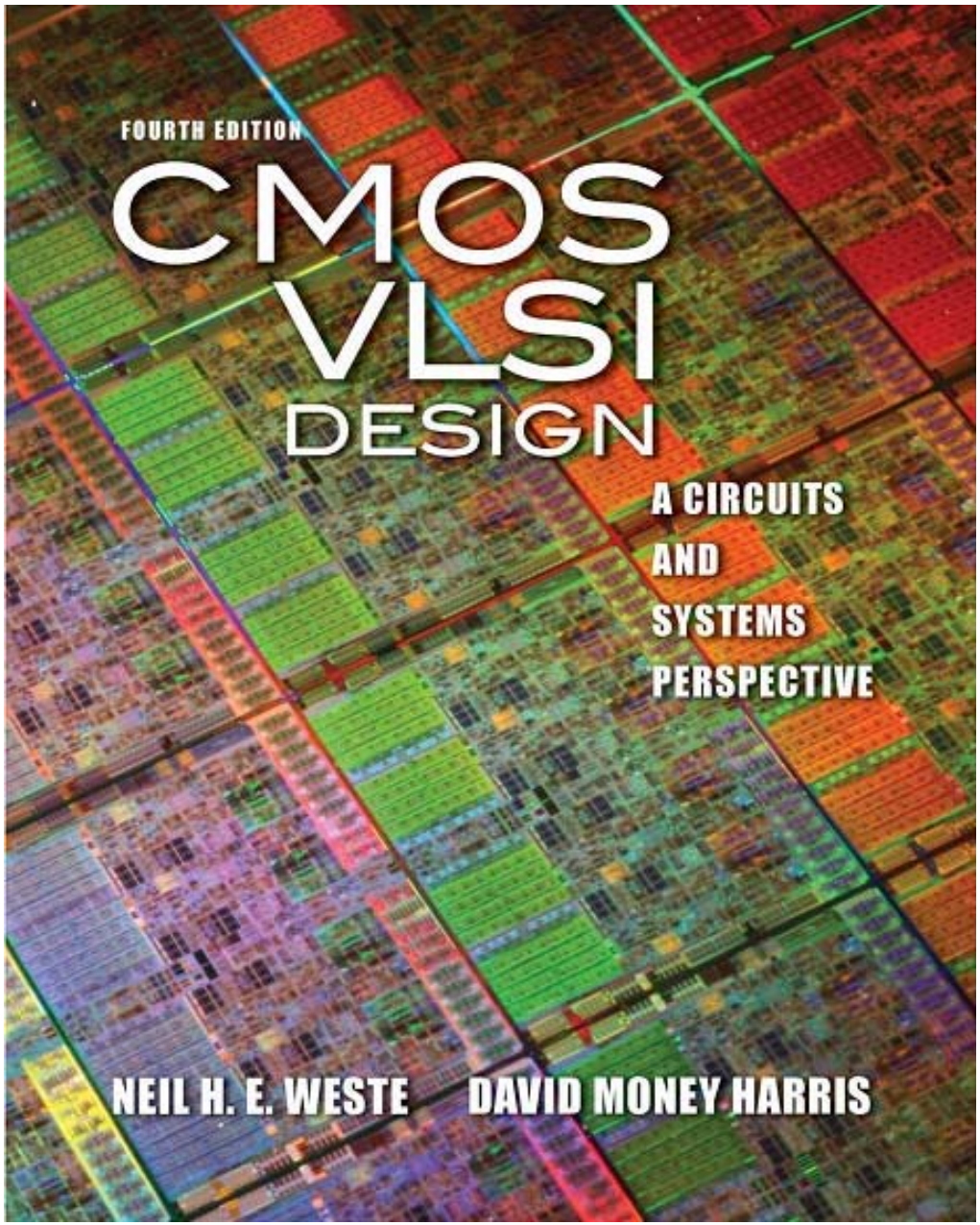


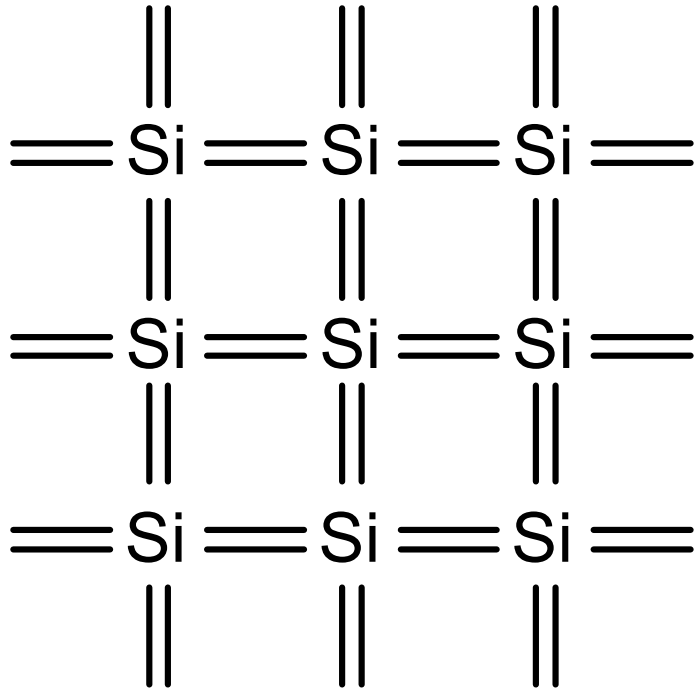
Week_1: Introduction

Introduction

- Integrated circuits: many transistors on one chip.
- *Very Large Scale Integration* (VLSI): bucketloads!
- *Complementary Metal Oxide Semiconductor*
  - Fast, cheap, low power transistors
- Today: How to build your own simple CMOS chip
  - CMOS transistors
  - Building logic gates from transistors
  - Transistor layout and fabrication
- Rest of the course: How to build a good CMOS chip

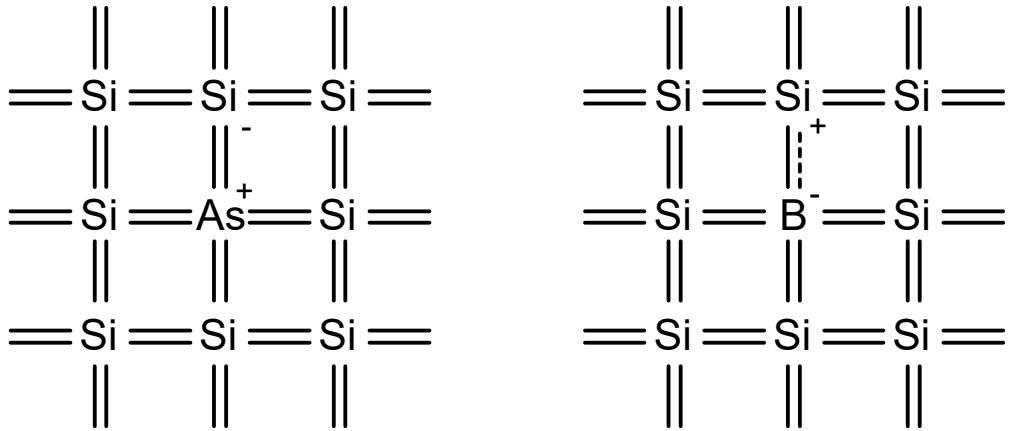
Silicon Lattice

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



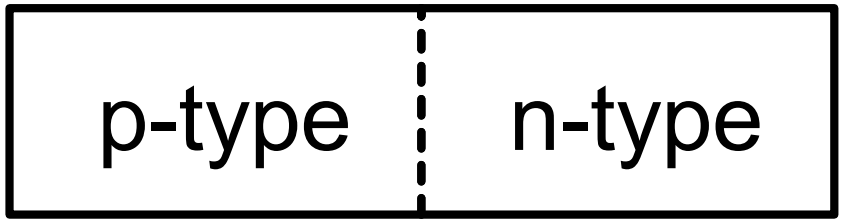
Dopants

- 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)

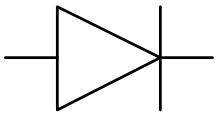


p-n Junctions

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

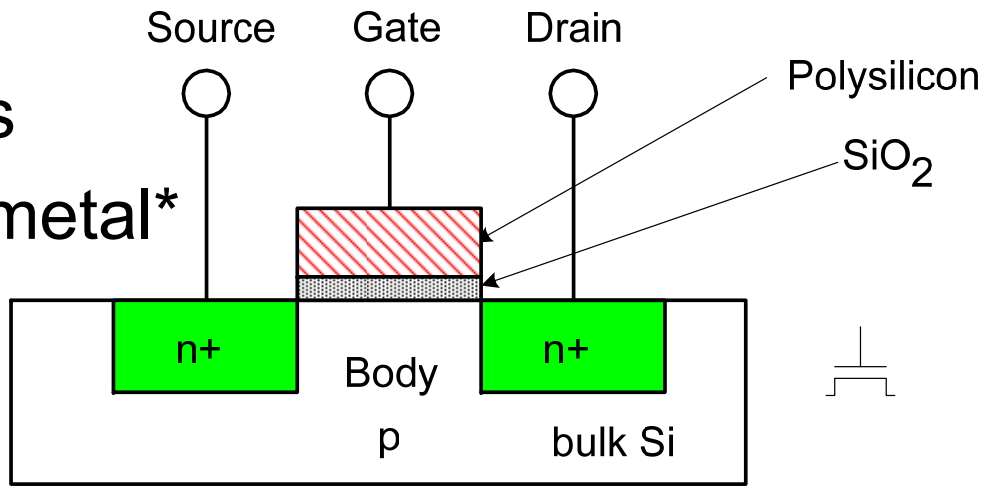


anode cathode



nMOS Transistor

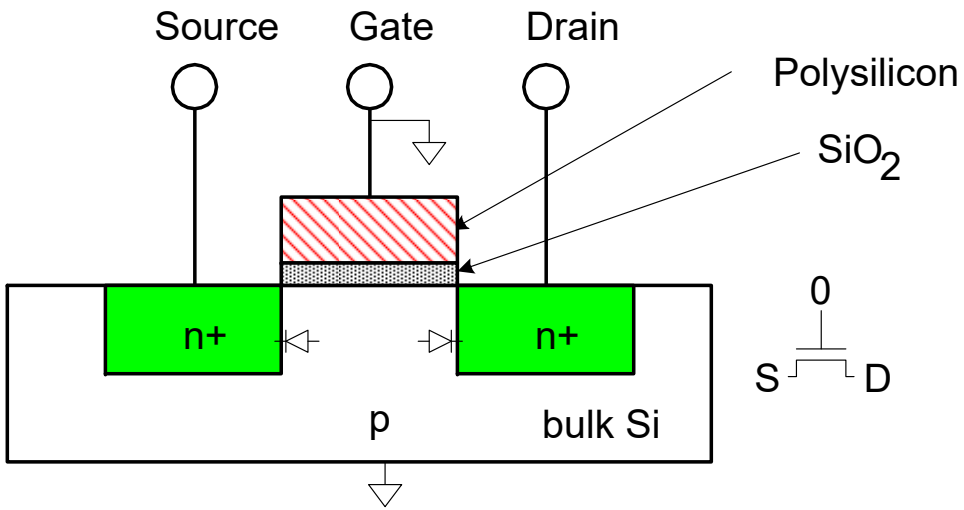
- Four terminals: gate, source, drain, body
- Gate – oxide – body stack looks like a capacitor
  - Gate and body are conductors
  - SiO_2 (oxide) is a very good insulator
  - Called metal – oxide – semiconductor (MOS) capacitor
  - Even though gate is no longer made of metal*



* Metal gates are returning today!

