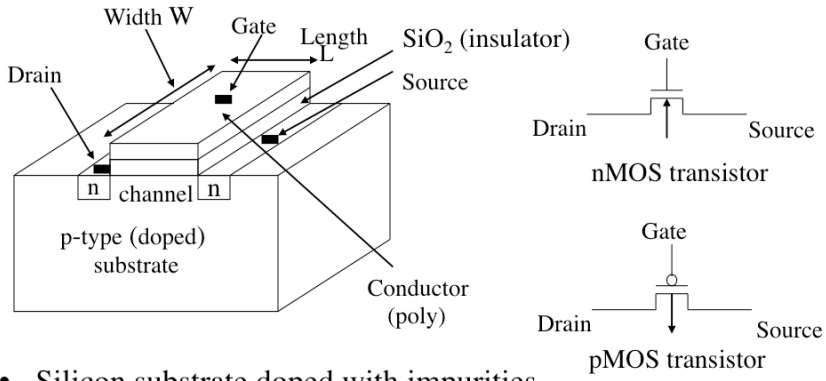
MOS Transistors



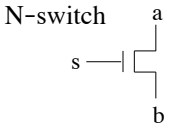
nMOS transistor

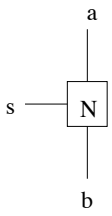
pMOS transistor

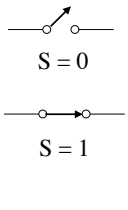
- Silicon substrate doped with impurities
- Adding or cutting away insulating glass (SiO_2)
- Adding wires made of polycrystalline silicon (*polysilicon, poly*) or metal, insulated from the substrate by SiO_2

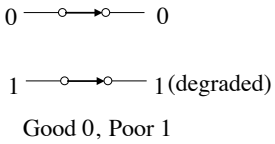
1

MOS Transistor Switches





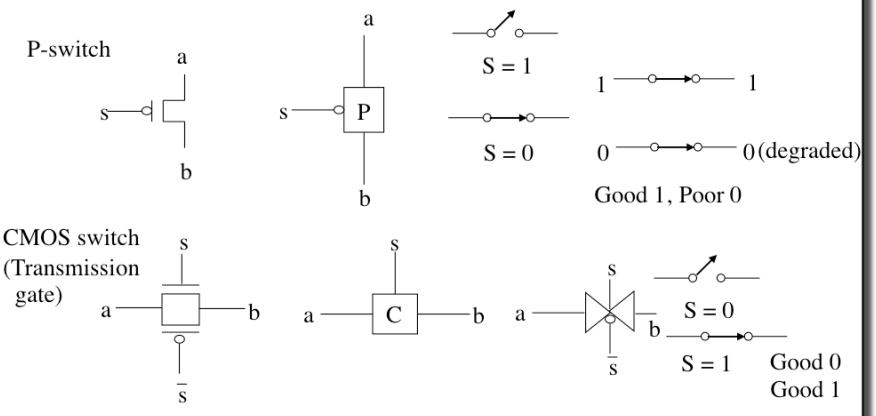




Good 0, Poor 1

2

MOS Transistor Switches



3

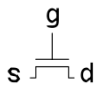
Signal Strength

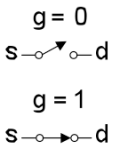
- *Strength* of signal
  - How close it approximates ideal voltage source
- V_{DD} and GND rails are strongest 1 and 0
- nMOS pass strong 0
  - But degraded or weak 1
- pMOS pass strong 1
  - But degraded or weak 0
- Thus nMOS are best for pull-down network

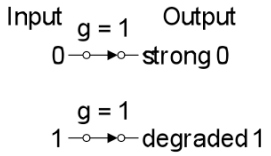
4

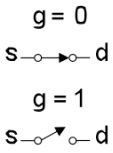
Pass Transistors

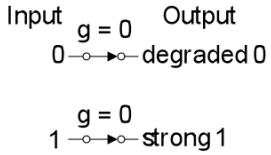
- Transistors can be used as switches







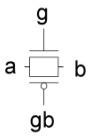


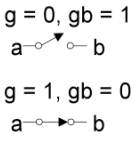


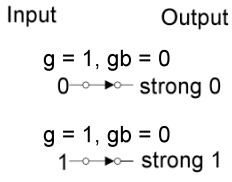
5

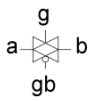
Transmission Gates

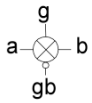
- Pass transistors produce degraded outputs
- *Transmission gates* pass both 0 and 1 well

