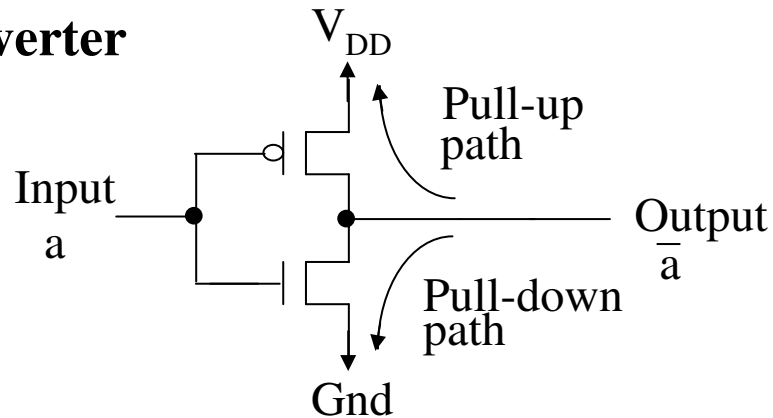
# CMOS Inverter

A	Y
0	
1	

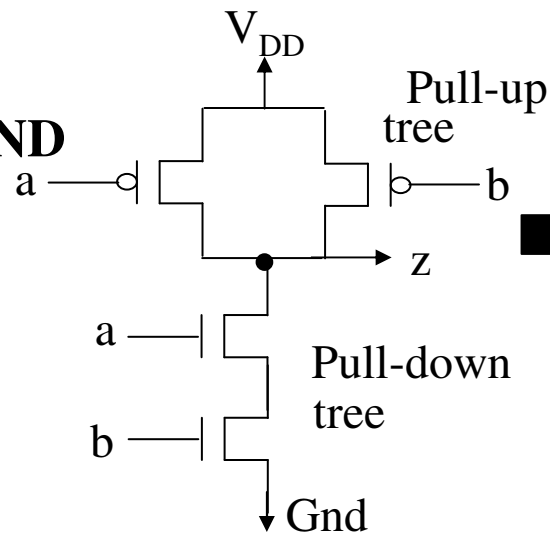


# CMOS Logic Gates-1

**Inverter**



**2-input NAND**



Pull-down truth table

a	b	z
0	0	Z
0	1	Z
1	0	Z
1	1	0

Pull-up truth table

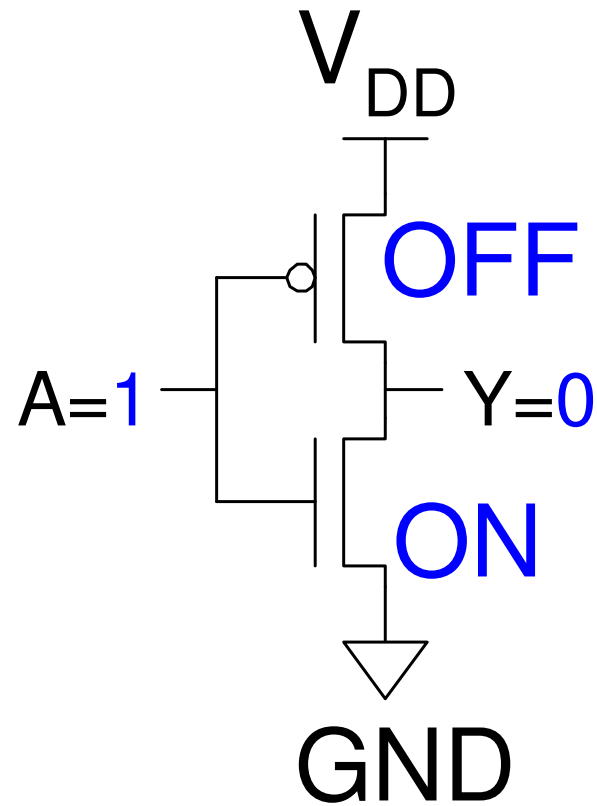
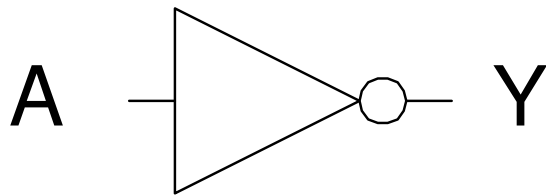
a	b	z
0	0	1
0	1	1
1	0	1
1	1	Z

NAND truth table

a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

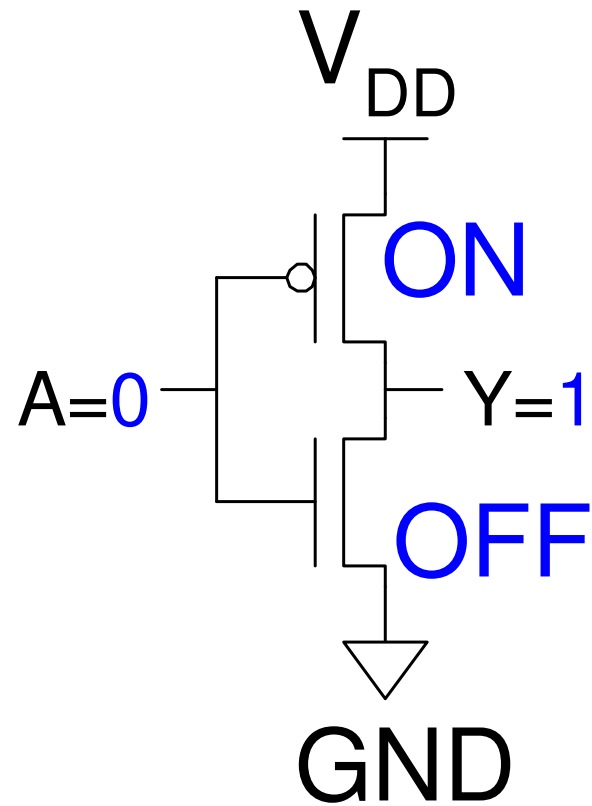
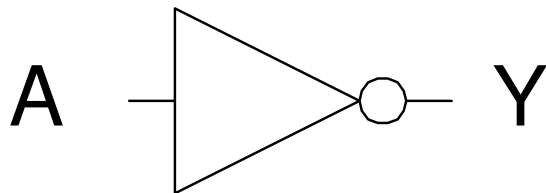
# CMOS Inverter

A	Y
0	
<b>1</b>	<b>0</b>



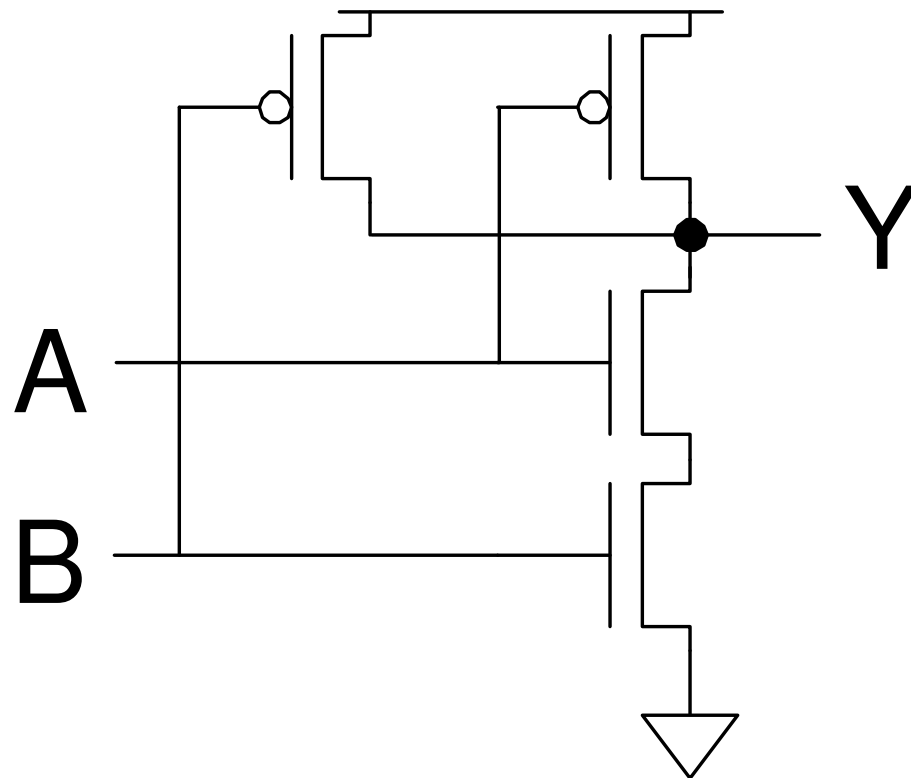
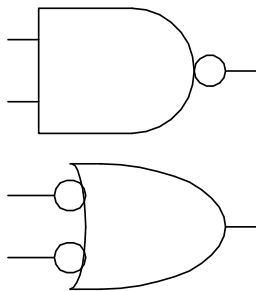
# CMOS Inverter

