# Introduction to CMOS VLSI Design

## Circuits Lecture C

**Peter Kogge**
**University of Notre Dame**
**Fall 2015**

**Based on material from**
**Prof. Jay Brockman, Joseph Nahas: University of Notre Dame**
**Prof. David Harris, Harvey Mudd College**
**http://www.cmosvlsi.com/coursematerials.html**

## Outline: Circuits

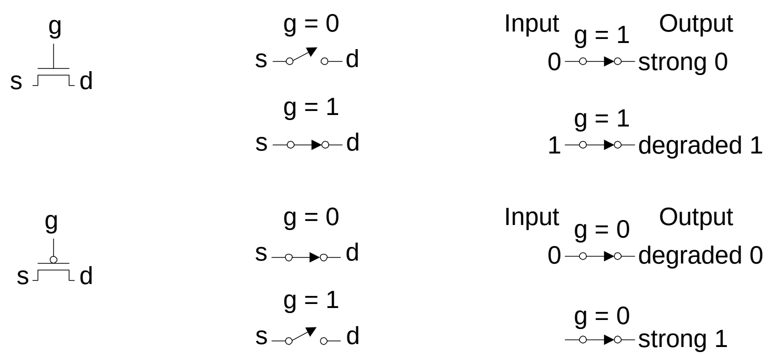
- Lecture A
  - Physics, EE 101
  - Semiconductors
  - CMOS Transistors
- Lecture B
  - NMOS Logic
  - CMOS Inverter and NAND Gate Operation
  - CMOS Gate Design
  - Adders
  - Multipliers
- ***Lecture C***
  - ***Transmission Gates***
  - ***Tri-states***
  - ***Multiplexors***
  - ***Latches***
  - ***FlipFlops***
  - ***Barrel Shifters***

# Transmission Gates

CMOS VLSI Design

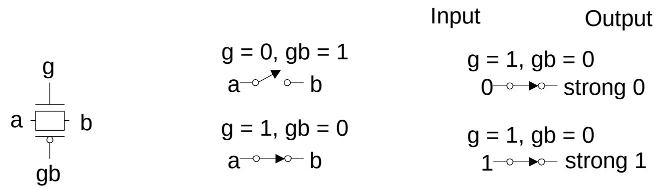
# Pass Transistors

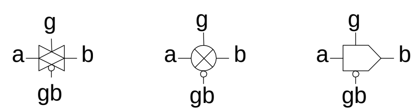
- Transistors can be used as switches in wire



# Transmission Gates

- Individual pass transistors produce degraded outputs
- But what if we *parallel* a p and n type?
- *Transmission gates* pass both 0 and 1 well

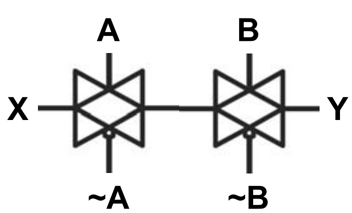



**A "switch in a wire"**

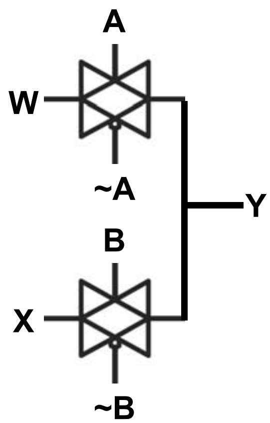


# Combining Transmission Gates




| A | B | Y |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

## Combining Transmission Gates




| A | B | Y |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

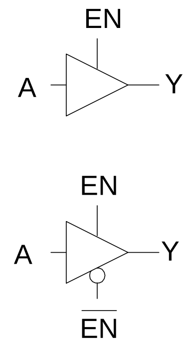


## Tri-state Logic

# Tri-states

- **_Tri-state buffer_ produces _indeterminant output Z_ when not enabled**
  - **Z – neither hi nor low – no current low in either direction**

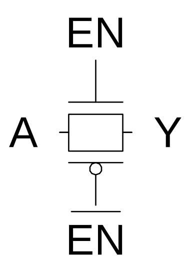
| EN | A | Y |
|---|---|---|
| 0 | 0 | Z |
| 0 | 1 | Z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