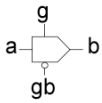








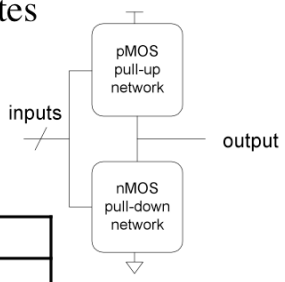




6

Complementary CMOS

- Complementary CMOS logic gates
  - nMOS *pull-down network*
  - pMOS *pull-up network*
  - a.k.a. static CMOS

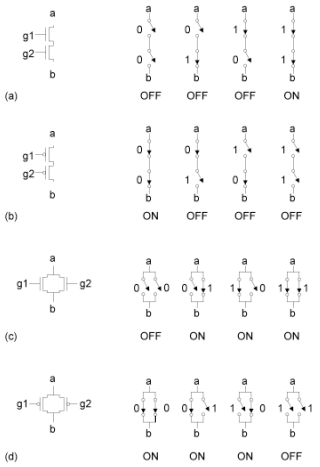


	Pull-up OFF	Pull-up ON
Pull-down OFF	Z (float)	1
Pull-down ON	0	X (not allowed)

7

Series and Parallel

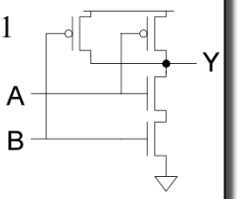
- nMOS: 1 = ON
- pMOS: 0 = ON
- *Series*: both must be ON
- *Parallel*: either can be ON



8

Conduction Complement

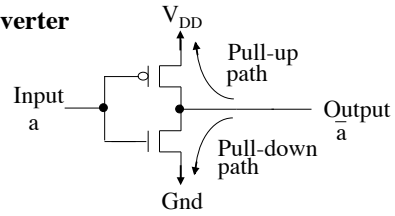
- Complementary CMOS gates always produce 0 or 1
- Ex: NAND gate
  - Series nMOS: Y=0 when both inputs are 1
  - Thus Y=1 when either input is 0
  - Requires parallel pMOS



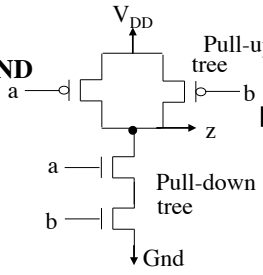
- Rule of *Conduction Complements*
  - Pull-up network is complement of pull-down
  - Parallel -> series, series -> parallel

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

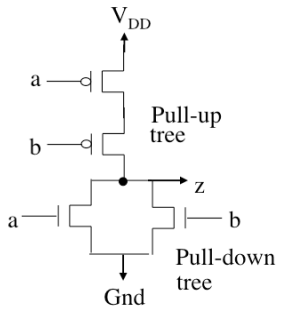
NAND truth table

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

10

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

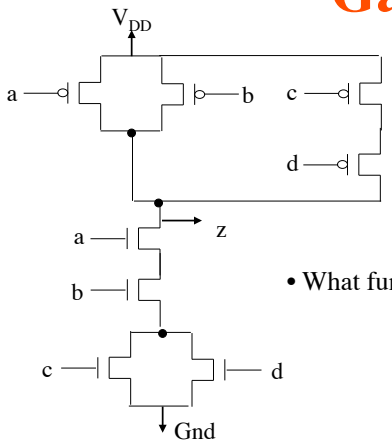
NOR truth table

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?

11

CMOS Compound (Complex) Gates-1



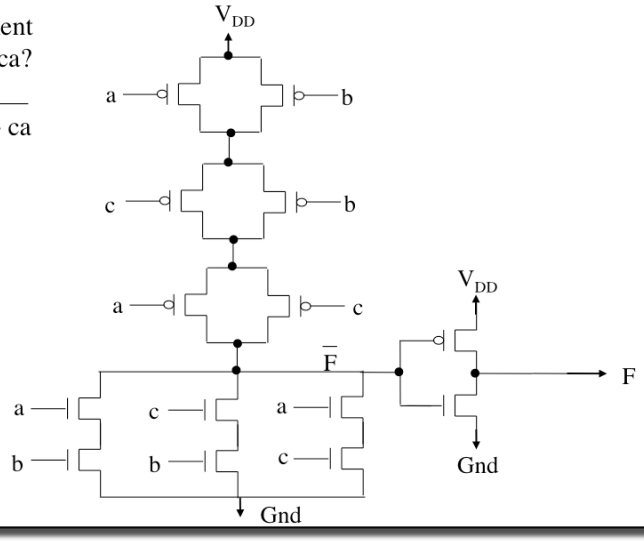
- What function is implemented by this circuit?