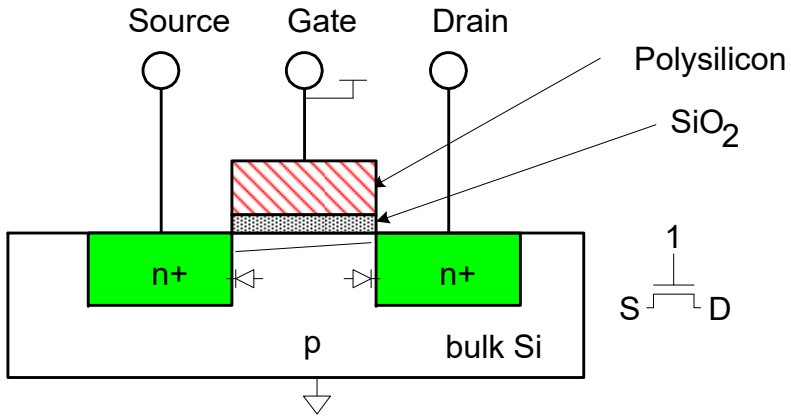
nMOS Operation

- Body is usually 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



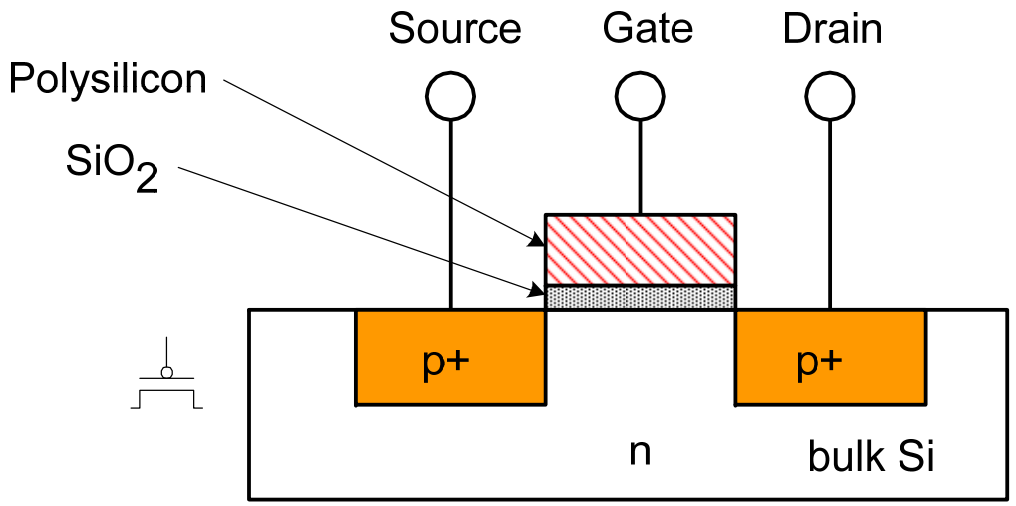
nMOS Operation Cont.

- 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

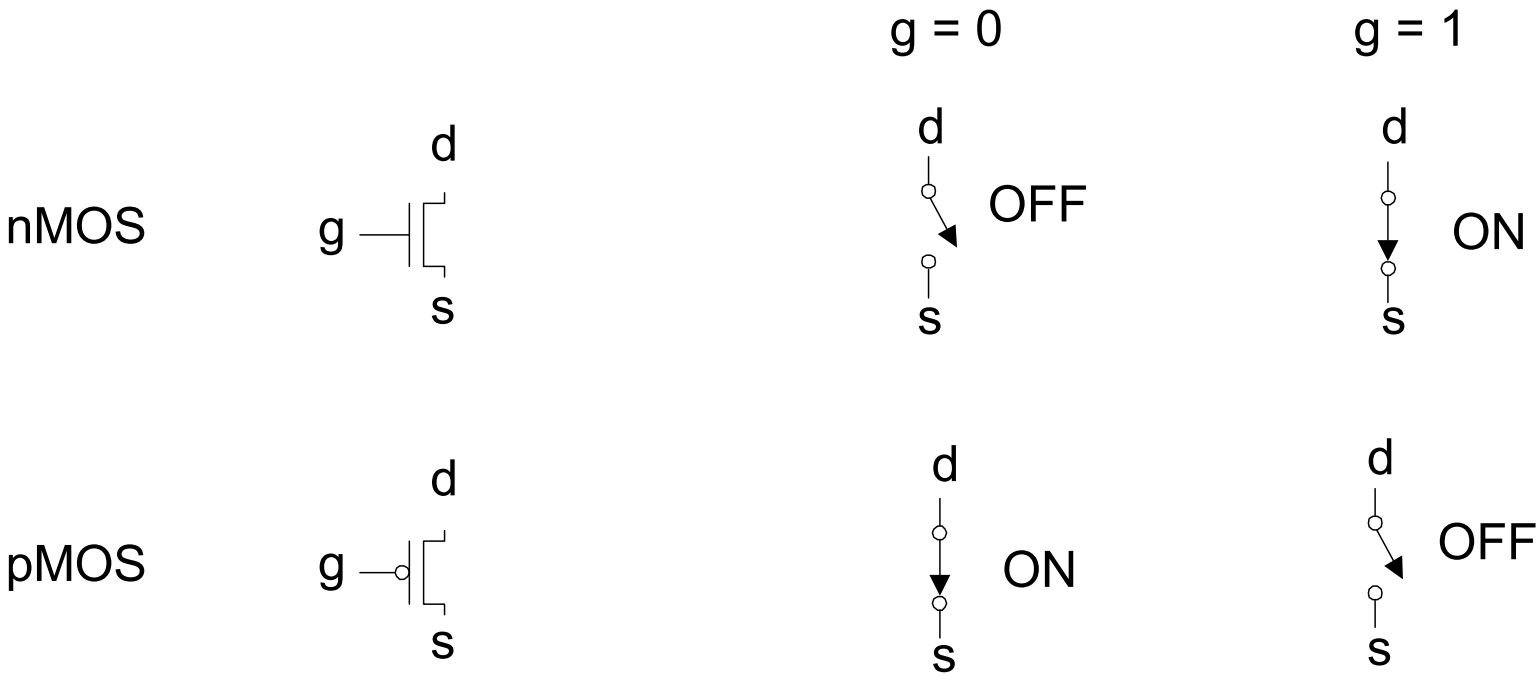


Power Supply Voltage

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

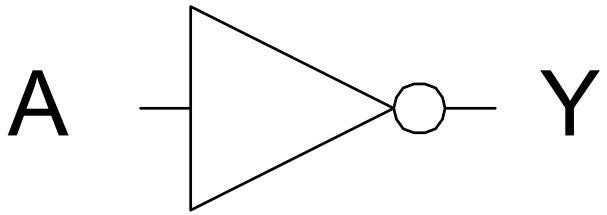
Transistors as Switches

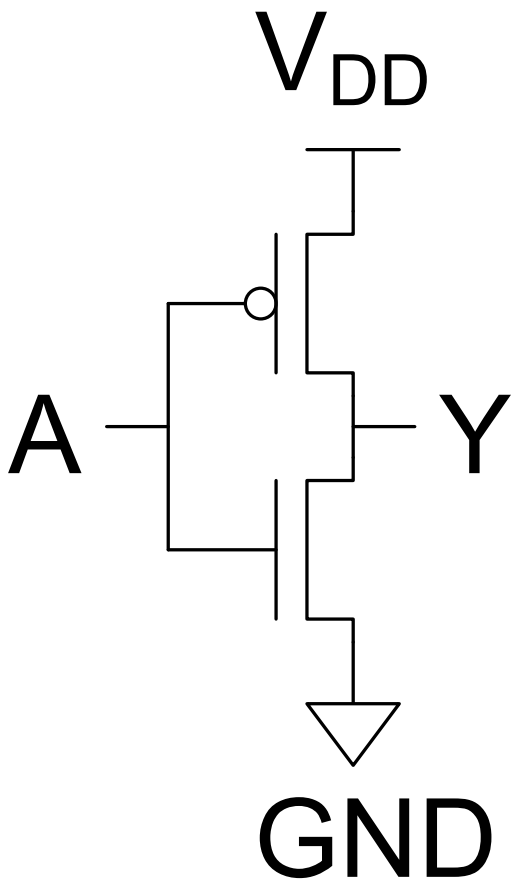
- We can view MOS transistors as electrically controlled switches
- Voltage at gate controls path from source to drain