Circuits-C CMOS VLSI Design Slide 9

# Nonrestoring Tri-state via Transmission Gate

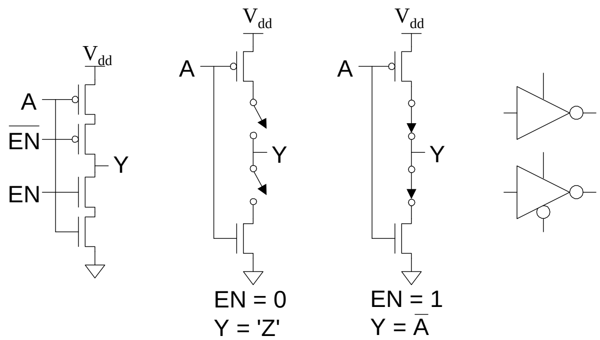
- **Transmission gate acts as Tri-state buffer**
  - **Only two transistors**
  - **But _nonrestoring_**
    - **Noise on A is passed on to Y**




Circuits-C CMOS VLSI Design Slide 10

# Tri-state Inverter

- Tri-state inverter produces restored output
  – Violates conduction complement rule
  – Because we want a Z output



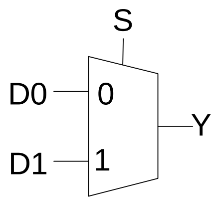



# Multiplexors

# Multiplexers

- 2:1 *multiplexer* chooses between two inputs

| S | D1 | D0 | Y |
|---|---|---|---|
| 0 | X | 0 | |
| 0 | X | 1 | |
| 1 | 0 | X | |
| 1 | 1 | X | |

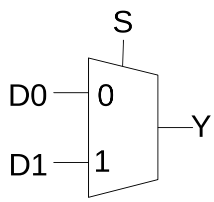





# Multiplexers

- 2:1 multiplexer chooses between two inputs

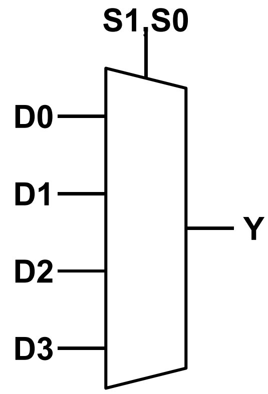
| S | D1 | D0 | Y |
|---|---|---|---|
| 0 | X | 0 | 0 |
| 0 | X | 1 | 1 |
| 1 | 0 | X | 0 |
| 1 | 1 | X | 1 |



## Multiplexers

- 4:1 *multiplexer* chooses between four inputs
- Uses 2 Control signals

| S1,S0 | Y |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |



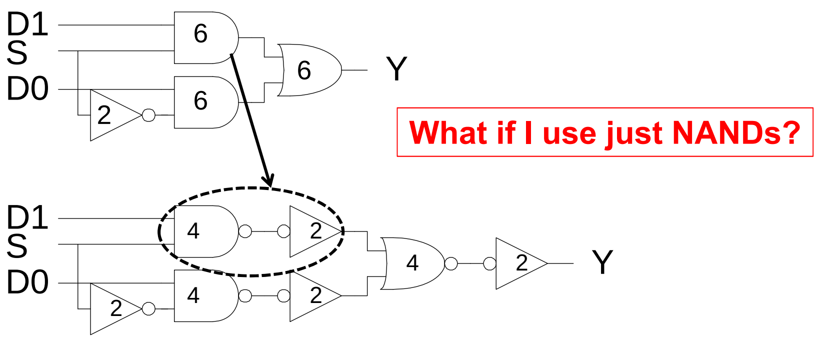



## Logic Gate-Level Mux Design

- $Y = SD_1 + \overline{S}D_0$ (too many transistors)
- How many transistors are needed?