12

Compound Gates-2

How to implement
F = ab + bc + ca?

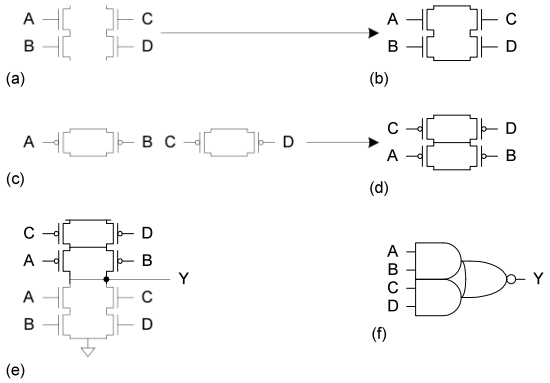
$\overline{F} = \overline{ab + bc + ca}$



13

Compound Gates

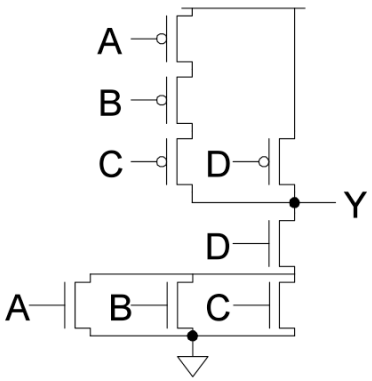
- *Compound gates* can do any inverting function
- Ex: $Y = \overline{A \bullet B + C \bullet D}$ (AND-AND-OR-INVERT, AOI22)



14

Example: O3AI

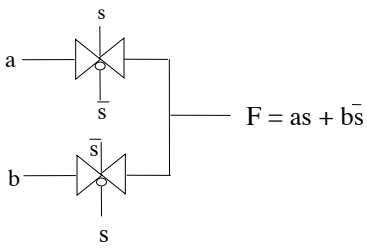
$$Y = \overline{(A + B + C) \bullet D}$$



15

CMOS Multiplexers

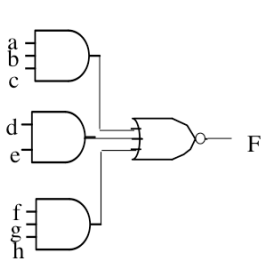
- Transmission gate implementation (4 transistors)
- Assume $\bar{s}$ is available

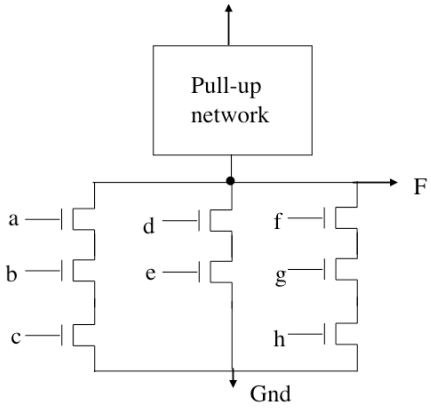


- Complex gate implementation based on $\bar{F} = \overline{as + b\bar{s}}$ requires 10 transistors

16

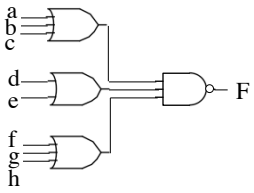
And-Or-Invert (AOI) Gates

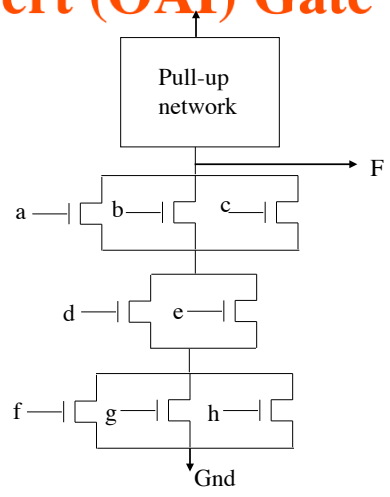




17

Or-And-Invert (OAI) Gate





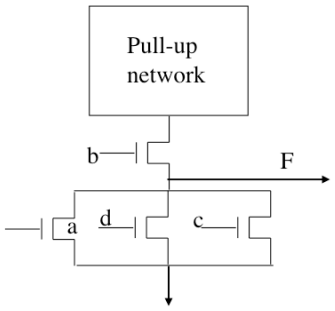
- Generally, complex CMOS gates can be derived directly from maxterms of the function (as in a Karnaugh map)