CMOS Inverter

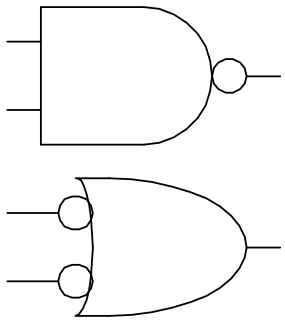
A	Y

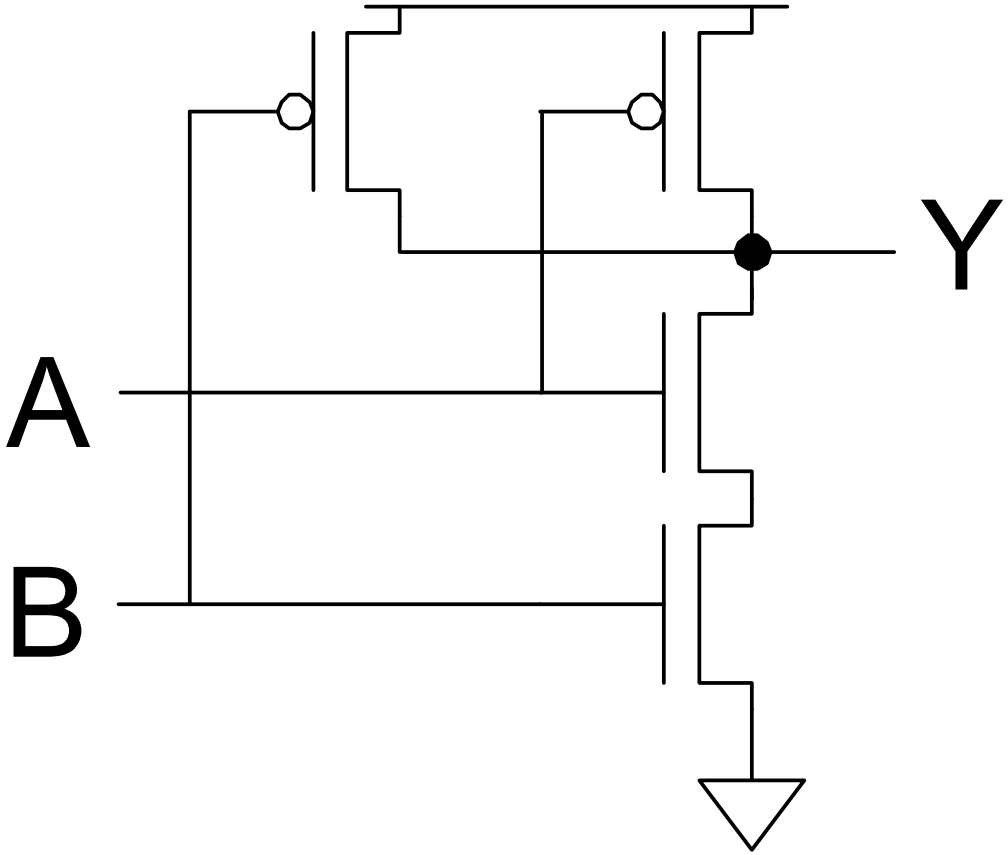




CMOS NAND Gate

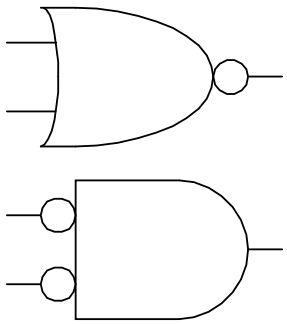
A	B	Y
0	0	
0	1	
1	0	
1	1	

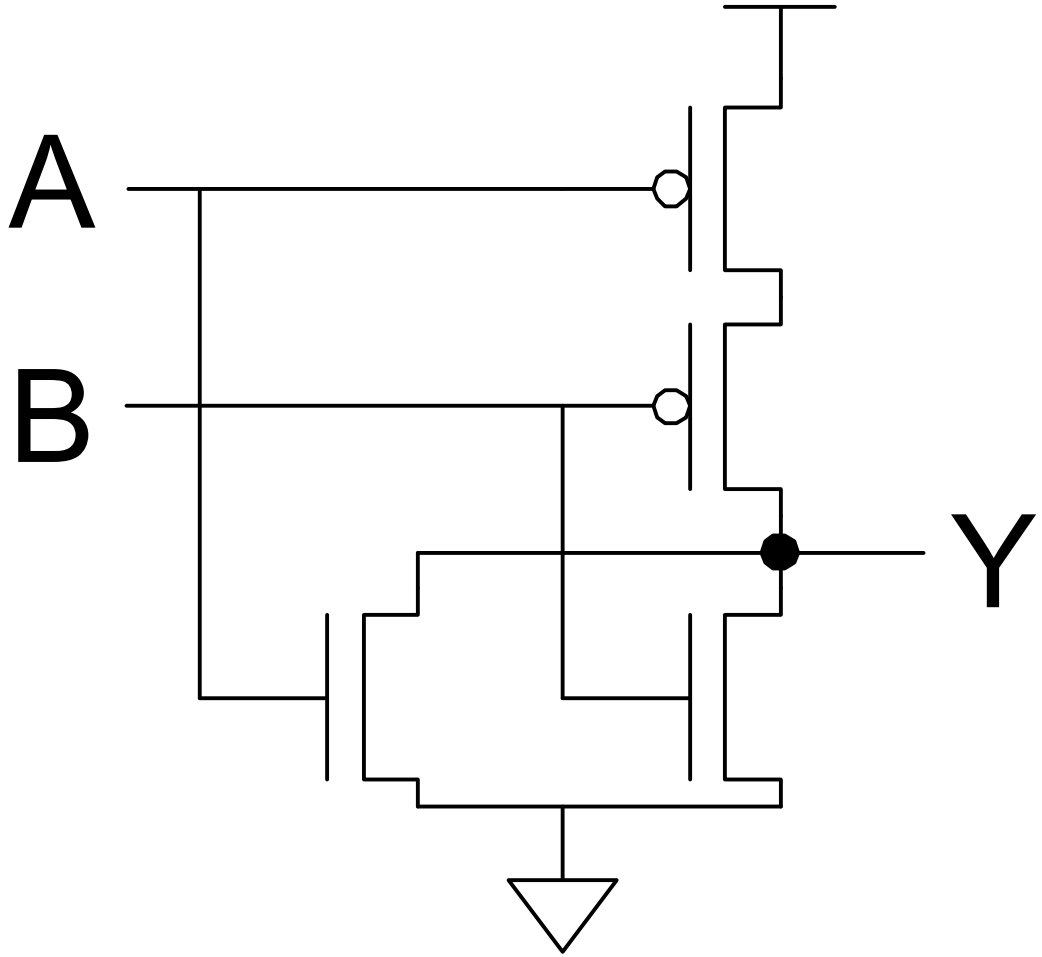




CMOS NOR Gate

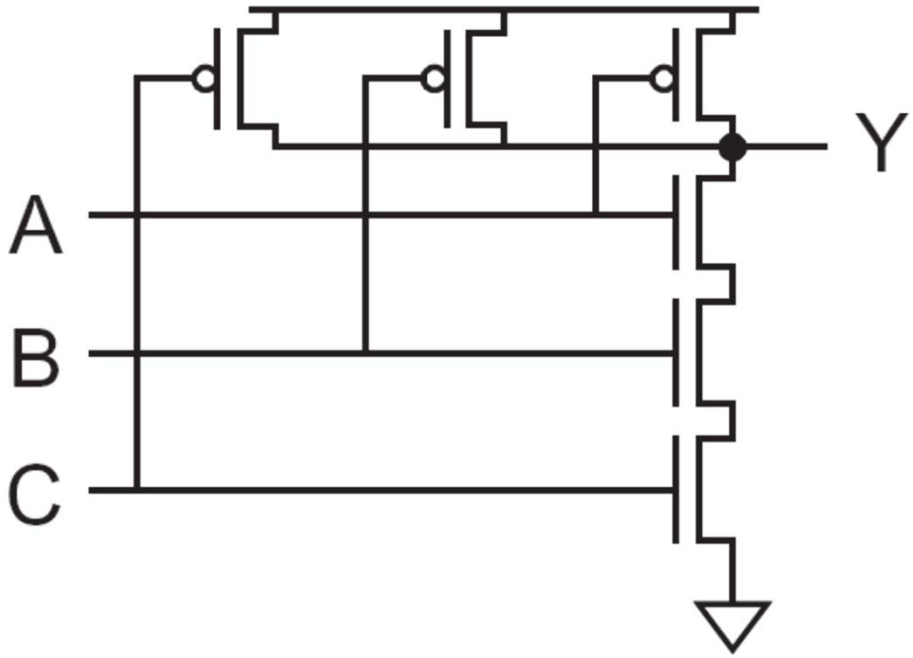
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0





3-input NAND Gate

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

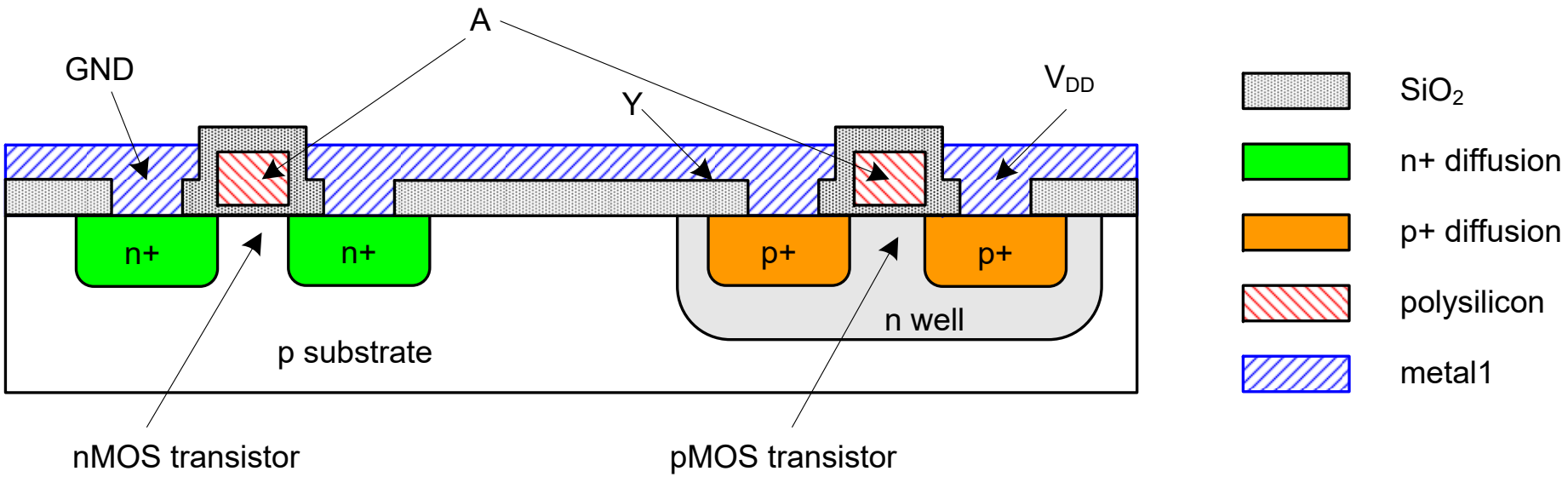


CMOS Fabrication

- CMOS transistors are fabricated on silicon wafer
- Lithography process similar to printing press
- On each step, different materials are deposited or etched
- Easiest to understand by viewing both top and cross-section of wafer in a simplified manufacturing process

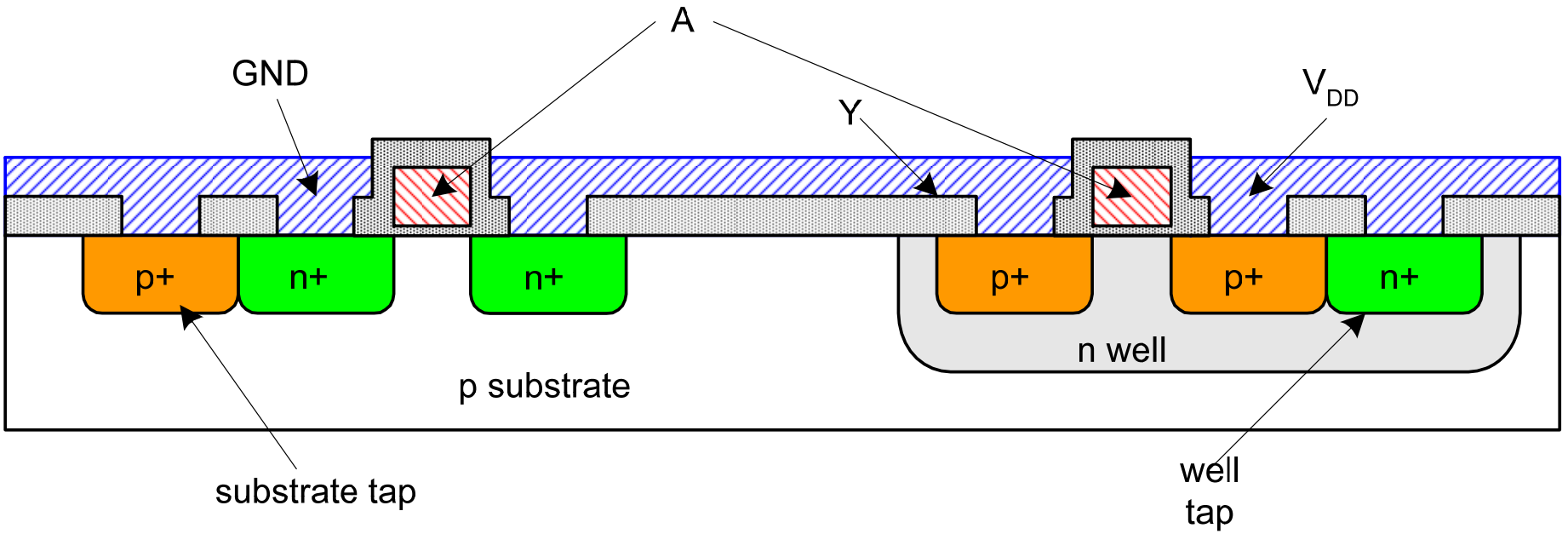
Inverter Cross-section

- Typically use p-type substrate for nMOS transistors
- Requires n-well for body of pMOS transistors



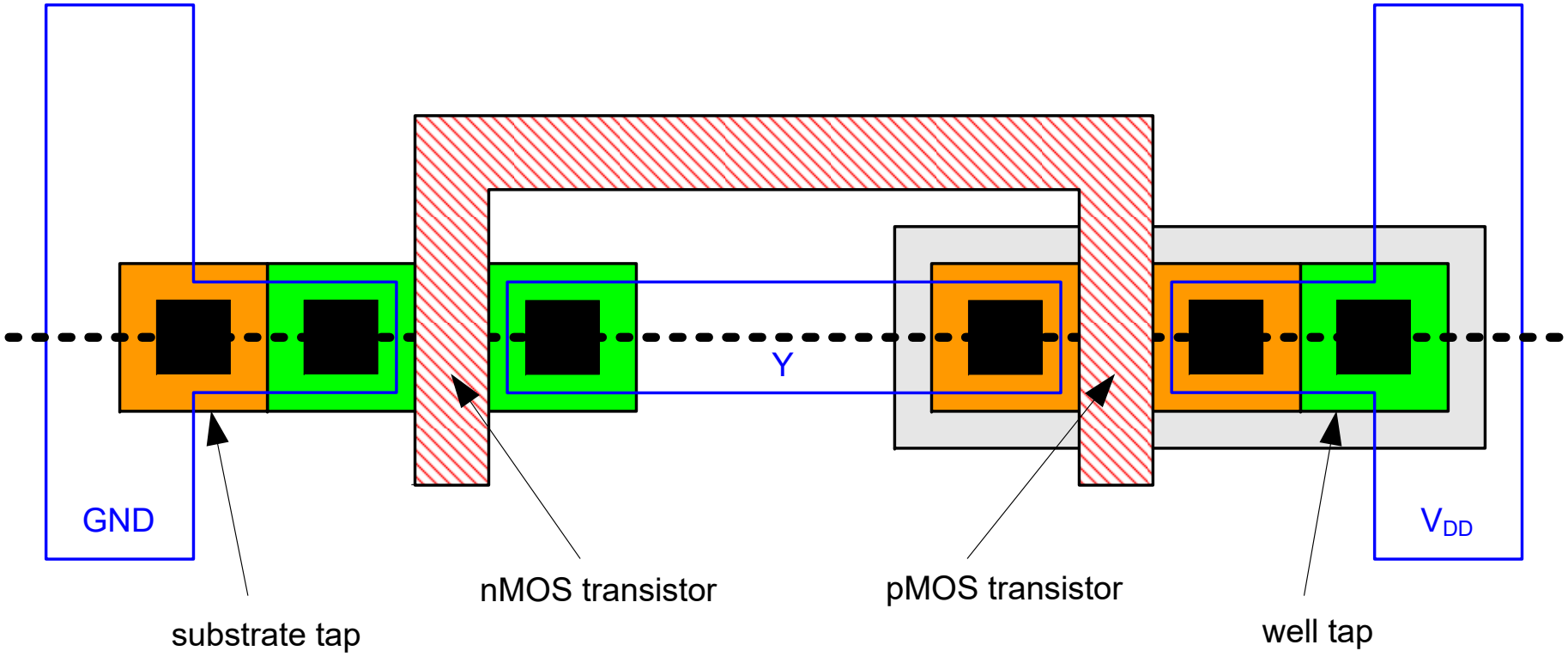
Well and Substrate Taps

- Substrate must be tied to GND and n-well to V_{DD}
- Metal to lightly-doped semiconductor forms poor connection called Shottky Diode
- Use heavily doped well and substrate contacts / taps