A	Y
0	1
1	0



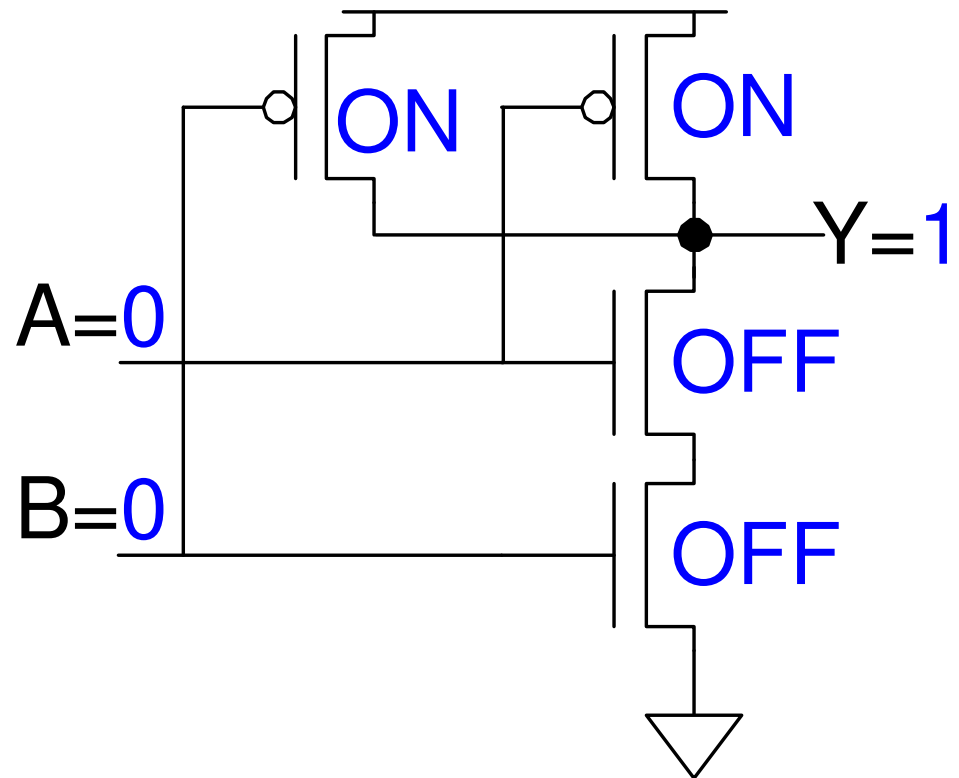
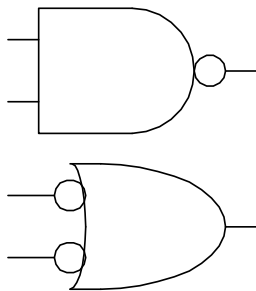
# CMOS NAND Gate

A	B	Y
0	0	
0	1	
1	0	
1	1	



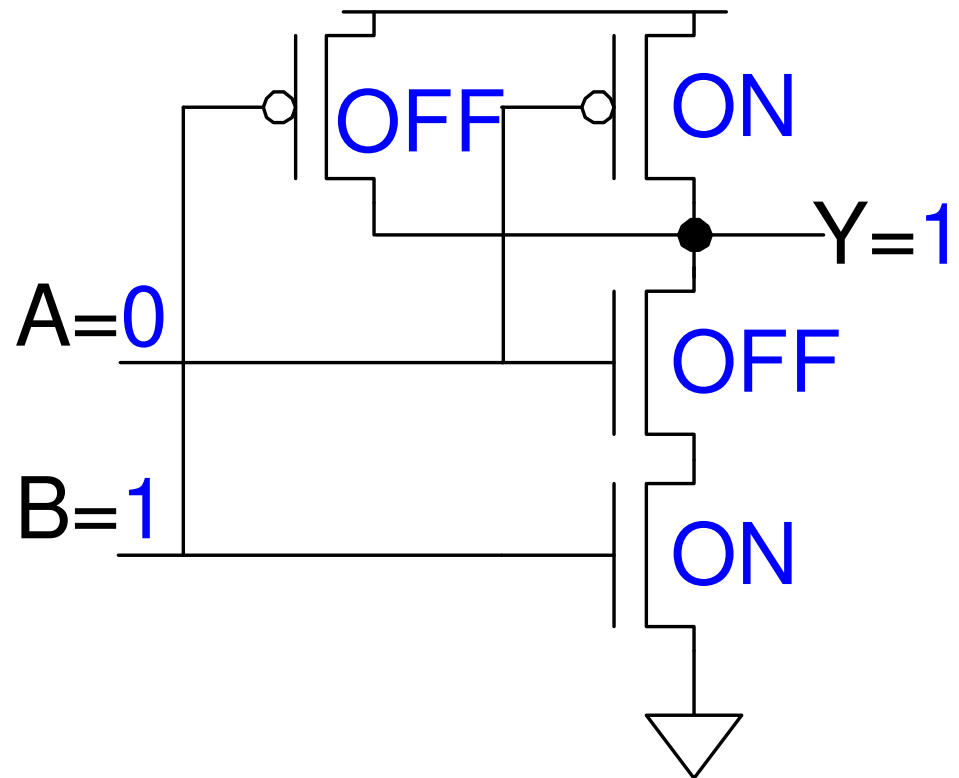
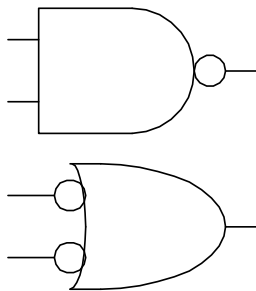
# CMOS NAND Gate

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



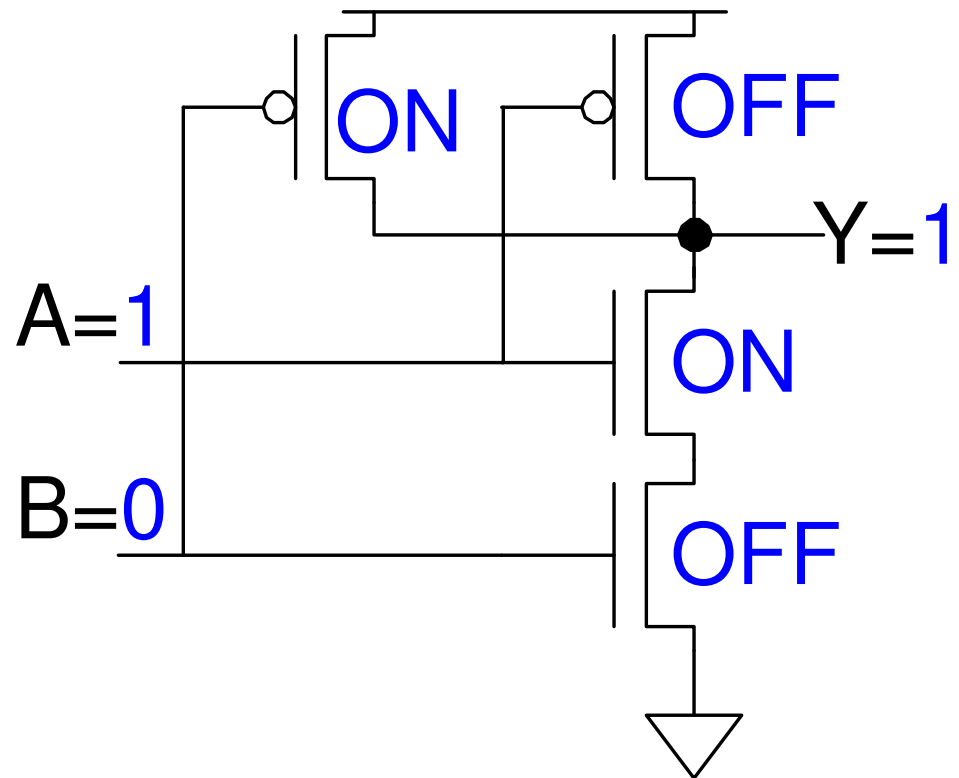
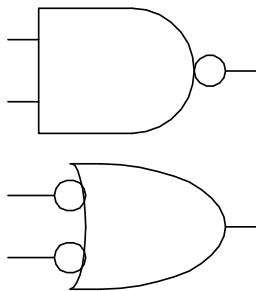
# CMOS NAND Gate

A	B	Y
0	0	1
<b>0</b>	<b>1</b>	<b>1</b>
1	0	
1	1	



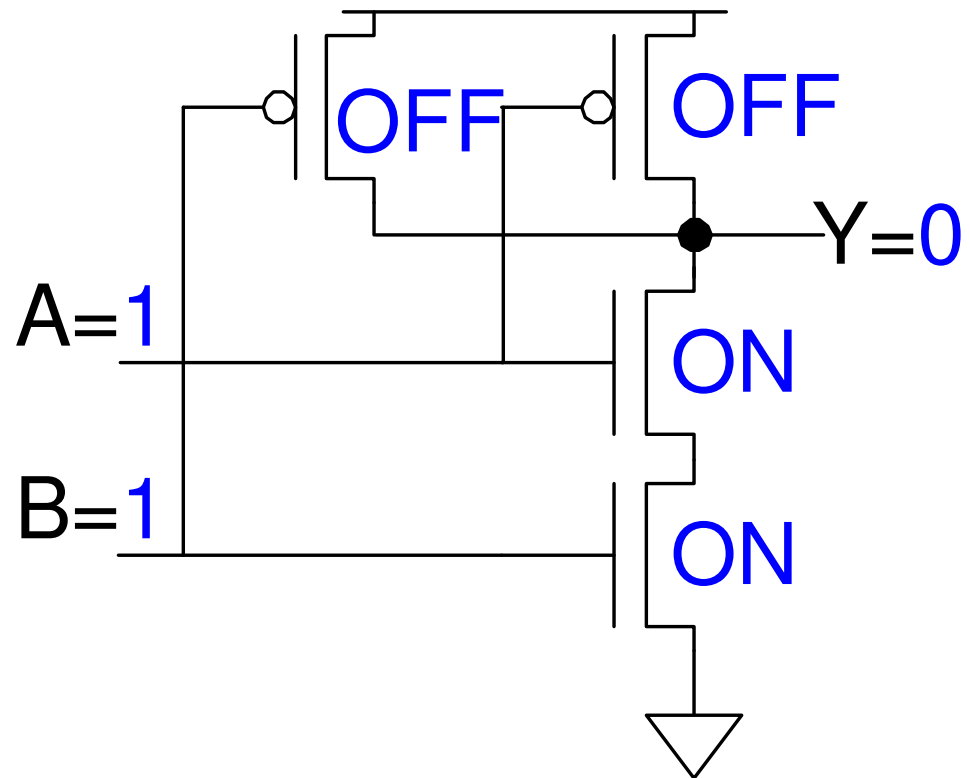
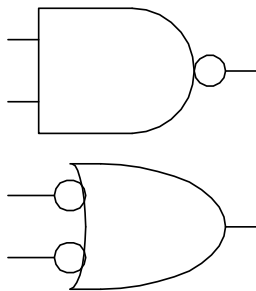
# CMOS NAND Gate