18

Designing Complex Gates Using K-Maps

ab \ cd	00	01	11	10
00	1	1	1	1
01	1	0	0	0
11	0	0	0	0
10	1	1	1	1

$\overline{F} = ab + bd + bc$
$= b(a+d+c)$



$F = \bar{b} + \bar{a}\bar{c}\bar{d}$ (how many transistors?)

19

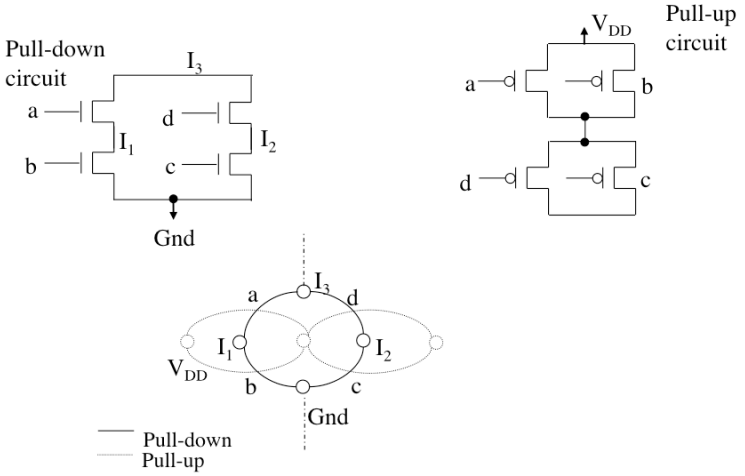
Graph Models

How to generate pull-up circuit from the pull-down circuit?

- Draw pull-down graph for pull-down circuit
  - Every vertex is a source-drain connection
  - Every edge is an nMOS transistor
- Generate pull-up graph from the pull-down graph
  - Add vertex for every "region" of pull-down graph
  - Add edge between between vertices lying in adjacent "regions"
  - Pull-up circuit corresponds to pull-up graph

20

Graph Models

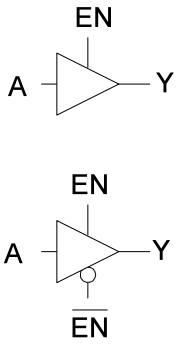


21

Tristates

- *Tristate buffer* produces Z when not enabled

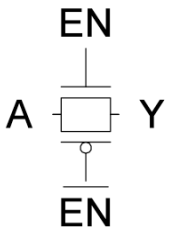
EN	A	Y
0	0	
0	1	
1	0	
1	1	



22

Nonrestoring Tristate

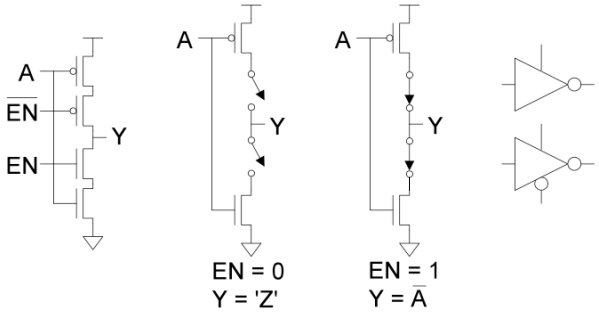
- Transmission gate acts as tristate buffer
  - Only two transistors
  - But *nonrestoring*
    - Noise on A is passed on to Y