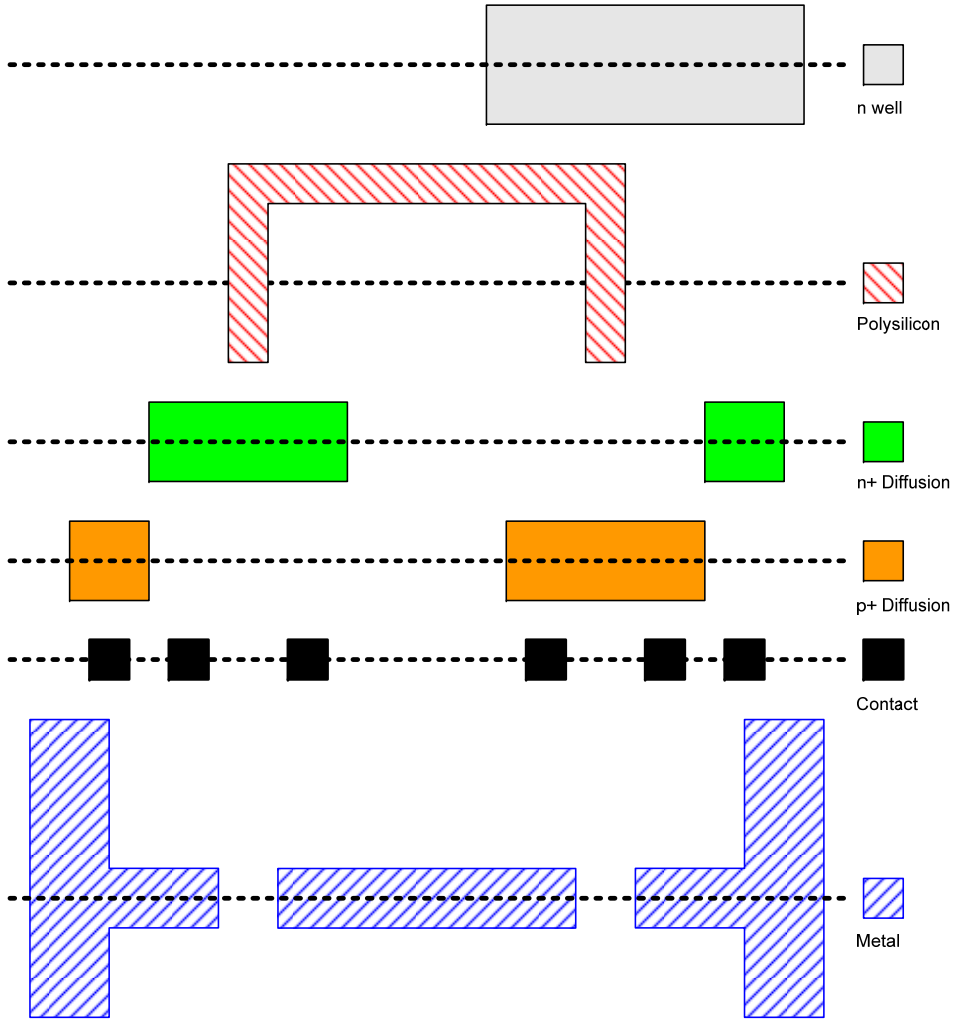
Inverter Mask Set

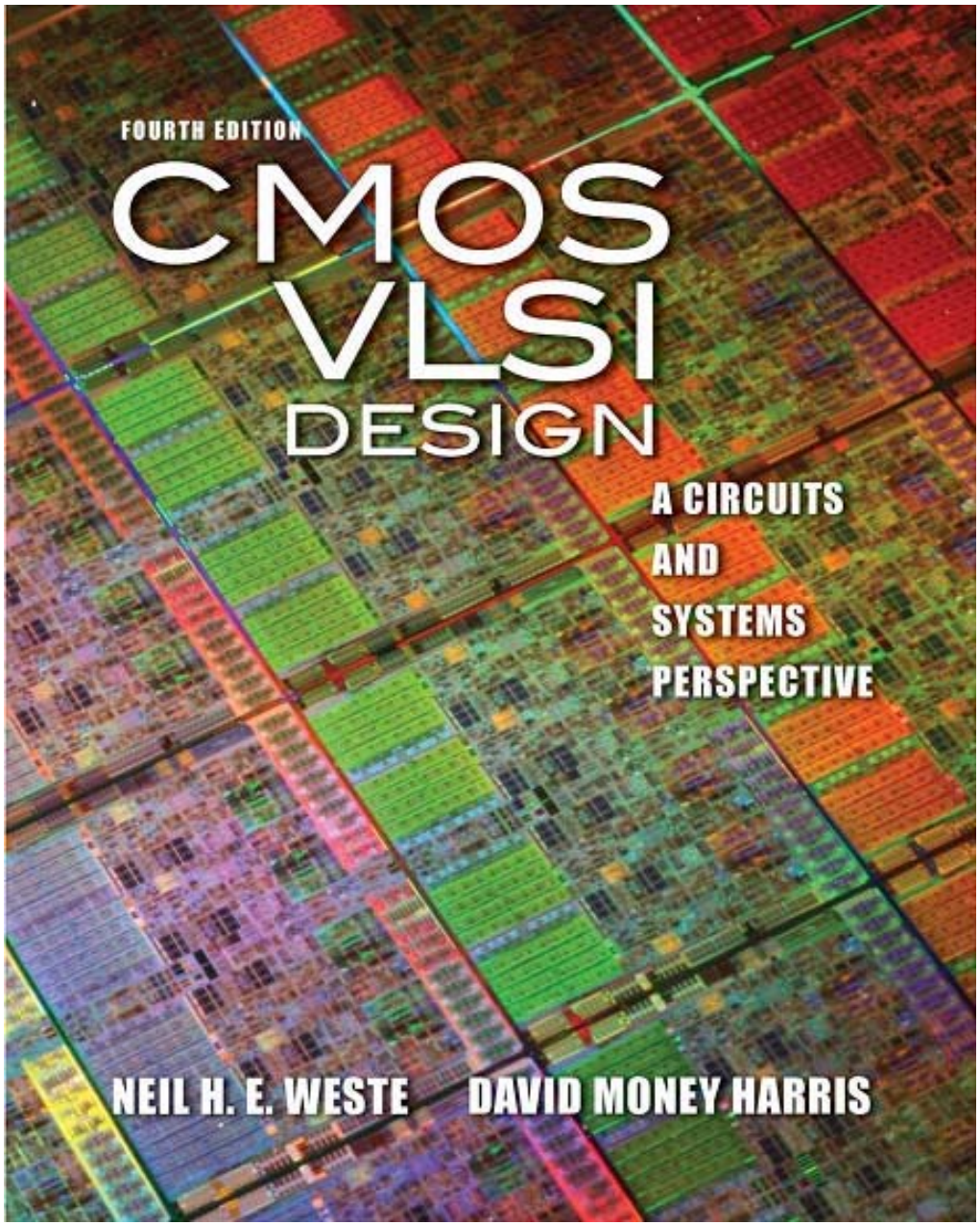
- Transistors and wires are defined by *masks*
- Cross-section taken along dashed line



Detailed Mask Views

- Six masks
  - n-well
  - Polysilicon
  - n+ diffusion
  - p+ diffusion
  - Contact
  - Metal

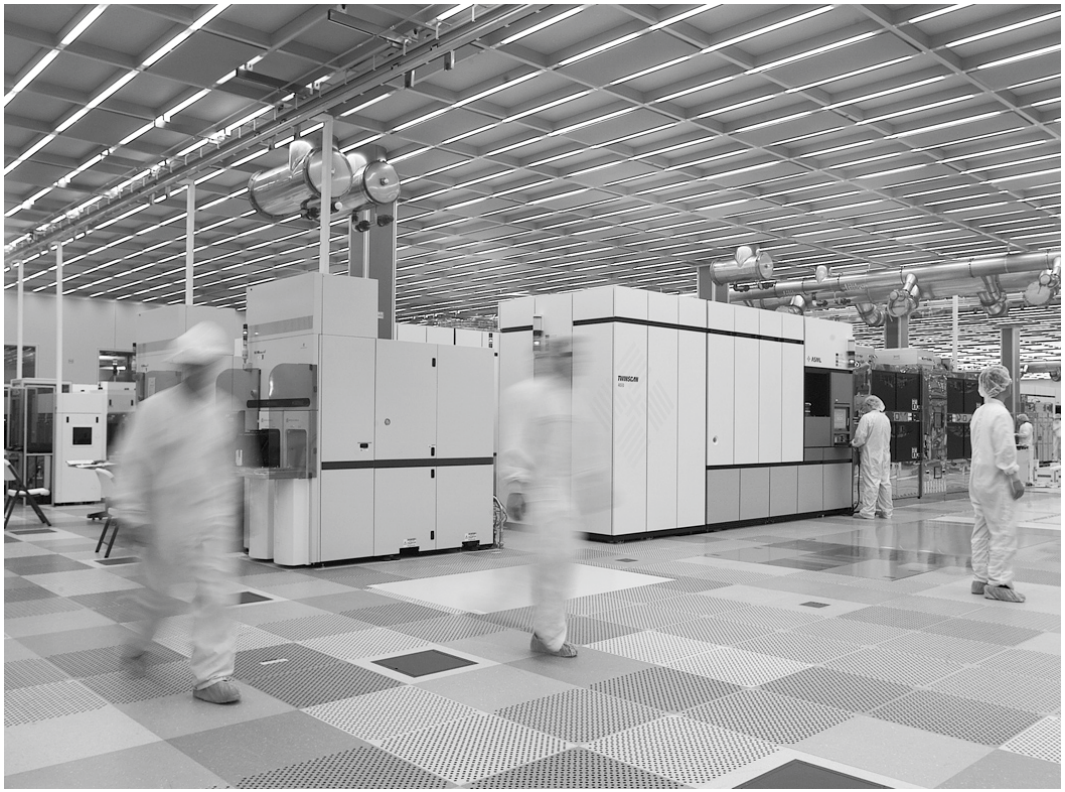




Week_2: Introduction

Fabrication

- Chips are built in huge factories called fabs
- Contain clean rooms as large as football fields