A	B	Y
0	0	1
0	1	1
<b>1</b>	<b>0</b>	<b>1</b>
1	1	



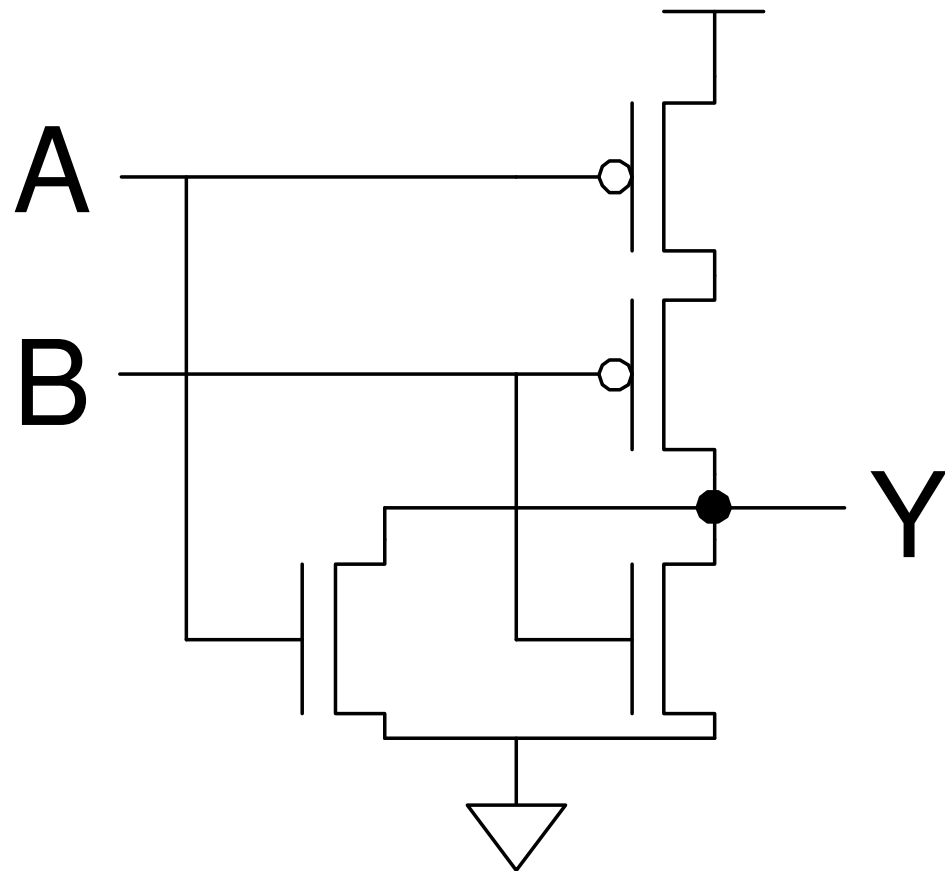
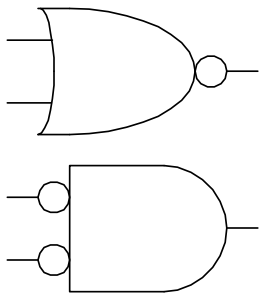
# CMOS NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
<b>1</b>	<b>1</b>	<b>0</b>



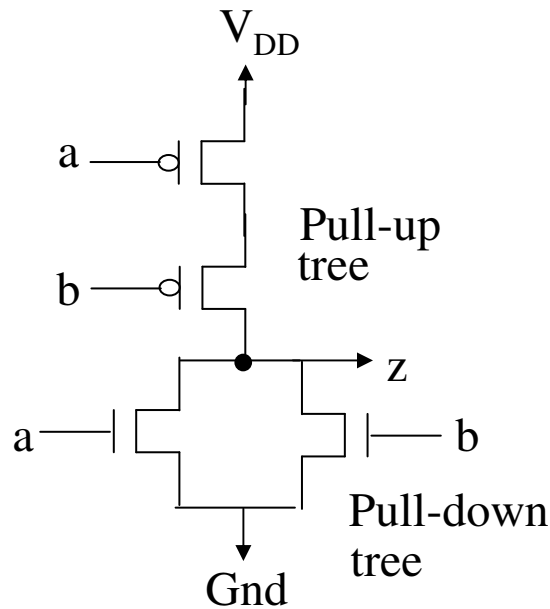
# CMOS NOR Gate

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



# CMOS Logic Gates-2

## 2-input NOR



Pull-down truth table		
a	b	z
0	0	Z
0	1	0
1	0	0
1	1	0

Pull-up truth table		
a	b	z
0	0	1
0	1	Z
1	0	Z
1	1	Z

NOR truth table		
a	b	z
0	0	1
0	1	0
1	0	0
1	1	0

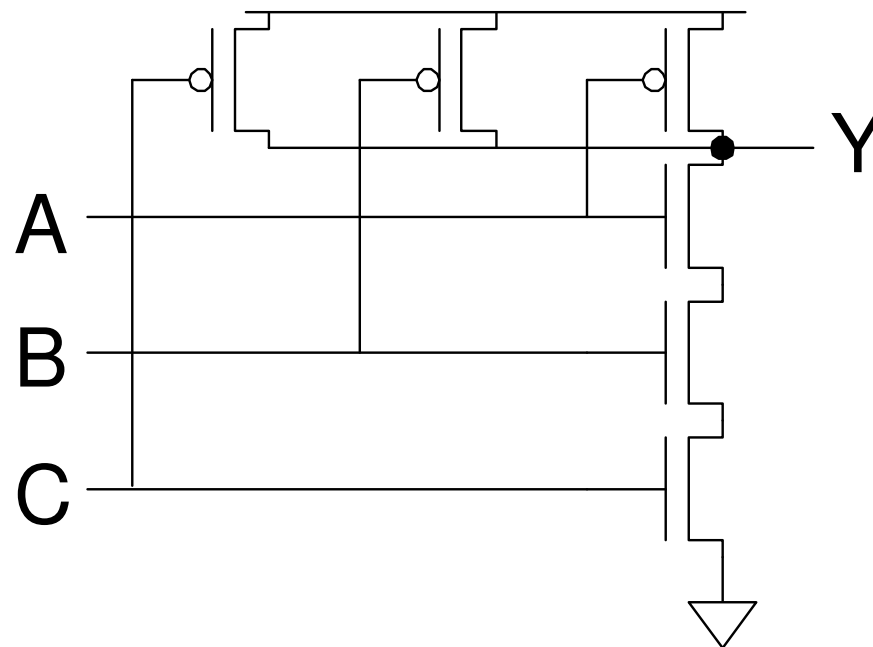
- There is always (for all input combinations) a path from *either* 1 or 0 to the output
- No direct path from 1 to 0 (low power dissipation)
- *Fully restored* logic
- No ratio-ing is necessary (*ratio-less* logic)
- Generalize to n-input NAND and n-input NOR?

# 3-input NAND Gate

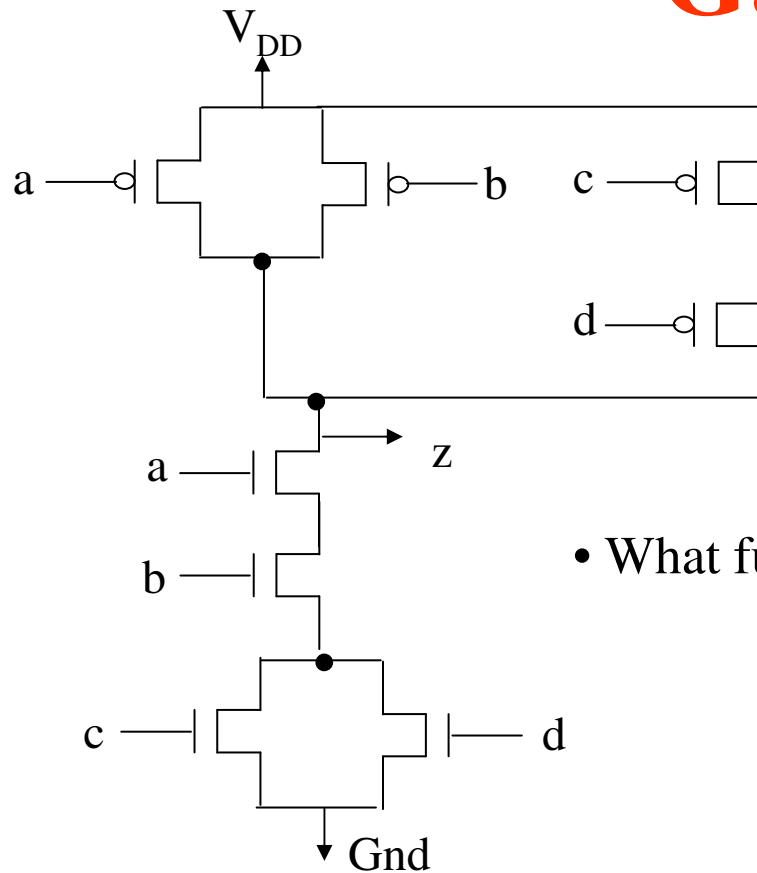
- Y pulls low if ALL inputs are 1
- Y pulls high if ANY input is 0