## Gate-Level Mux Design

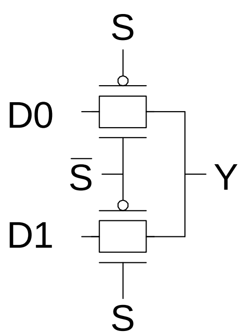
- $Y = SD_1 + \overline{S}D_0$ (too many transistors)
- **How many transistors are needed? 20**





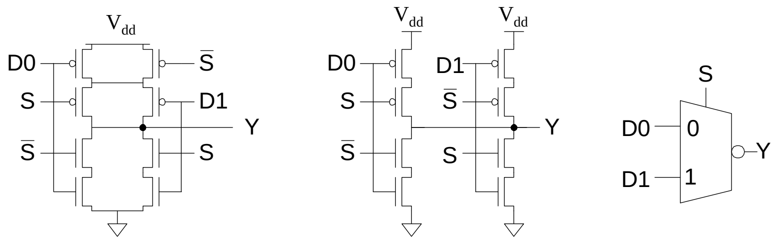

## Transmission Gate Mux

- **Nonrestoring mux uses two transmission gates**
  - **Only 4 transistors**



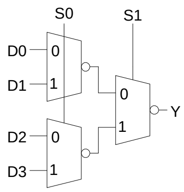
# Inverting Mux

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

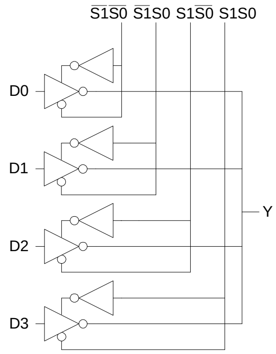




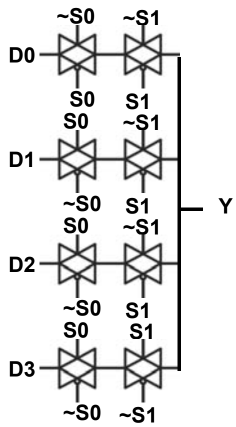

# 4:1 Multiplexer




**3 2input Muxs**




**4 Tri-states**




**8 Transmission Gates**

# Barrel Shifters



# Shifter

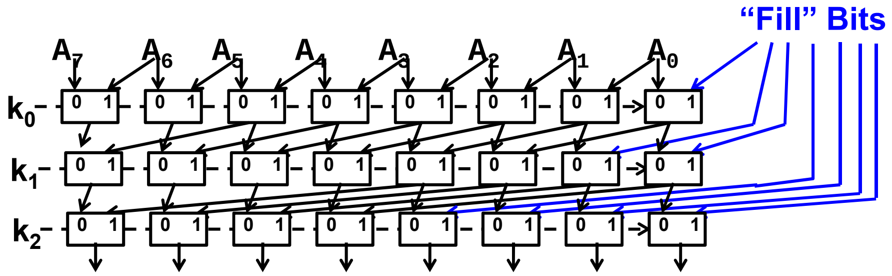
- Given N-bit number, often want to ***shift*** bits in sequence
  - Left or right
  - Circular, or fill with 0, or "arithmetic"
- Equivalent to multiplying/dividing by power of 2
- Options on what gets "shifted in"

Eg: shift "95" by 3 right

```
10010101
wxy10010101
00010010 (0-fill or "logical")
10110010 (circular)
11110010 (arithmetic)
```

## Barrel Shifter

- Assume want to shift left by k, $0 \le k \le N-1$ ($N = 2^n$)
- k espressible as n-bit number:
  - $k = k_{n-1}2^{n-1} + k_{n-1}2^{n-2} + \ldots k_1 2 + k_0$ , $k_i$ a 0 or 1
- Barrel Shifter: construct from n levels of N 2-in multiplexors
  - When level i either shifts last level by $2^{i-1}$ or pass unchanged