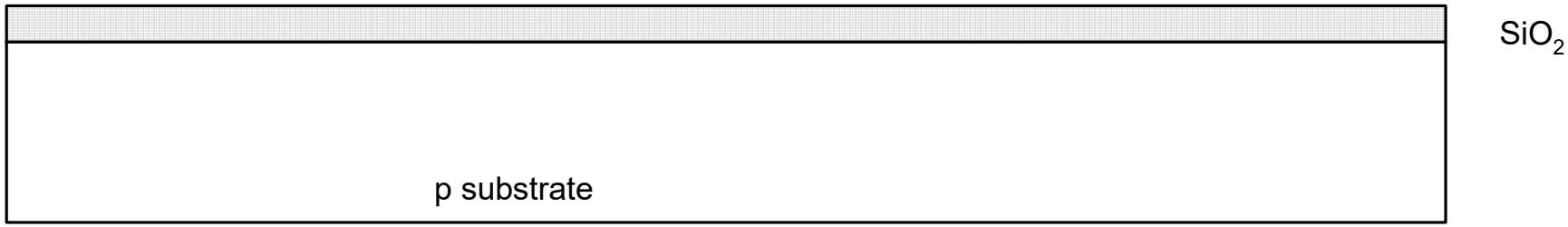


Fabrication Steps

- Start with blank wafer
- Build inverter from the bottom up
- First step will be to form the n-well
  - Cover wafer with protective layer of SiO_2 (oxide)
  - Remove layer where n-well should be built
  - Implant or diffuse n dopants into exposed wafer
  - Strip off SiO_2

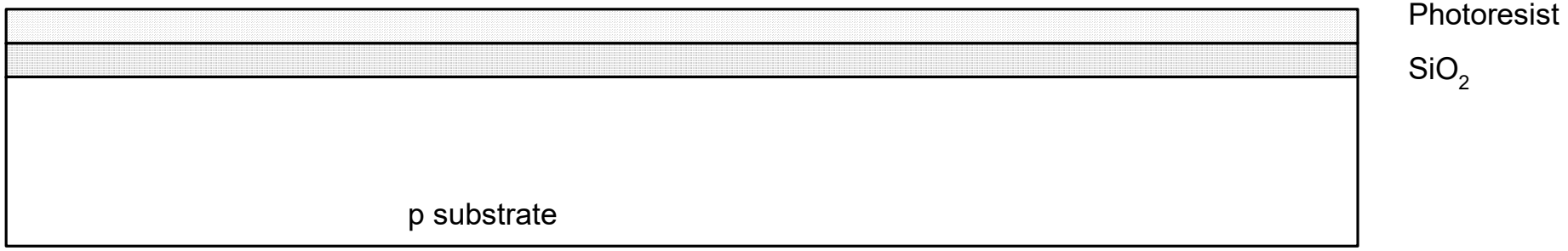
Oxidation

- Grow SiO_2 on top of Si wafer
  - 900 – 1200 C with H_2O or O_2 in oxidation furnace



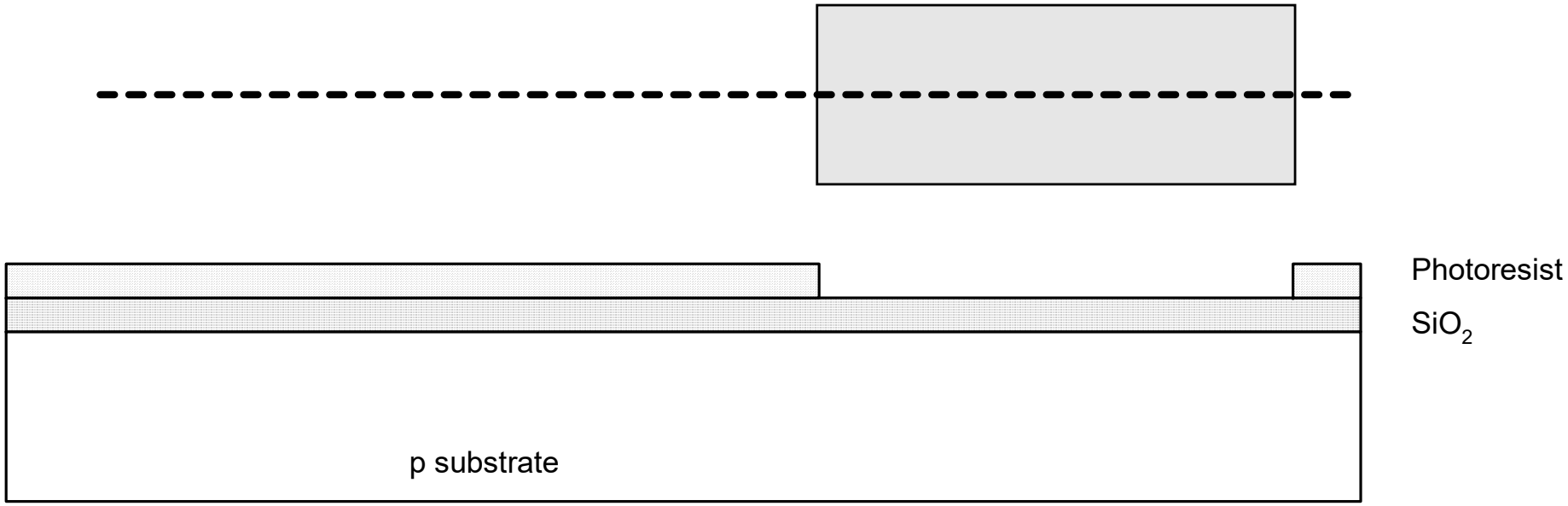
Photoresist

- Spin on photoresist
  - Photoresist is a light-sensitive organic polymer
  - Softens where exposed to light



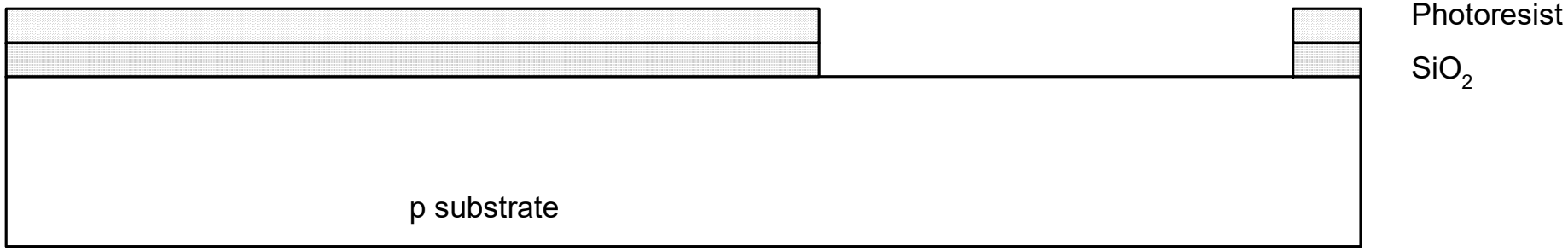
Lithography

- Expose photoresist through n-well mask
- Strip off exposed photoresist



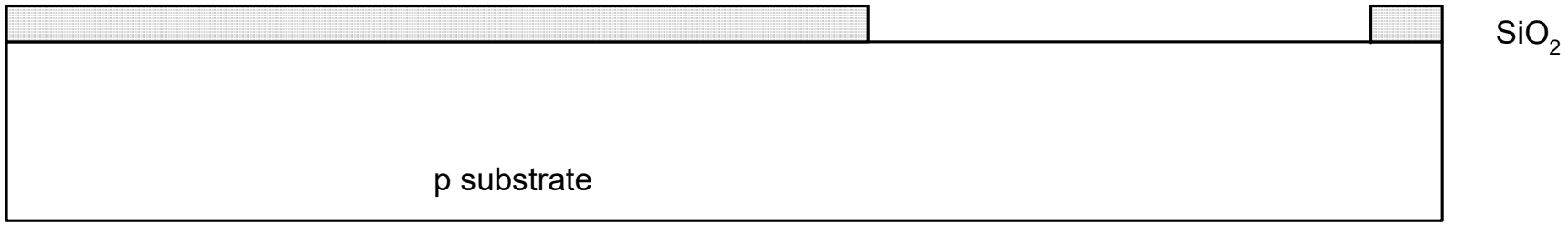
Etch

- Etch oxide with hydrofluoric acid (HF)
  - Seeps through skin and eats bone; nasty stuff!!!
- Only attacks oxide where resist has been exposed



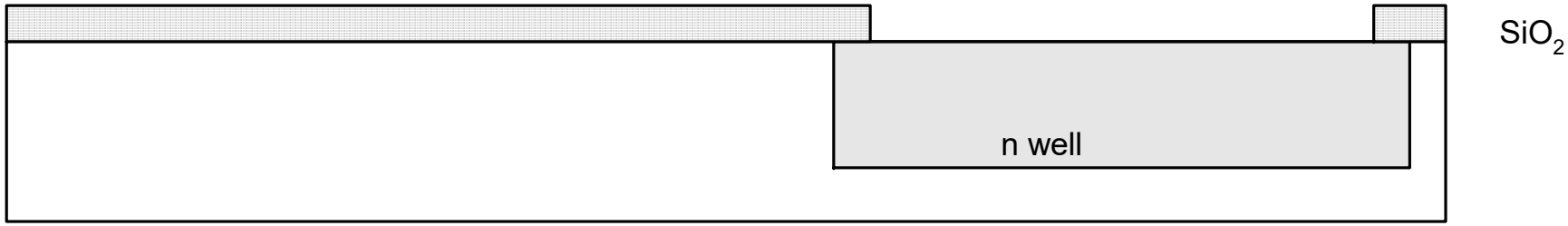
Strip Photoresist

- Strip off remaining photoresist
  - Use mixture of acids called piranah etch
- Necessary so resist doesn't melt in next step



n-well

- n-well is formed with diffusion or ion implantation
- Diffusion
  - Place wafer in furnace with arsenic gas
  - Heat until As atoms diffuse into exposed Si
- Ion Implantation
  - Blast wafer with beam of As ions
  - Ions blocked by SiO_2, only enter exposed Si

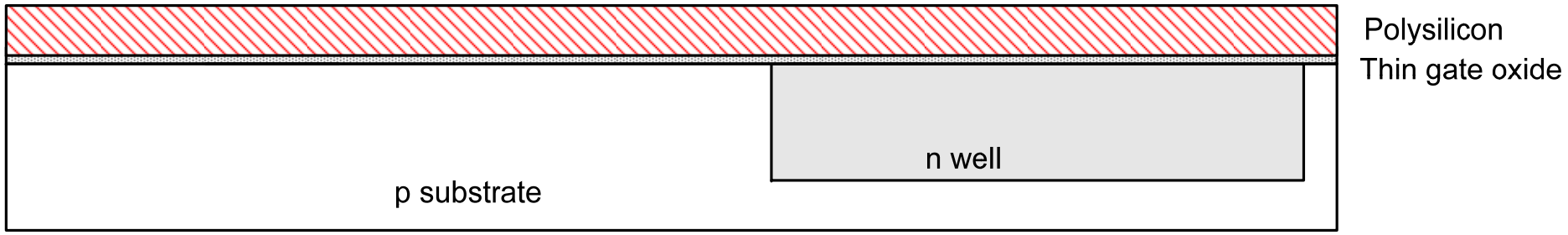


Strip Oxide

- Strip off the remaining oxide using HF
- Back to bare wafer with n-well
- Subsequent steps involve similar series of steps