# 3-input NAND Gate

- Y pulls low if ALL inputs are 1
- Y pulls high if ANY input is 0



# CMOS Compound (Complex) Gates-1

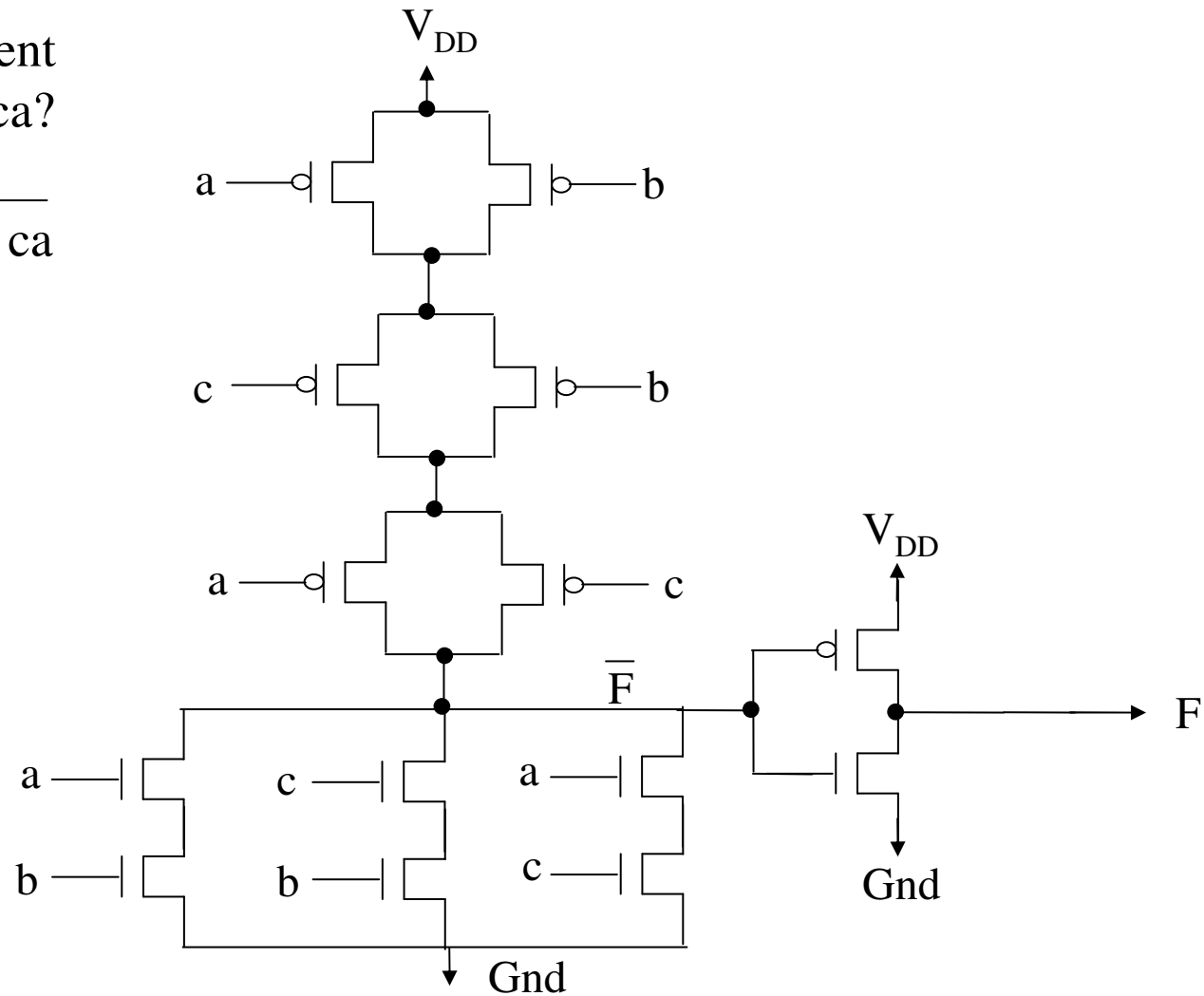


- What function is implemented by this circuit?

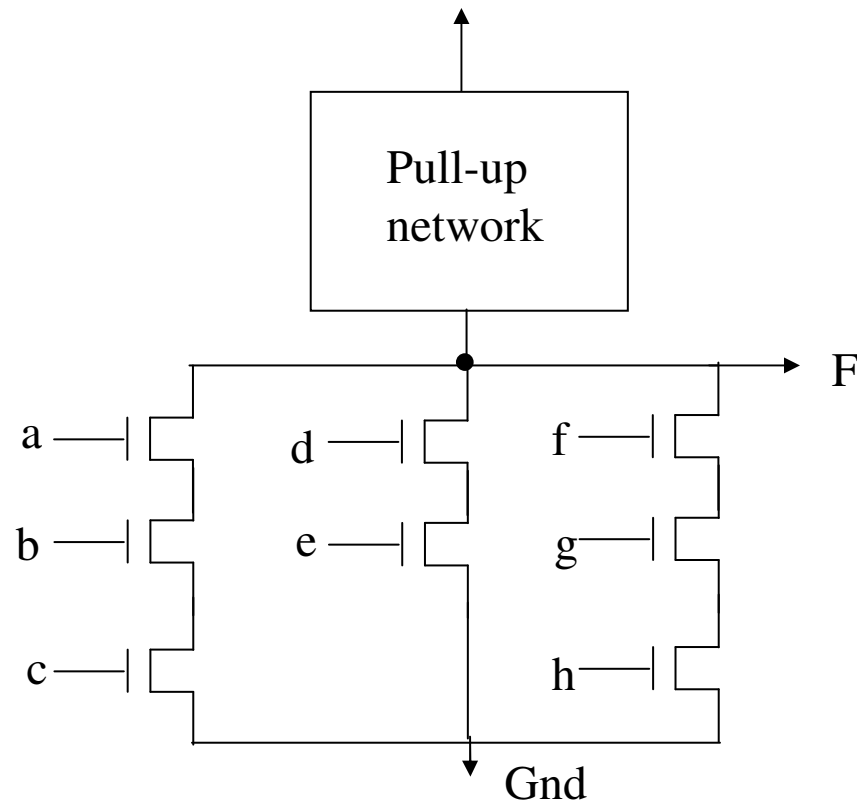
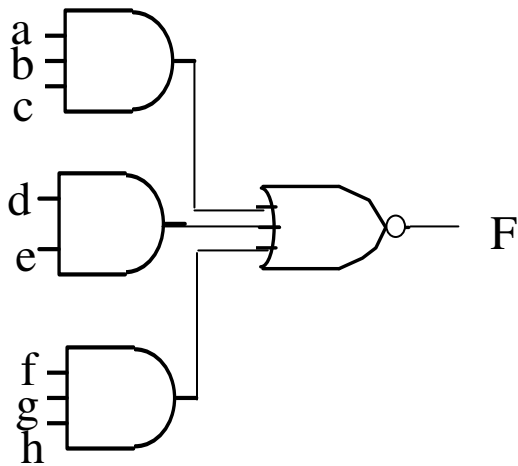
# Compound Gates-2

How to implement  
 $F = ab + bc + ca$ ?

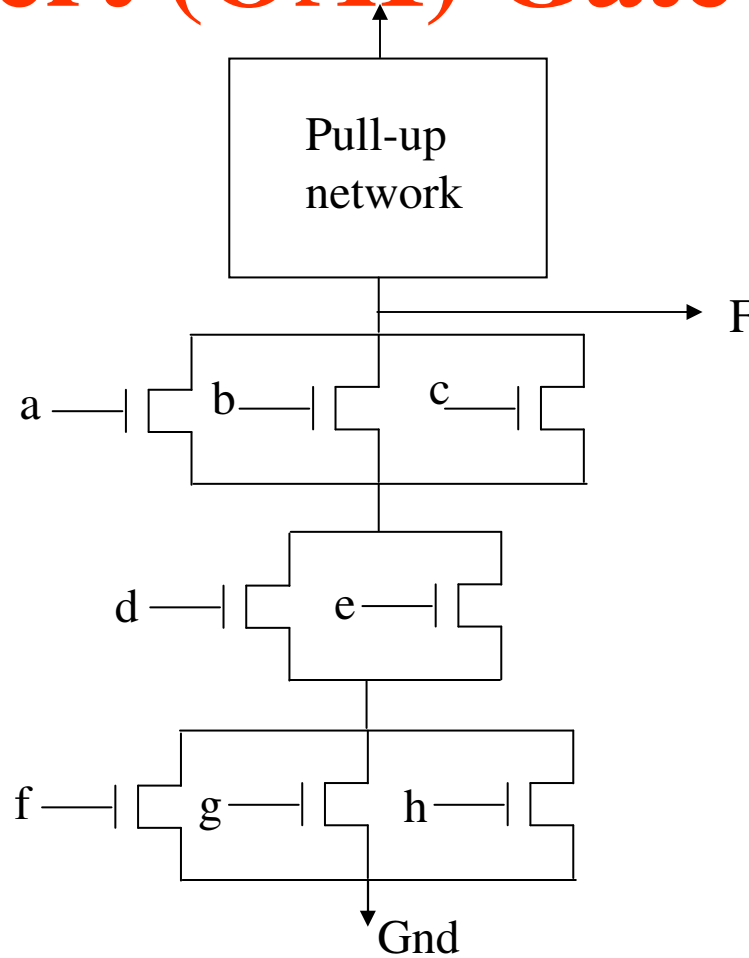
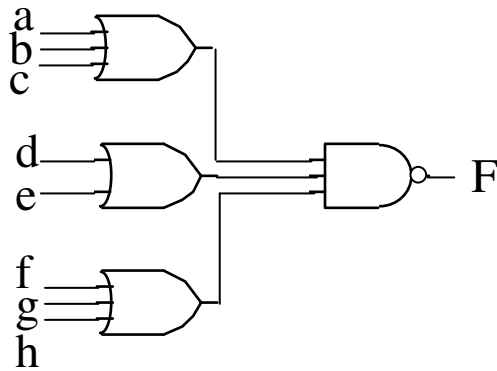
- $F = ab + bc + ca$