Many Variations Possible

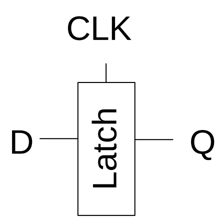


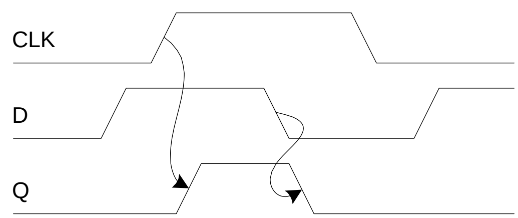
## Basic Latches

see https://en.wikipedia.org/wiki/Flip-flop_%28electronics%29

# 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***

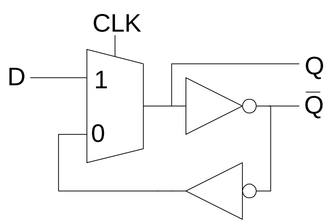


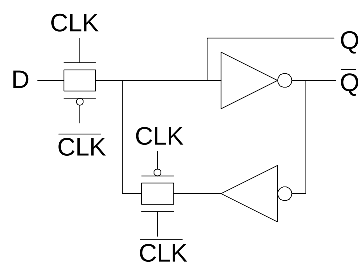





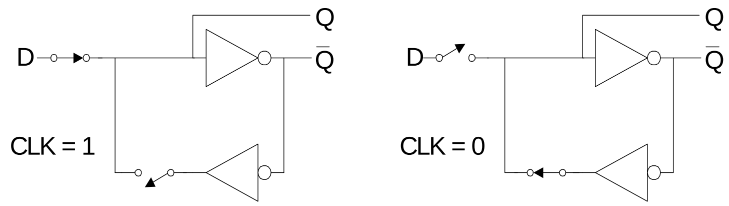

# D Latch Design

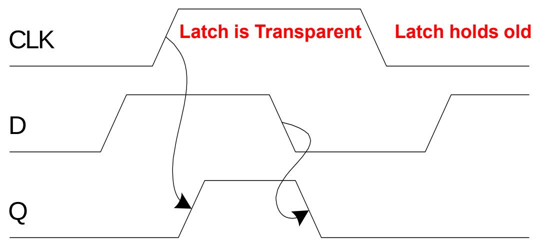
- Multiplexer chooses D or old Q



## D Latch Operation

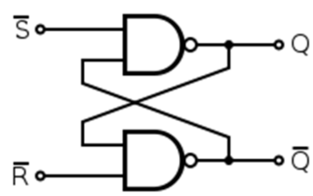




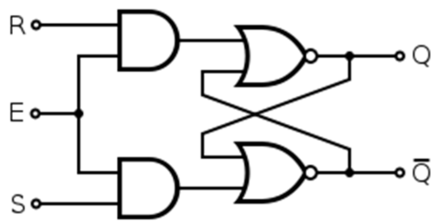

## Set-Reset Latch

| SR Latch (Set-Reset) | | |
|---|---|---|
| S | R | Action |
| 0 | 0 | No Change |
| 0 | 1 | Q = 0 |
| 1 | 0 | Q = 1 |
| 1 | 1 | Invalid |




**Note: its common to use negation of S & R as control inputs**

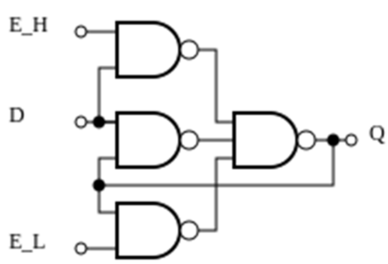
## Gated Set-Reset Latch




**When E is high, acts like prior latch**
**When E is low, no change in output**



## Earle Latch




- **Uses constant 2 gate delays**
- **Needs only 1 input (not inverted)**
- **Can merge more complex logic functions into latch**
- **Hazard free**
- **Used in IBM 360/Mod 91 pipeline**