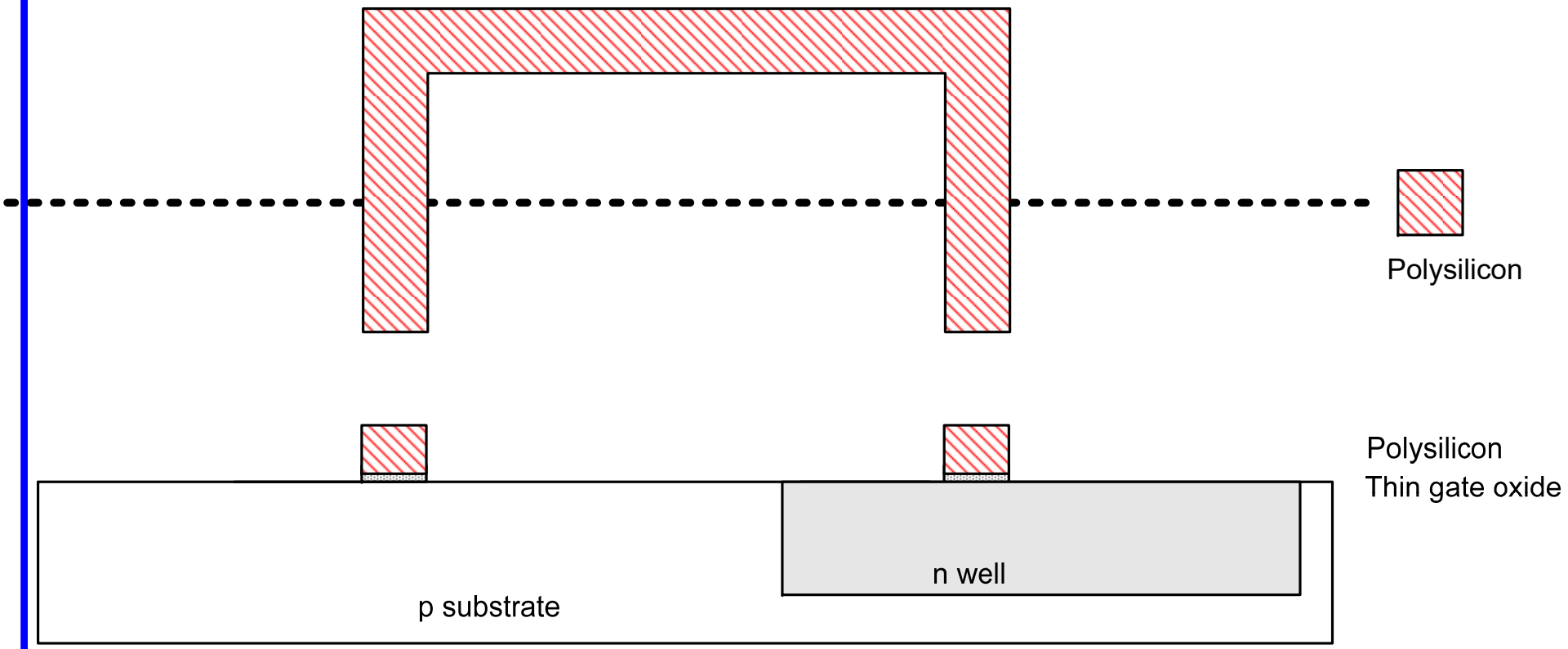
Polysilicon

- Deposit very thin layer of gate oxide
  - < 20 Å (6-7 atomic layers)
- Chemical Vapor Deposition (CVD) of silicon layer
  - Place wafer in furnace with Silane gas (SiH_4)
  - Forms many small crystals called polysilicon
  - Heavily doped to be good conductor



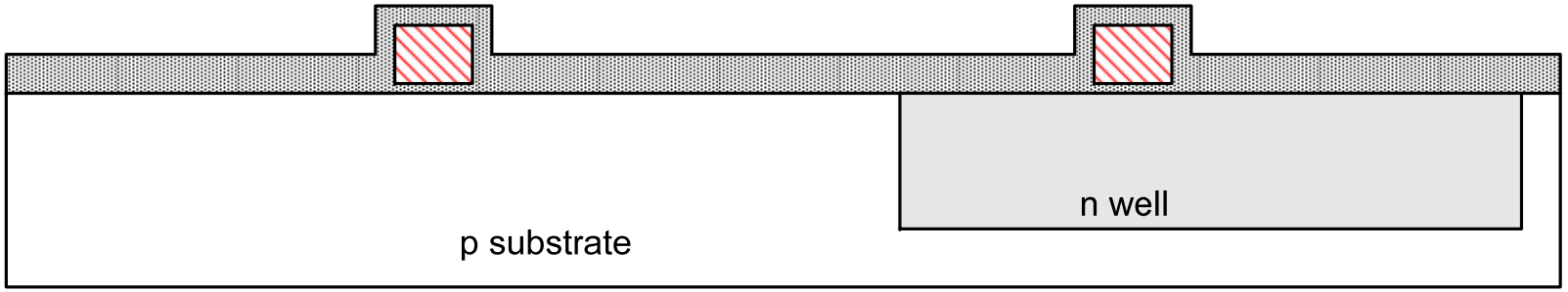
Polysilicon Patterning

- Use same lithography process to pattern polysilicon



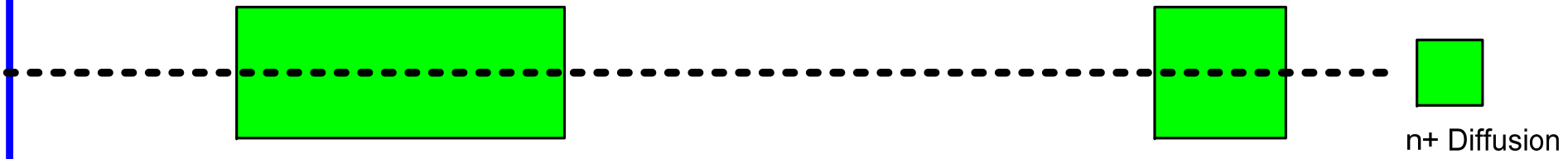
Self-Aligned Process

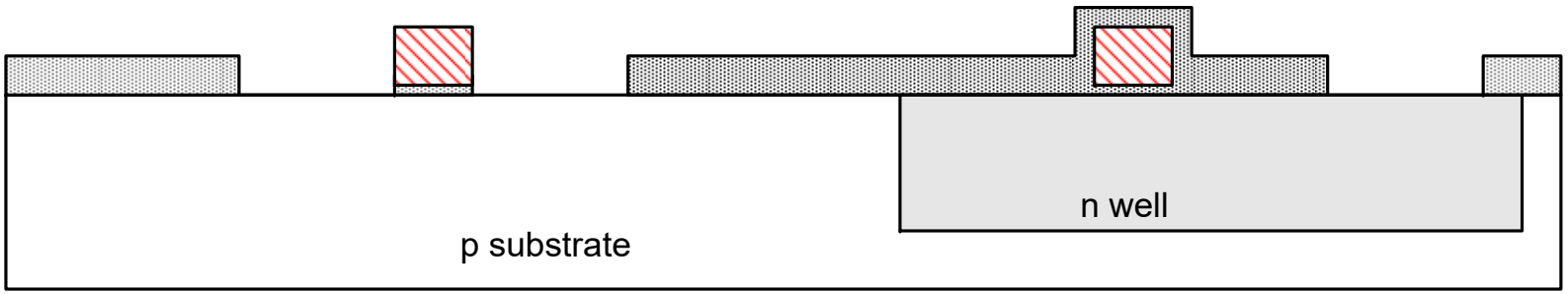
- Use oxide and masking to expose where n+ dopants should be diffused or implanted
- N-diffusion forms nMOS source, drain, and n-well contact



N-diffusion

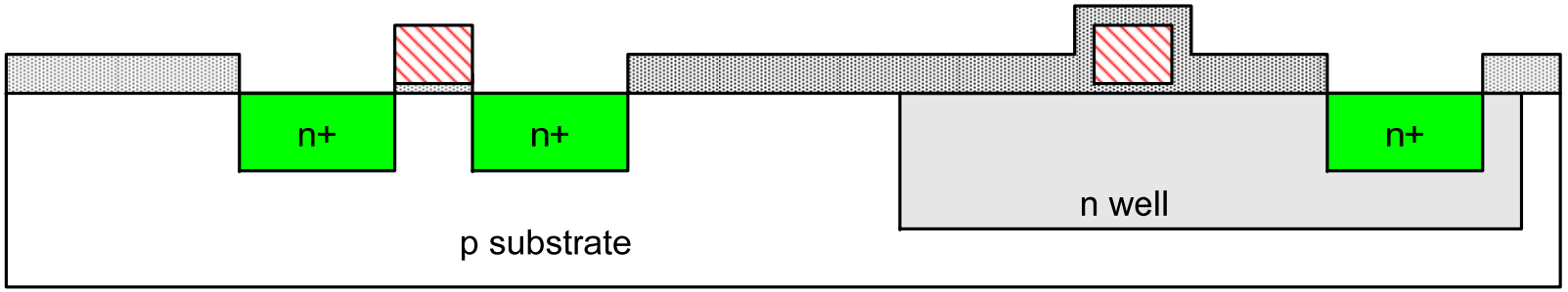
- Pattern oxide and form n+ regions
- *Self-aligned process* where gate blocks diffusion
- Polysilicon is better than metal for self-aligned gates because it doesn't melt during later processing





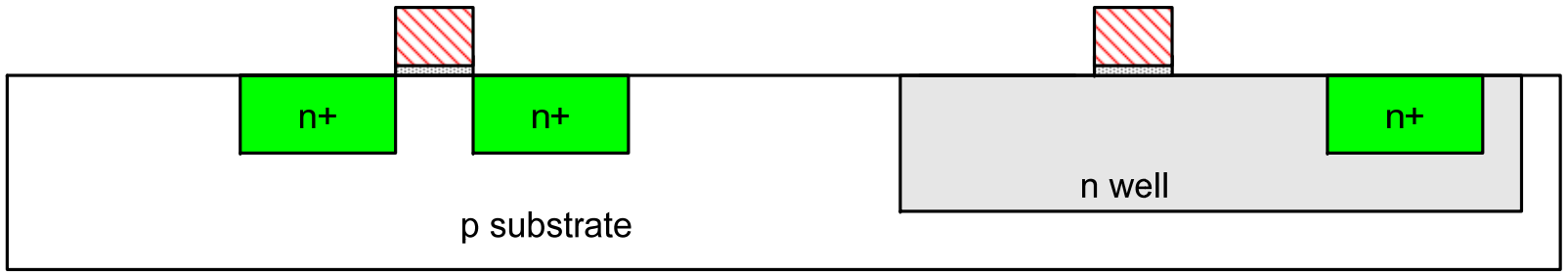
N-diffusion cont.

- Historically dopants were diffused
- Usually ion implantation today
- But regions are still called diffusion



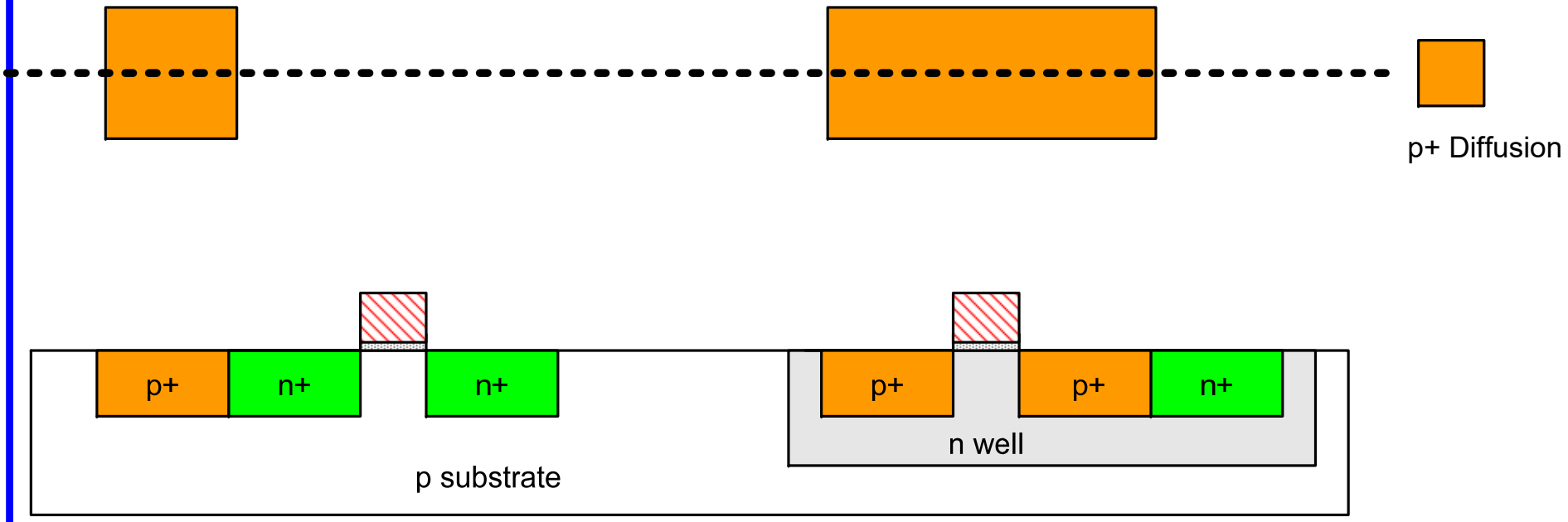
N-diffusion cont.