# And-Or-Invert (AOI) Gates



# Or-And-Invert (OAI) Gate



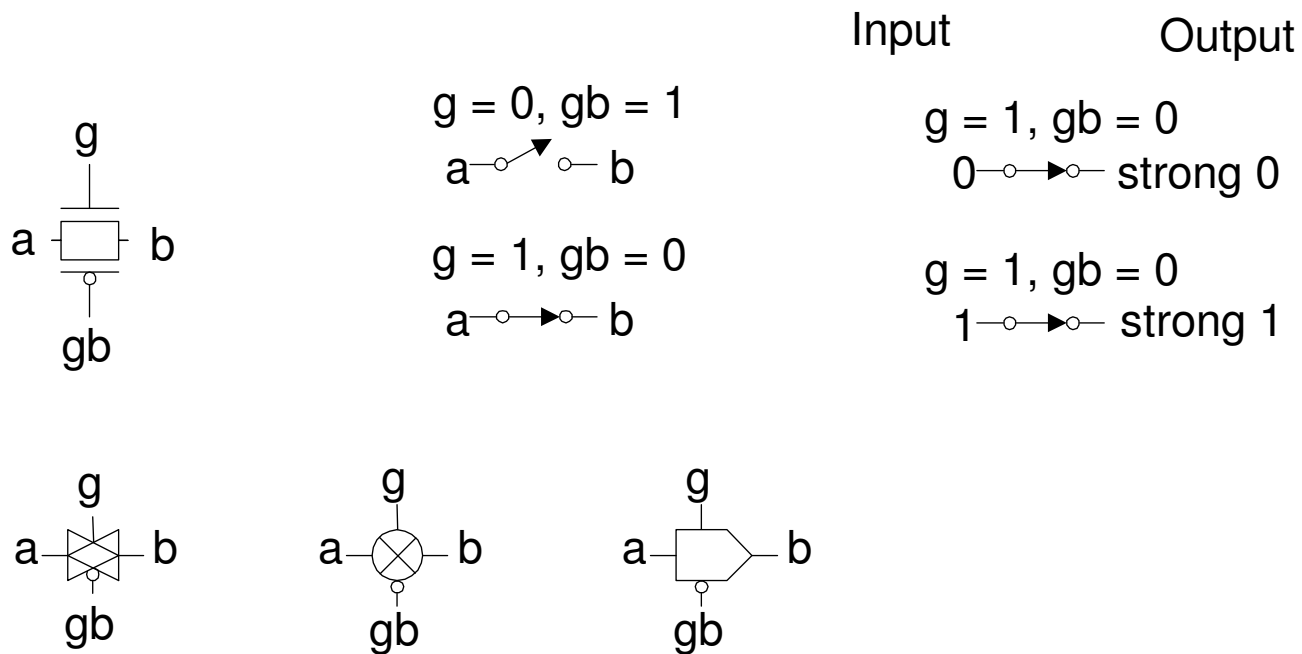
- Generally, complex CMOS gates can be derived directly from maxterms of the function (as in a Karnaugh map)

# Transmission Gates

- Pass transistors produce degraded outputs
- *Transmission gates* pass both 0 and 1 well

# Transmission Gates

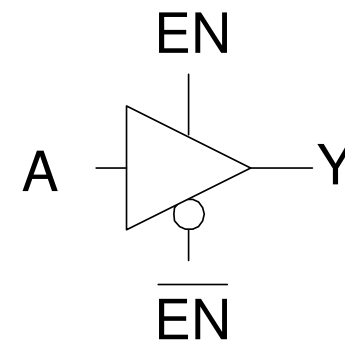
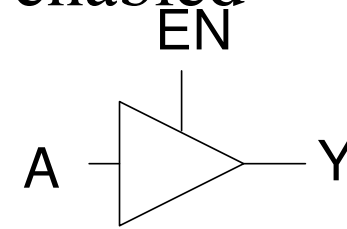
- Pass transistors produce degraded outputs
- *Transmission gates* pass both 0 and 1 well



# Tristates

- *Tristate buffer* produces Z when not enabled

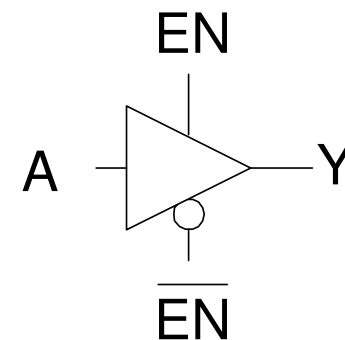
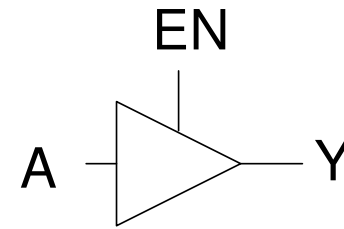
EN	A	Y
0	0	
0	1	
1	0	
1	1	



# Tristates

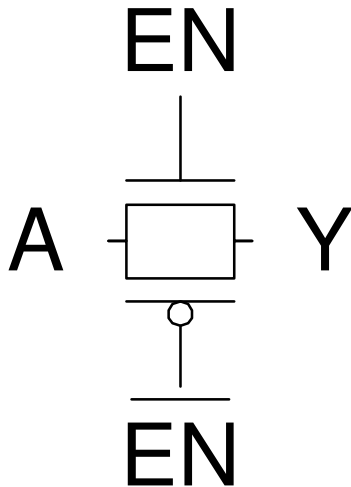
- *Tristate buffer* produces Z when not enabled

EN	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1



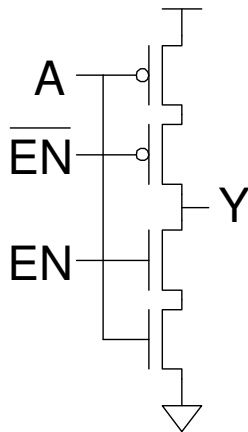
# Nonrestoring Tristate

- Transmission gate acts as tristate buffer
  - Only two transistors
  - But *nonrestoring*
    - Noise on A is passed on to Y



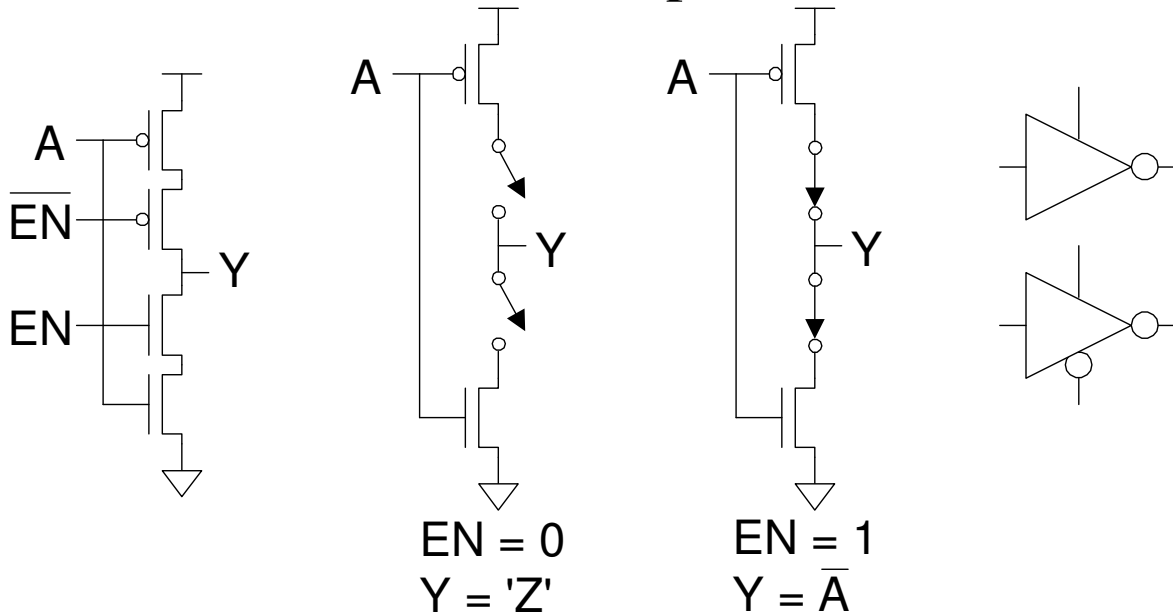
# Tristate Inverter

- Tristate inverter produces restored output
  - Violates conduction complement rule
  - Because we want a Z output



# Tristate Inverter

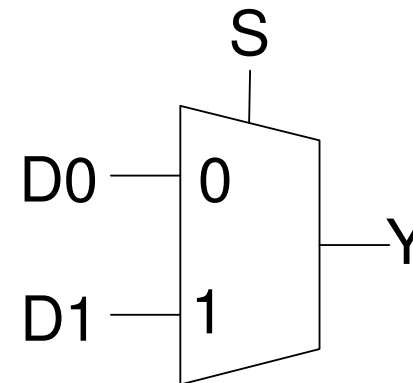
- Tristate inverter produces restored output
  - Violates conduction complement rule
  - Because we want a Z output



# Multiplexers

- *2:1 multiplexer* chooses between two inputs

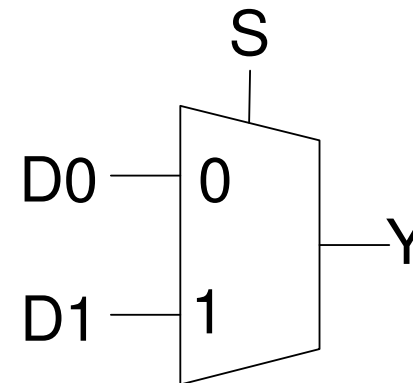
S	D1	D0	Y
0	X	0	
0	X	1	
1	0	X	
1	1	X	



# Multiplexers

- 2:1 multiplexer chooses between two inputs

S	D1	D0	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1



# Gate-Level Mux Design

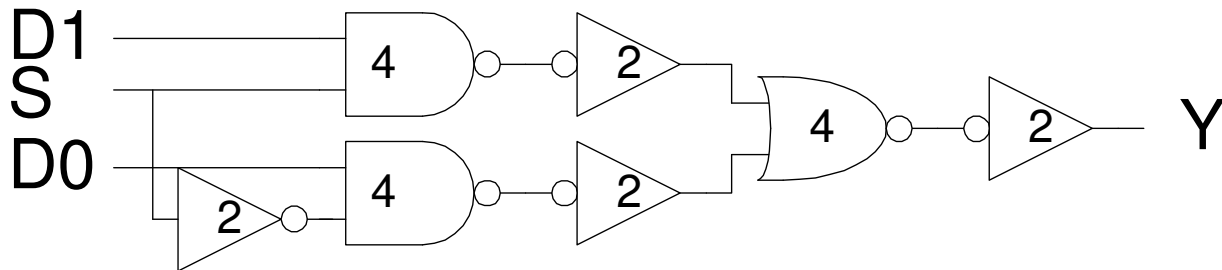
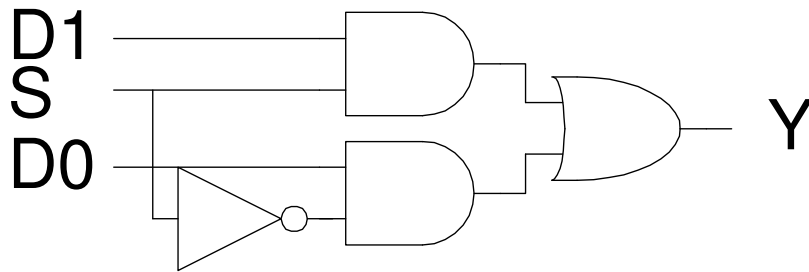
$$Y = SD_1 + \bar{S}D_0 \text{ (too many transistors)}$$

- How many transistors are needed?