23

Tristate Inverter

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

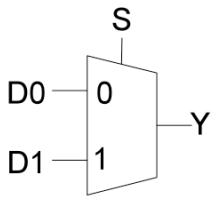


24

Multiplexers

- 2:1 multiplexer chooses between two inputs

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

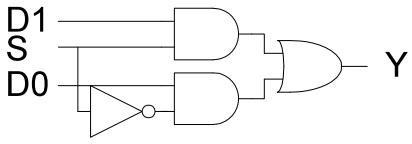


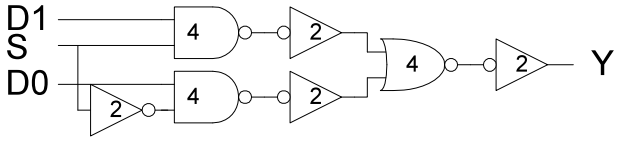
25

Gate-Level Mux Design

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

- How many transistors are needed? 20

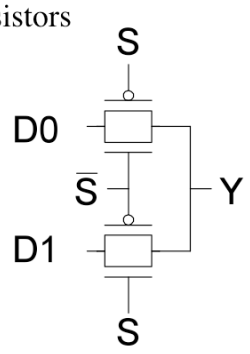




26

Transmission Gate Mux

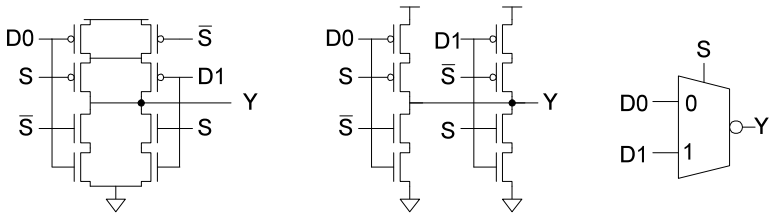
- Nonrestoring mux uses two transmission gates
  - Only 4 transistors



27

Inverting Mux

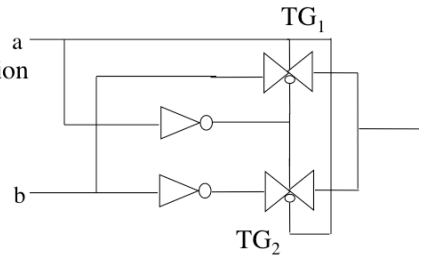
- Inverting multiplexer
  - Use compound AOI22
  - Or pair of tristate inverters
  - Essentially the same thing
- Noninverting multiplexer adds an inverter



28

CMOS Exclusive-Nor Gate

• 8-transistor implementation



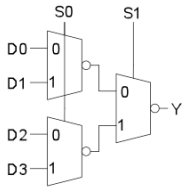
a	b	TG_1	TG_2	F
0	0	nonconducting	conducting	$\overline{B}$ (1)
0	1	nonconducting	conducting	B (0)
1	0	conducting	nonconducting	B (0)
1	1	conducting	nonconducting	B (1)

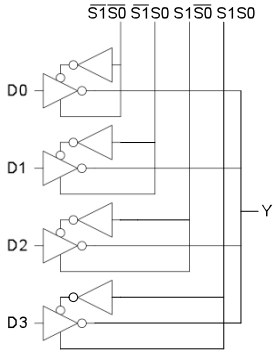
• Better, 6-transistor implementation is possible!

29

4:1 Multiplexer

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

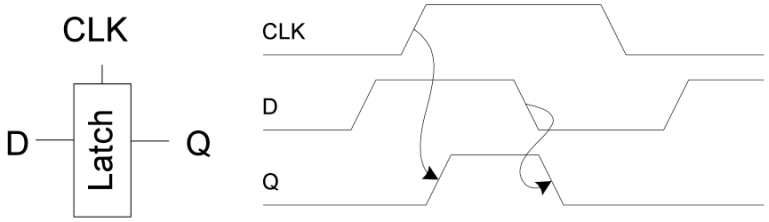




30

D Latch

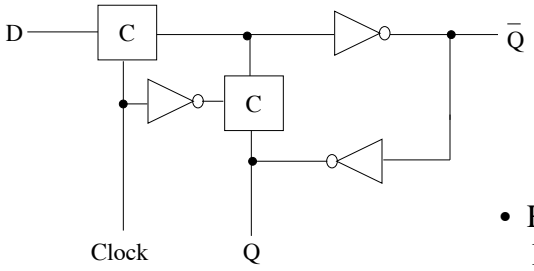
- When CLK = 1, latch is *transparent*
  - D flows through to Q like a buffer
- When CLK = 0, the latch is *opaque*
  - Q holds its old value independent of D
- a.k.a. *transparent latch* or *level-sensitive latch*



31

Memory Elements: Latches and Flip-Flops

- Difference between a latch and a flip-flop?