- Strip off oxide to complete patterning step



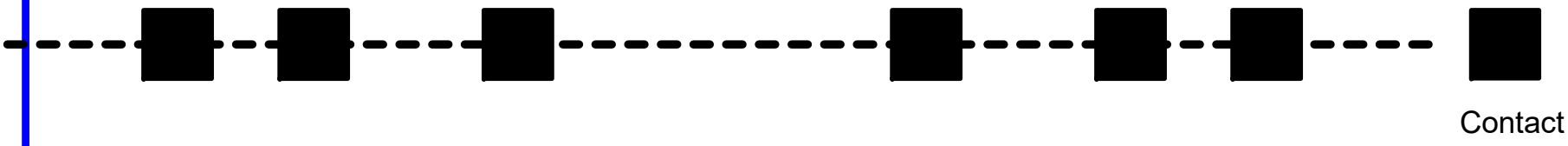
P-Diffusion

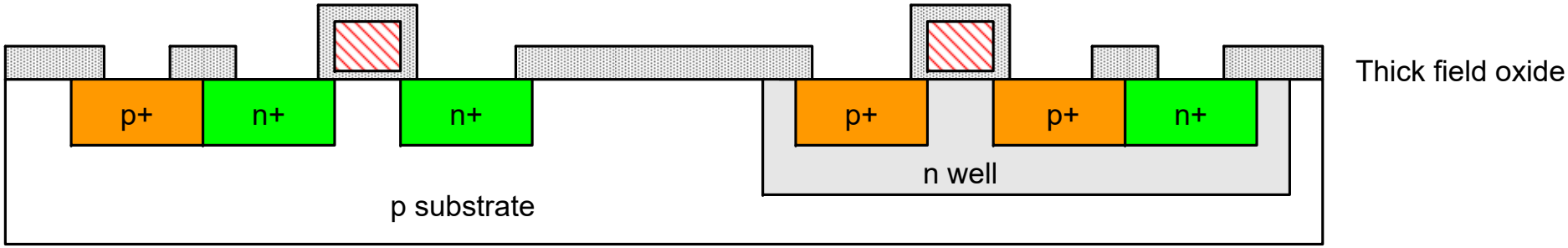
- Similar set of steps form p+ diffusion regions for pMOS source and drain and substrate contact



Contacts

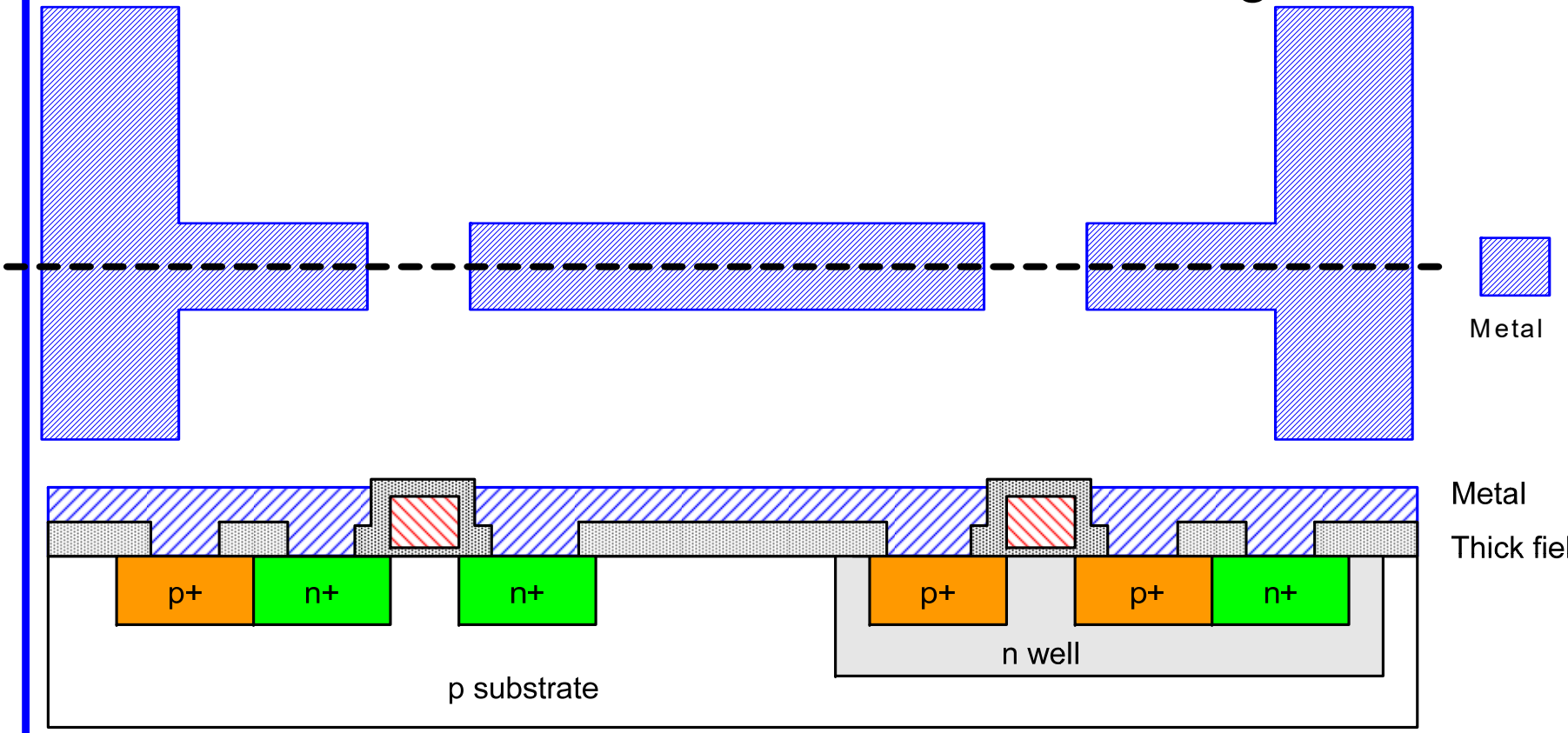
- Now we need to wire together the devices
- Cover chip with thick field oxide
- Etch oxide where contact cuts are needed





Metalization

- Sputter on aluminum over whole wafer
- Pattern to remove excess metal, leaving wires

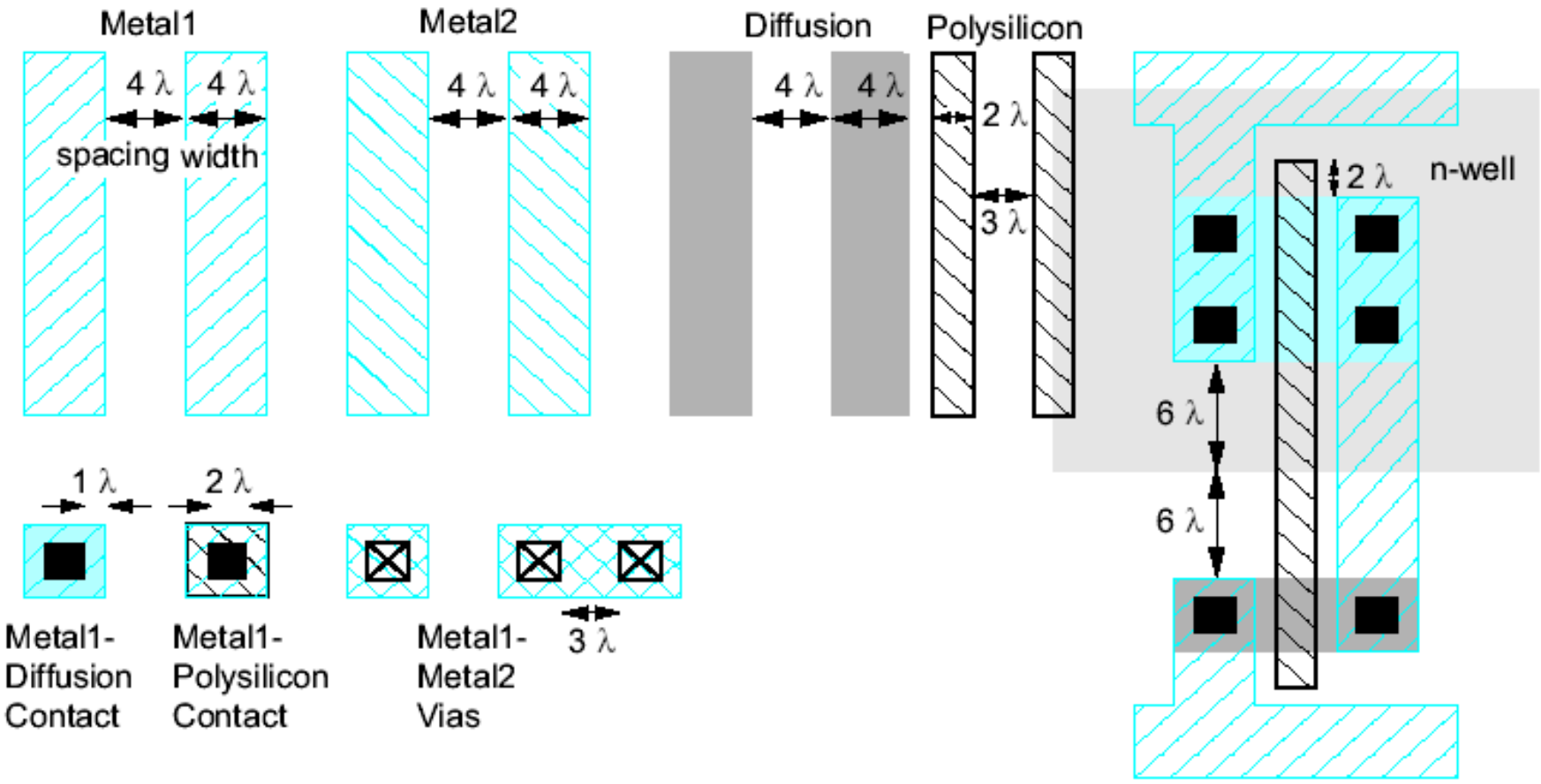


Layout

- Chips are specified with set of masks
- Minimum dimensions of masks determine transistor size (and hence speed, cost, and power)
- Feature size *f* = distance between source and drain
  - Set by minimum width of polysilicon
- Feature size improves 30% every 3 years or so
- Normalize for feature size when describing design rules
- Express rules in terms of $\lambda = f/2$
  - E.g. λ = 0.3 μm in 0.6 μm process

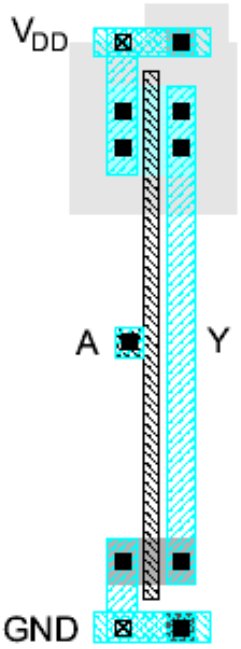
Simplified Design Rules

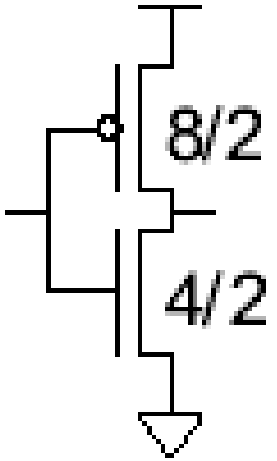
- Conservative rules to get you started

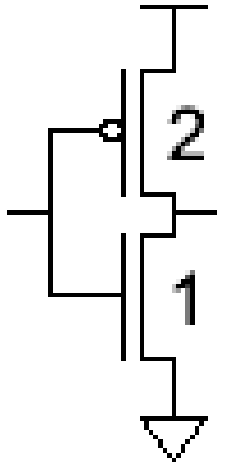


Inverter Layout

- Transistor dimensions specified as Width / Length
  - Minimum size is 4λ / 2λ, sometimes called 1 unit
  - In *f* = 0.6 μm process, this is 1.2 μm wide, 0.6 μm long