# Gate-Level Mux Design

$$Y = SD_1 + \bar{S}D_0 \text{ (too many transistors)}$$

- How many transistors are needed? 20

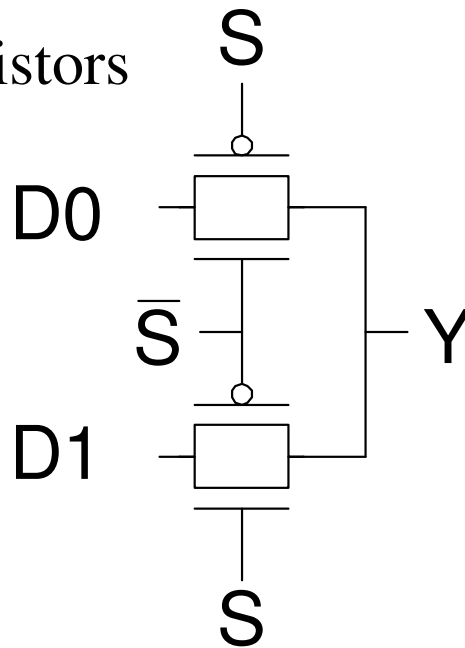


# Transmission Gate Mux

- Nonrestoring mux uses two transmission gates

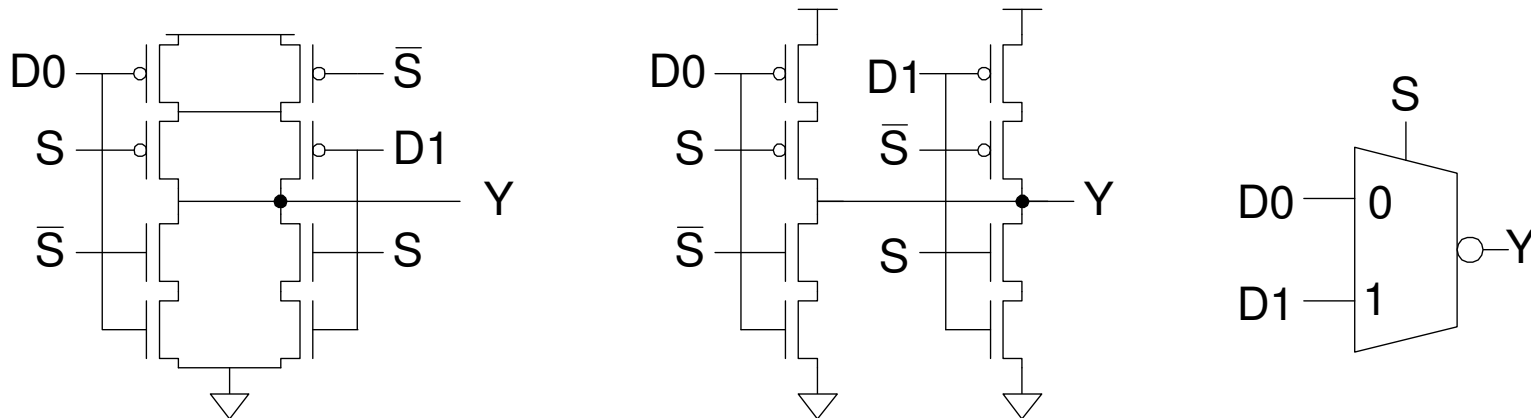
# Transmission Gate Mux

- Nonrestoring mux uses two transmission gates
  - Only 4 transistors



# Inverting Mux

- Inverting multiplexer
  - Use compound AOI22
  - Or pair of tristate inverters
  - Essentially the same thing
- Noninverting multiplexer adds an inverter

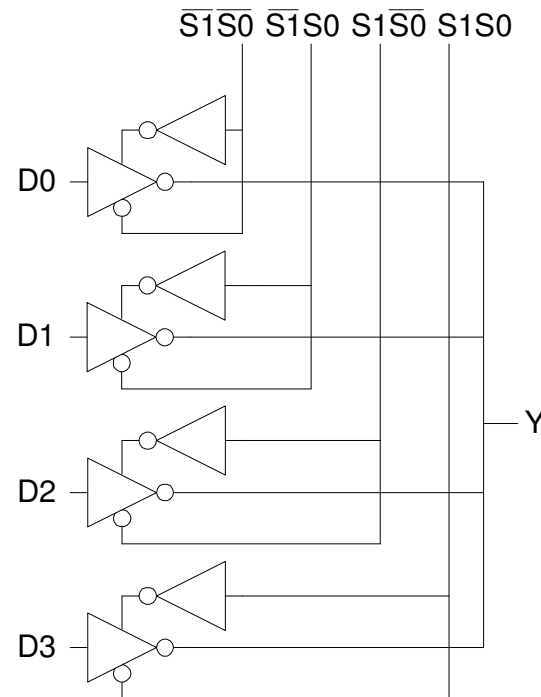
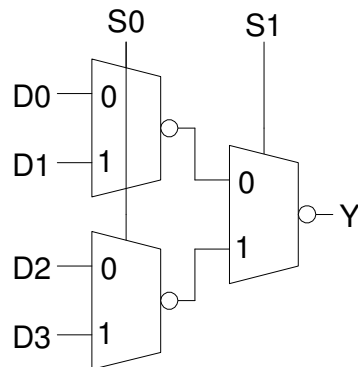


# 4:1 Multiplexer

- 4:1 mux chooses one of 4 inputs using two selects

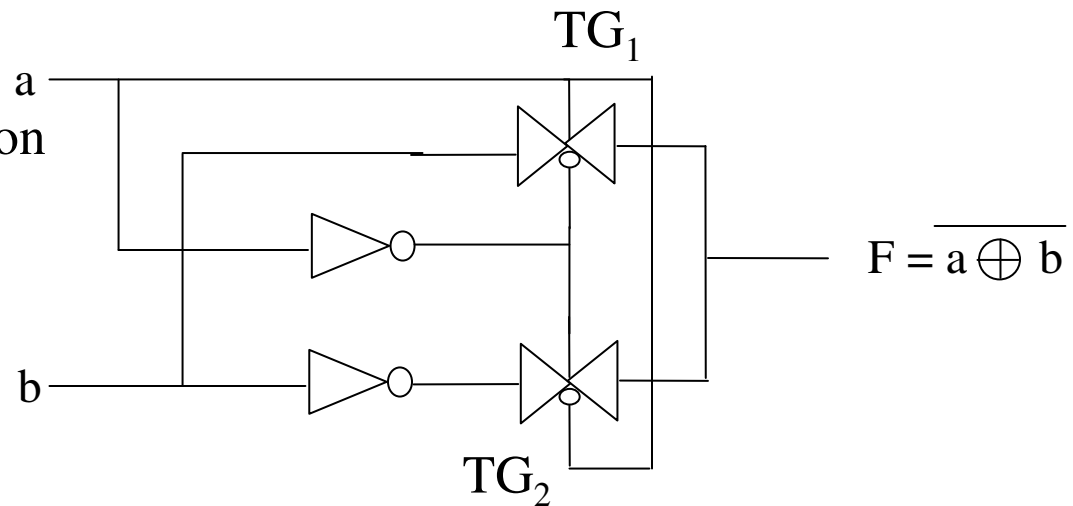
# 4:1 Multiplexer

- 4:1 mux chooses one of 4 inputs using two selects
  - Two levels of 2:1 muxes
  - Or four tristates



# CMOS Exclusive-Nor Gate

- 8-transistor implementation

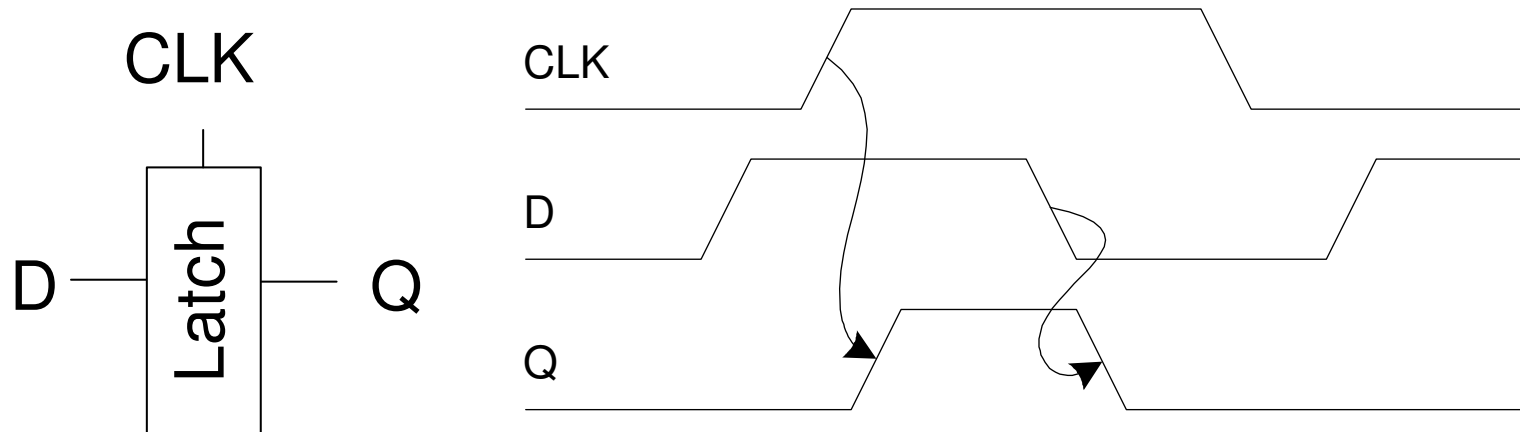


a	b	TG <sub>1</sub>	TG <sub>2</sub>	F
0	0	nonconducting	conducting	$\overline{B}$ (1)
0	1	nonconducting	conducting	$\overline{B}$ (0)
1	0	conducting	nonconducting	B (0)
1	1	conducting	nonconducting	B (1)

- Better, 6-transistor implementation is possible!