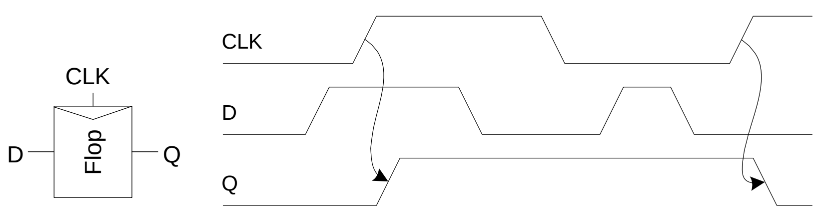
# Clocked Latches:
# Flip-Flops

**see https://en.wikipedia.org/wiki/Flip-flop_%28electronics%29**



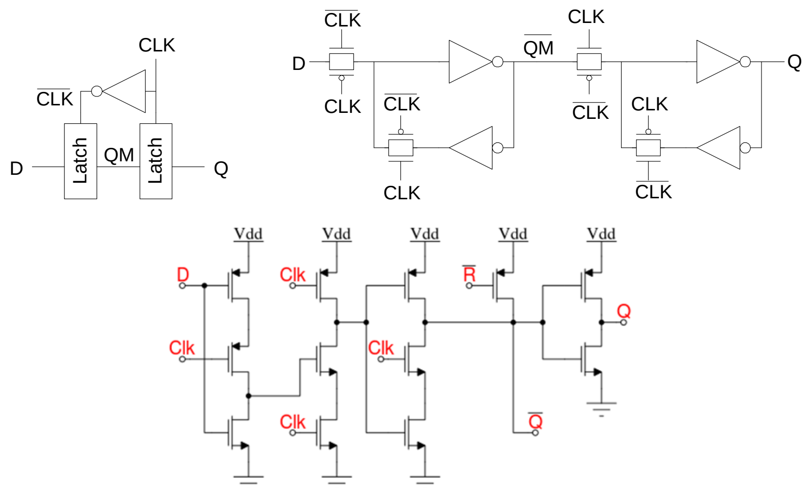
# Clocked Latches: D Flip-flop

- **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***



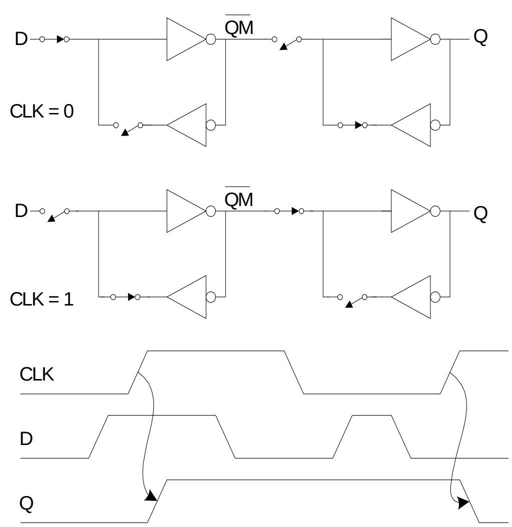
# D Flip-flop Design

- Built from **master** and **slave** D latches



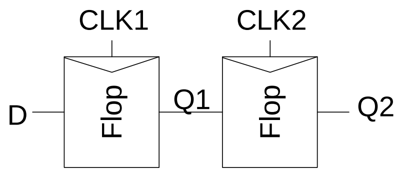


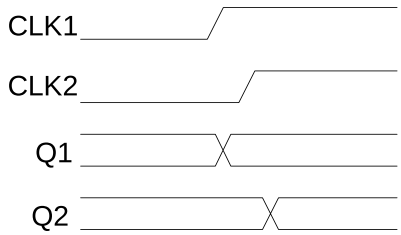

# D Flip-flop Operation



# 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*

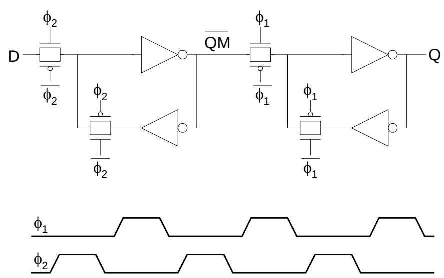




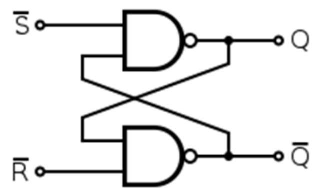

# Nonoverlapping Clocks

- 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



# Set-Reset Latch

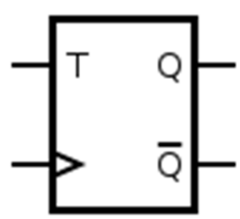
| SR Latch (Set-Reset) | | |
|---|---|---|
| S | R | Action |
| 0 | 0 | Invalid |
| 0 | 1 | Q = 0 |
| 1 | 0 | Q = 1 |
| 1 | 1 | No Change |






# T Flip Flop: $Q_{next}$ = T xor Q

| T Flip-flop | | | |
|---|---|---|---|
| T | Q | After Clock Rise | Action |
| 0 | 0 | 0 | No Change |
| 0 | 1 | 1 | No Change |
| 1 | 0 | 1 | Complement |
| 1 | 1 | 0 | Complement |




**If T is held high, output switches from 0 to 1 at ½ the rate of the Clock I.E. "Divide by 2"**

# JK Flip Flop: $Q_{next} = J\overline{Q} + \overline{K}Q$

| T Flip-flop | | | |
|---|---|---|---|
| J | K | Q After Clock Rise | Action |
| 0 | 0 | Q old | No Change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | ~Q | Complement |