Positive level-sensitive D-latch

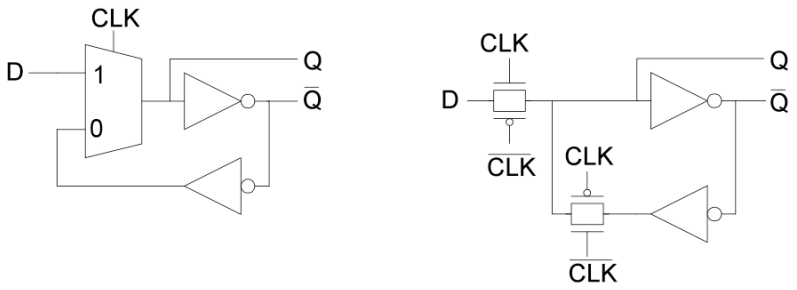
Recirculating latch

- How to design negative level-sensitive D-latch?

32

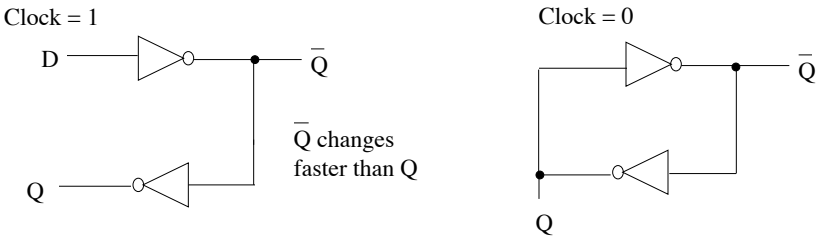
D Latch Design

- Multiplexer chooses D or old Q



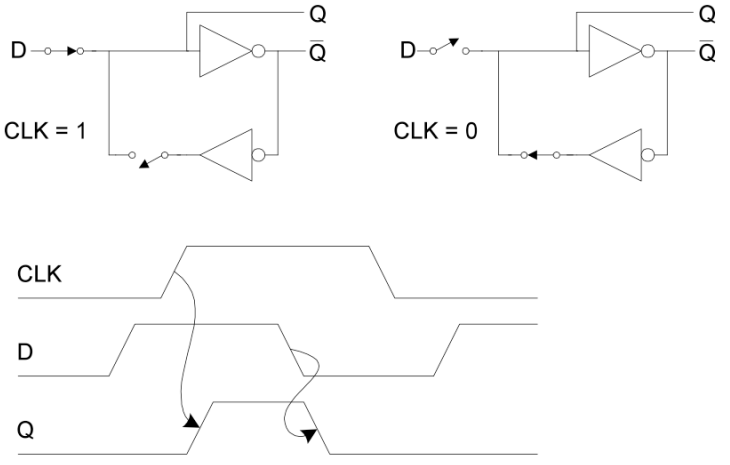
33

Memory Elements: Latches and Flip-Flops



34

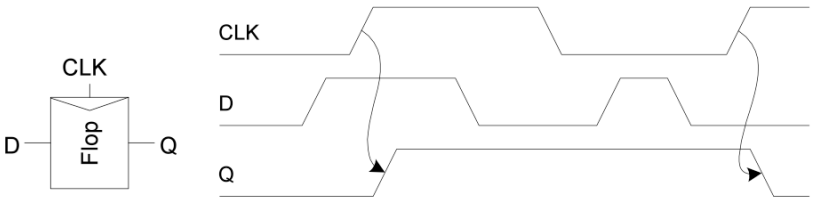
D Latch Operation



35

D Flip-flop

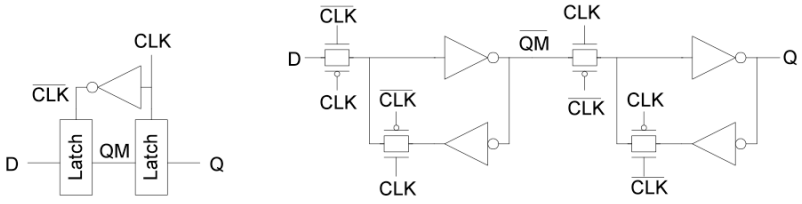
- When CLK rises, D is copied to Q
- At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop*, *master-slave flip-flop*