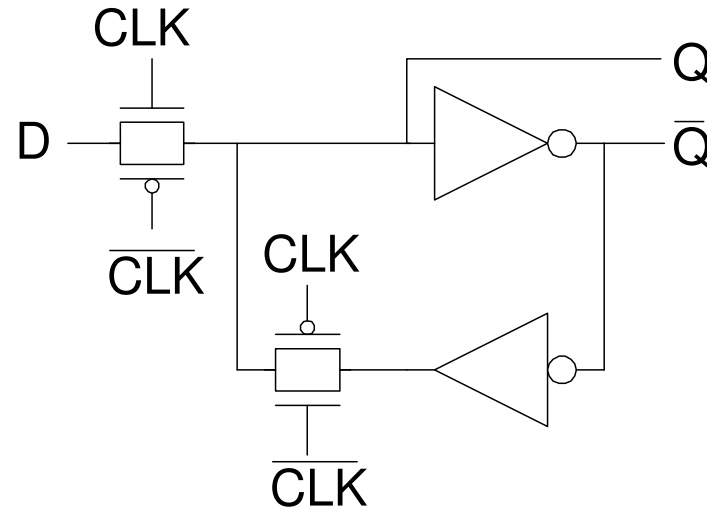
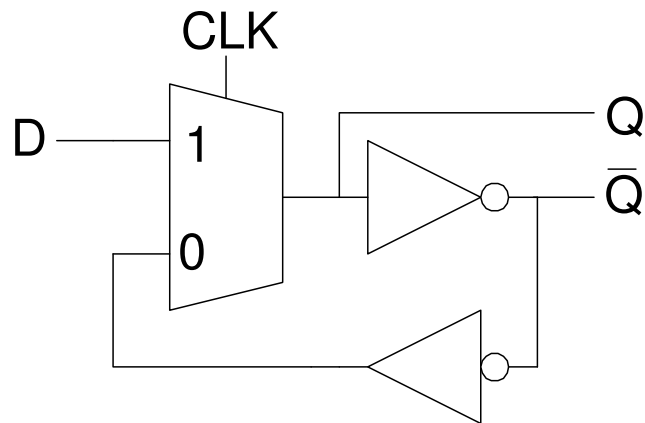
# D Latch

- When  $CLK = 1$ , latch is *transparent*
  - D flows through to Q like a buffer
- When  $CLK = 0$ , the latch is *opaque*
  - Q holds its old value independent of D
- a.k.a. *transparent latch* or *level-sensitive latch*

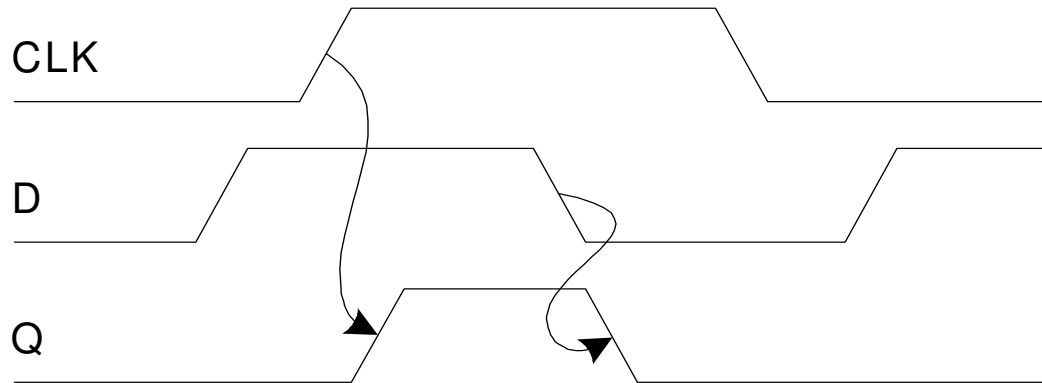
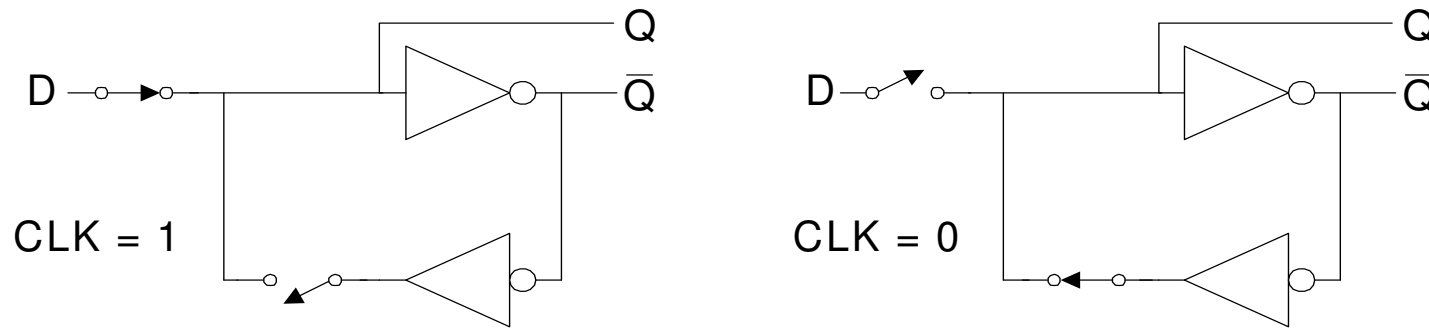


# D Latch Design

- Multiplexer chooses D or old Q

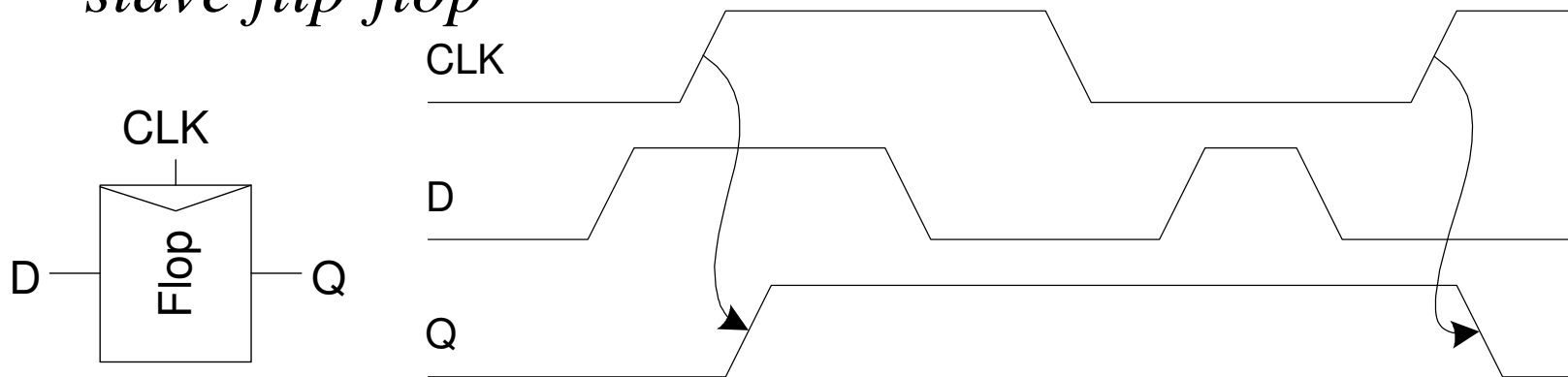


# D Latch Operation



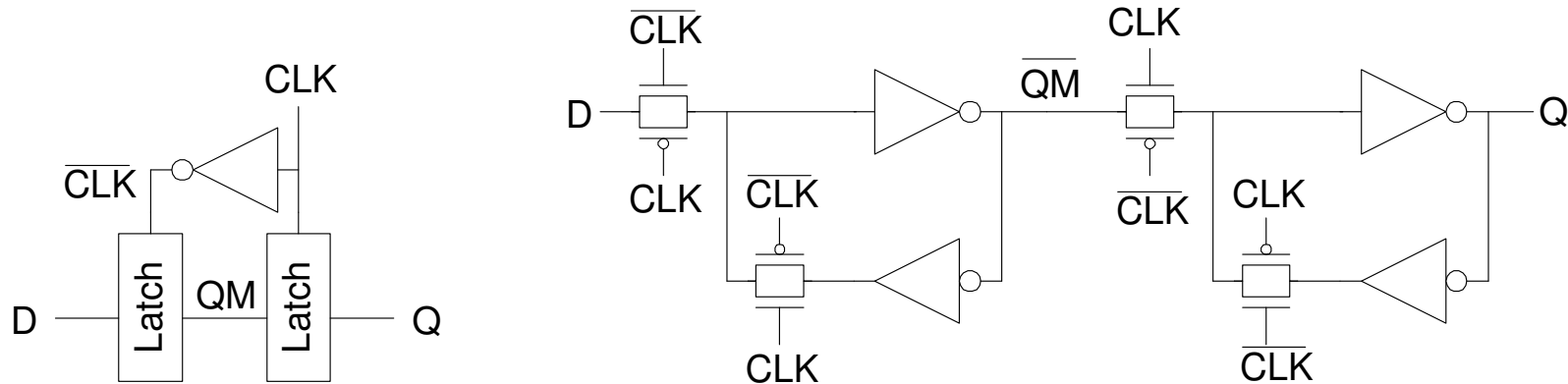
# D Flip-flop

- When CLK rises, D is copied to Q
- At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop, master-slave flip-flop*

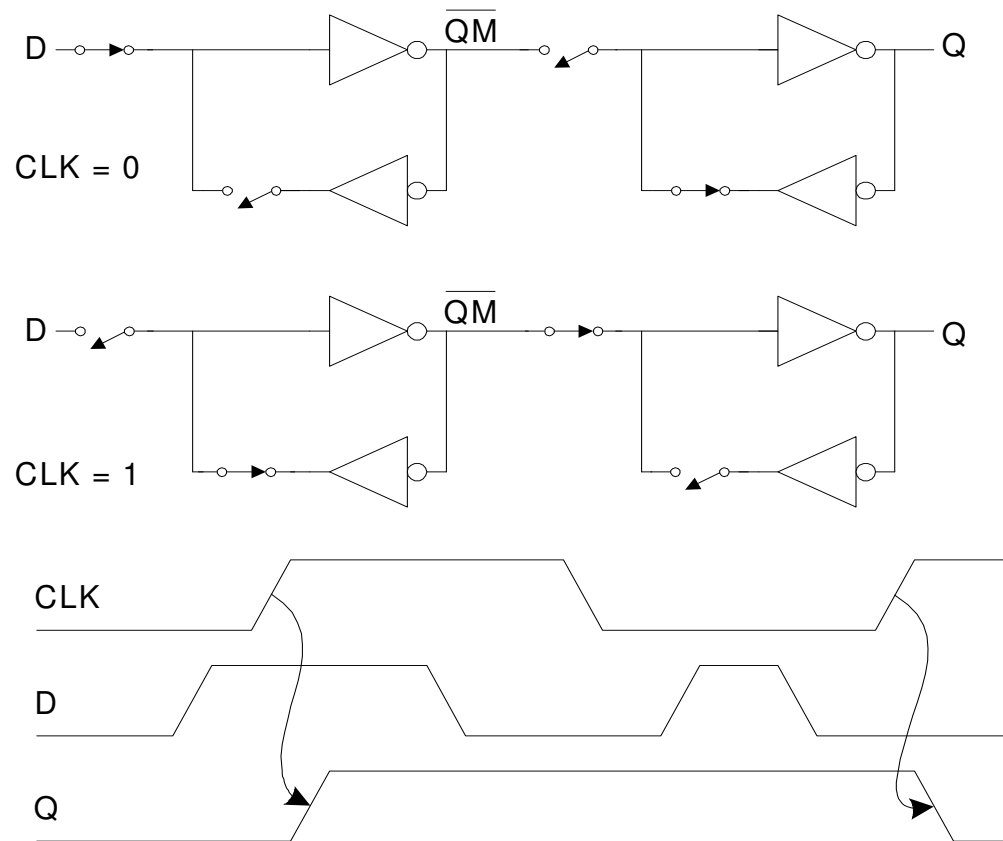


# D Flip-flop Design

- Built from master and slave D latches

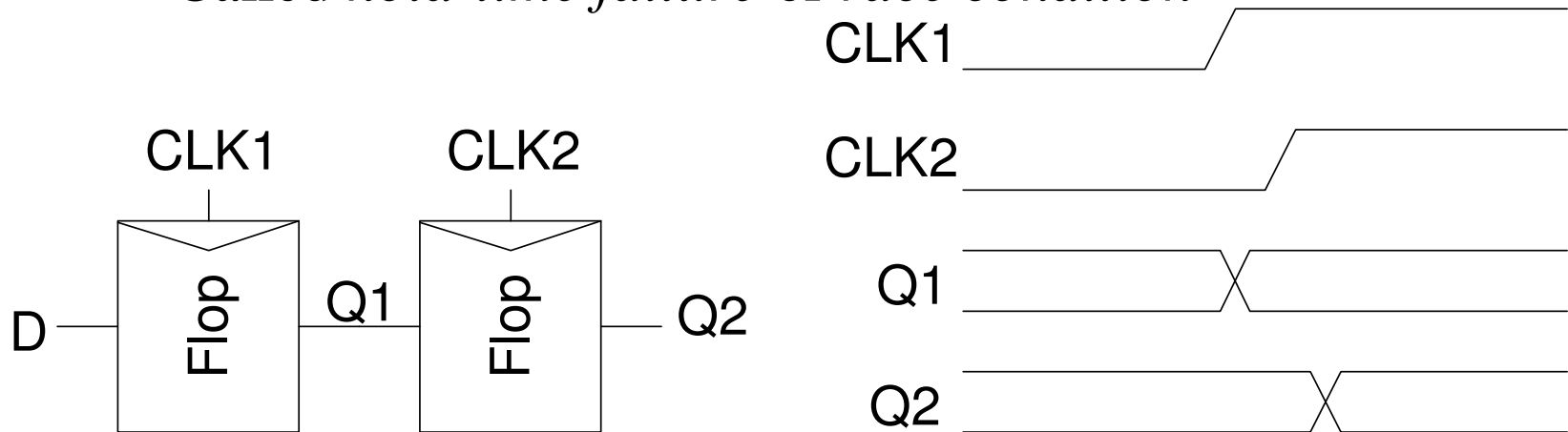


# D Flip-flop Operation



# Race Condition

- Back-to-back flops can malfunction from clock skew
  - Second flip-flop fires late
  - Sees first flip-flop change and captures its result
  - Called *hold-time failure* or *race condition*



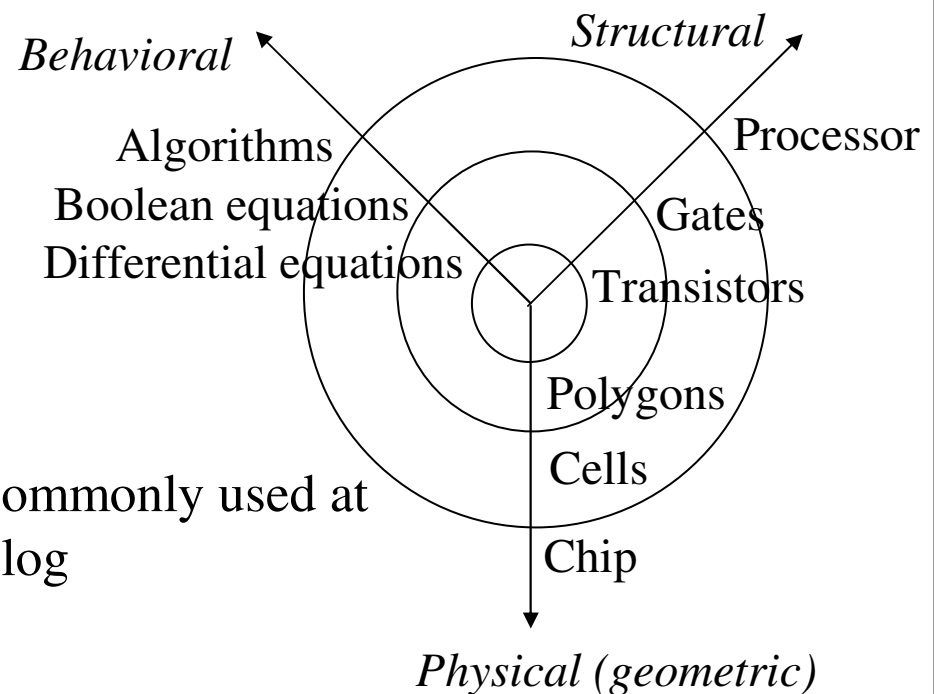


# Design Representation Levels

- Design domains

- Behavioral
- Structural
- Physical

Gajski and Kuhn's Y-chart  
(layered like an onion)



- Hardware description languages commonly used at behavioral level, e.g. VHDL, Verilog

- Example: Consider the carry function  $c_o = ab + bc + c_i a$

# Verilog Example (Behavioral)

Boolean equation form:

```
module carry (co, a, b, ci);  
output co;  
input a, b, ci;  
assign  
    co = (a & b) | (a & ci) | (b & ci);  
end module
```

Timing information:

```
module carry (co, a, b, ci);  
output co;  
input a, b, ci;  
Wire #10 co = (a & b) | (a & ci) | (b & ci);  
end module
```

c<sub>o</sub> changes 10 time units after a, b, or c changes

Boolean truth table form:

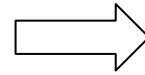
```
primitive carry (co, a, b, ci);  
output co;  
input a, b, ci;  
table  
    // a  b  c  co  
    1  1  ?  : 1;  
    1  ?  1  : 1;  
    ?  1  1  : 1;  
    0  0  ?  : 0;  
    0  ?  0  : 0;  
    ?  0  0  : 0;  
end table  
end module
```

# Verilog Example (Structural)

Structural representation of 4-bit adder (top-down) :

```
module add4 (s, c4, ci, a, b);  
  output [3:0] s;  
  output [3:0] c4;  
  input [3:0] a, b;  
  input ci;  
  wire [2:0] co;  
  add a0 (co[0], s[0], a[0], b[0], ci);  
  add a1 (co[1], ..., b[1], co[0]);  
  add a2 (co[2], ..., co[1]);  
  add a3 (c4, s[3], a[3], b[3], co[2]);  
end module
```

3-bit  
internal  
signal



```
module add (co, s, a, b, ci);  
  output s, co;  
  input a, b, ci;  
  sum s1 (s, a, b, ci);  
  carry c1 (co, a, b, ci);  
end module
```



```
module carry (co, a, b, ci);  
  output co;  
  input a, b, ci;  
  wire x, y, z;  
  and g1 (y, z, b);  
  and g2 (z, b, ci);  
  and g3 (z, a, ci);  
  or g4 (co, x, y, z);  
end module
```

*Technology-independent*