36

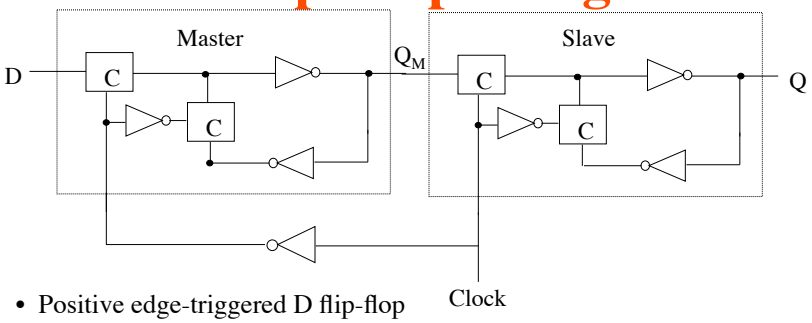
D Flip-flop Design

- Built from master and slave D latches

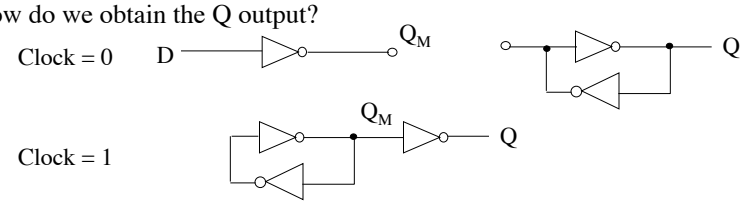


37

Flip-Flop Design

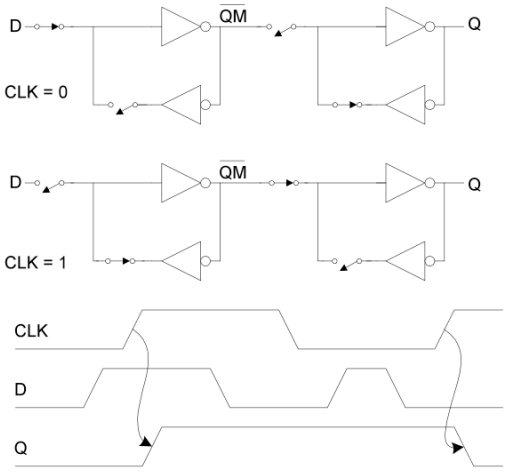


- Positive edge-triggered D flip-flop
- How do we obtain the $\overline{Q}$ output?



38

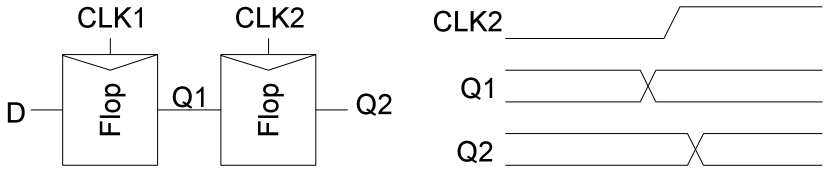
D Flip-flop Operation



39

Race Condition

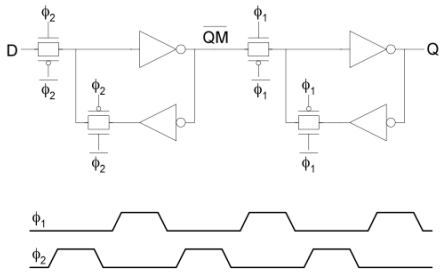
- Back-to-back flops can malfunction from clock skew
  - Second flip-flop fires late
  - Sees first flip-flop change and captures its result
  - Called *hold-time failure* or *race condition*



40

Nonoverlapping Clocks

- Nonoverlapping clocks can prevent races
  - As long as nonoverlap exceeds clock skew
- We will use them in this class for safe design
  - Industry manages skew more carefully instead

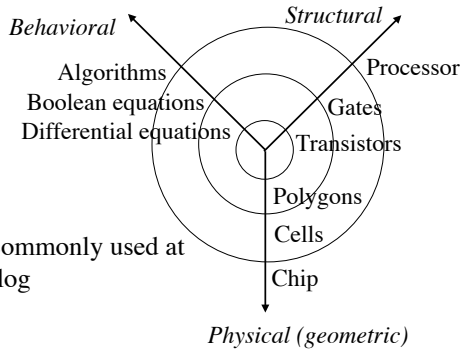


41

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 $c_o = ab + bc + c_i a$

42

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
```

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
```

c_o changes 10 time units after a, b, or c changes

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