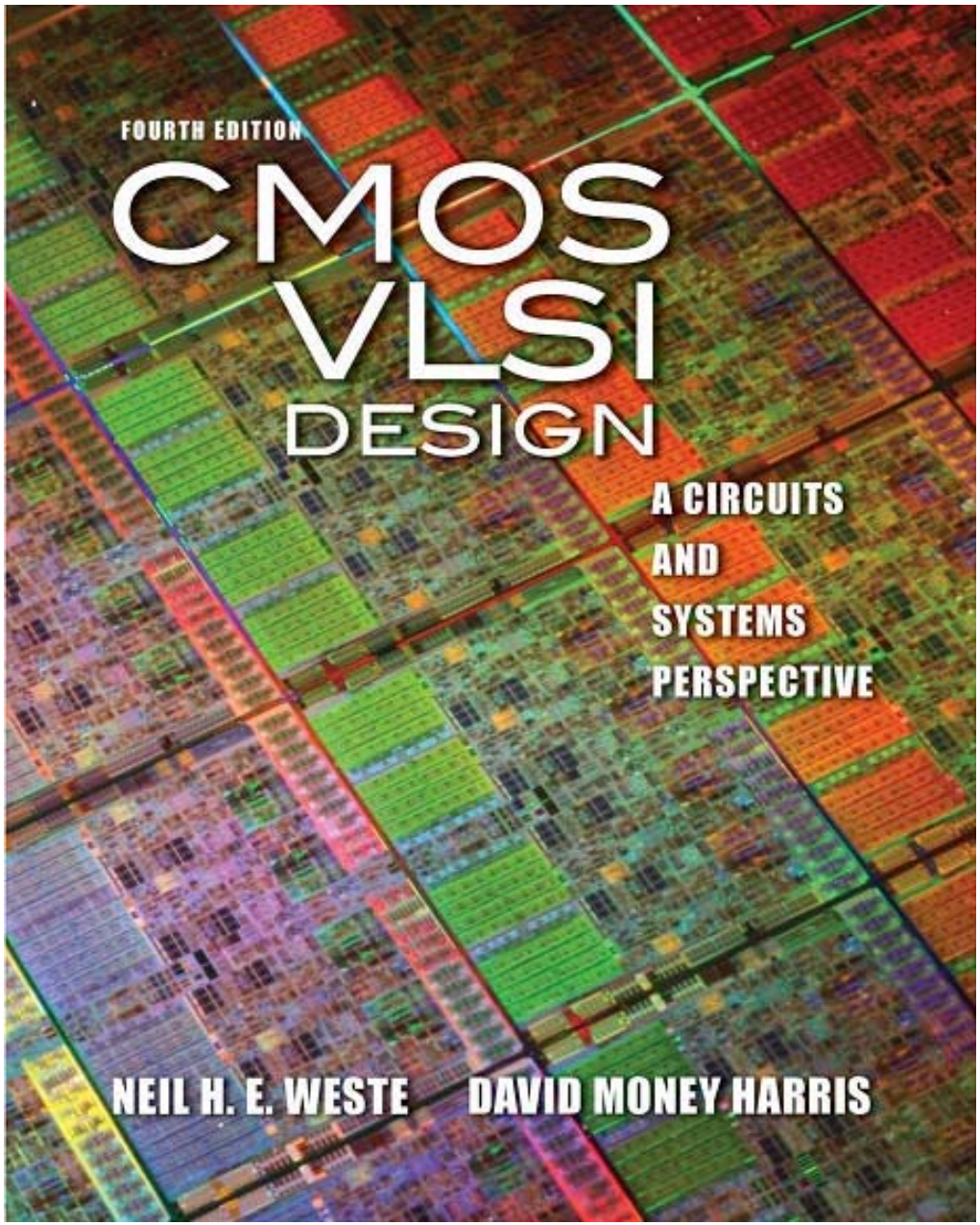




Summary

- MOS transistors are stacks of gate, oxide, silicon
- Act as electrically controlled switches
- Build logic gates out of switches
- Draw masks to specify layout of transistors

- Now you know everything necessary to start designing schematics and layout for a simple chip!



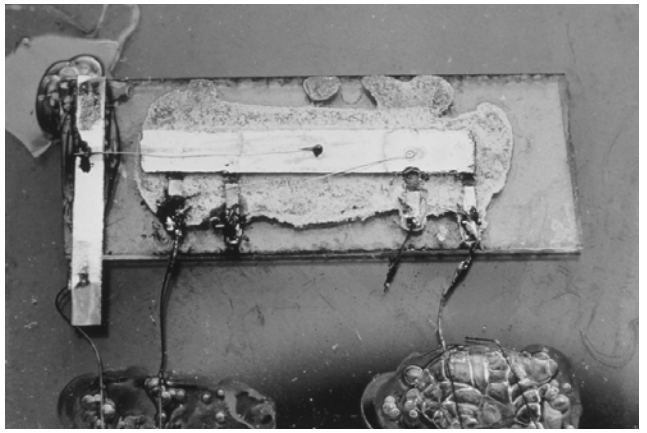
Lecture_3: Circuits & Layout

Outline

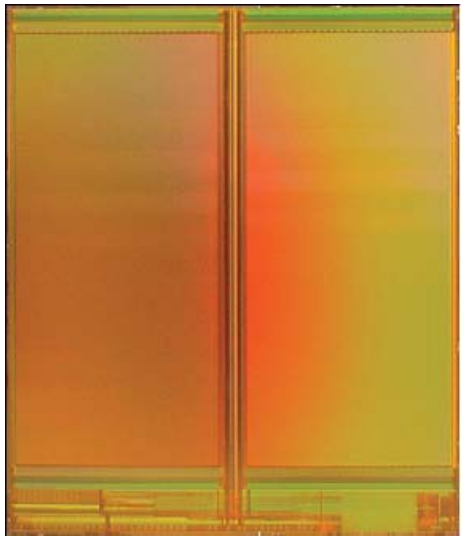
- A Brief History
- CMOS Gate Design
- Pass Transistors
- CMOS Latches & Flip-Flops
- Standard Cell Layouts
- Stick Diagrams

A Brief History

- 1958: First integrated circuit
  - Flip-flop using two transistors
  - Built by Jack Kilby at Texas Instruments
- 2010
  - Intel Core i7 μprocessor
    - 2.3 billion transistors
  - 64 Gb Flash memory
    - > 16 billion transistors



Courtesy Texas Instruments



[Trinh09]

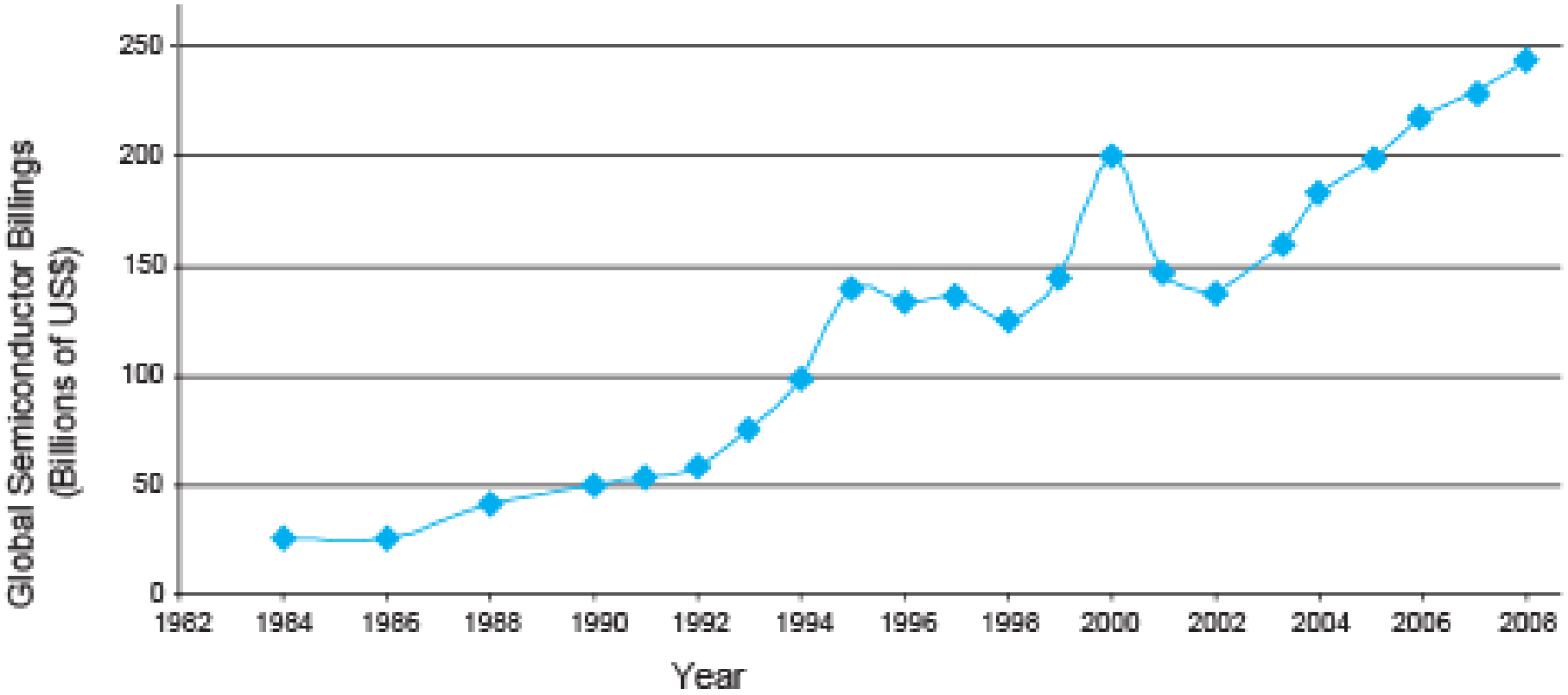
Growth Rate

- 53% compound annual growth rate over 50 years
  - No other technology has grown so fast so long
- Driven by miniaturization of transistors
  - Smaller is cheaper, faster, lower in power!
  - Revolutionary effects on society

[Moore65]
Electronics Magazine

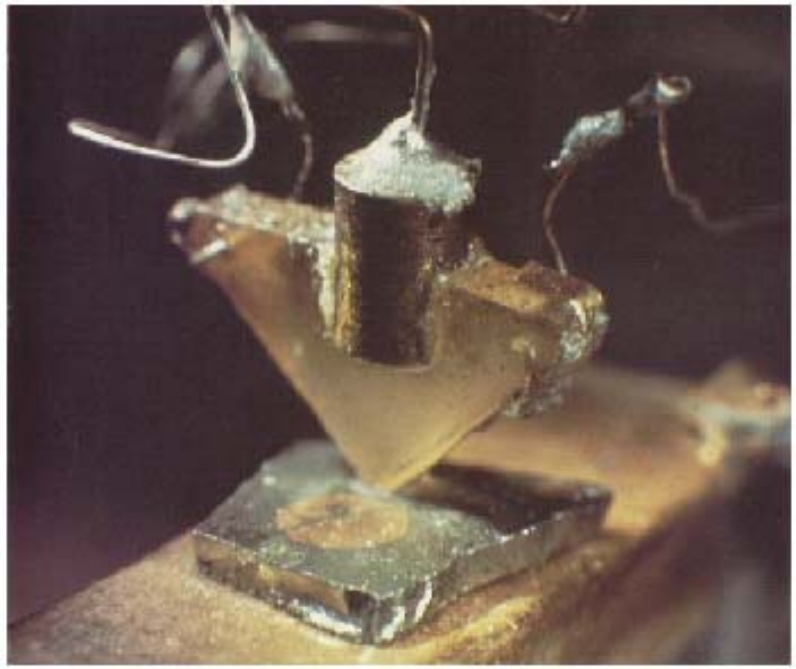
Annual Sales

- >10^{19} transistors manufactured in 2008
  - 1 billion for every human on the planet



Invention of the Transistor

- Vacuum tubes ruled in first half of 20th century
  Large, expensive, power-hungry, unreliable
- 1947: first point contact transistor
  - John Bardeen and Walter Brattain at Bell Labs
  - See *Crystal Fire*
    by Riordan, Hoddeson



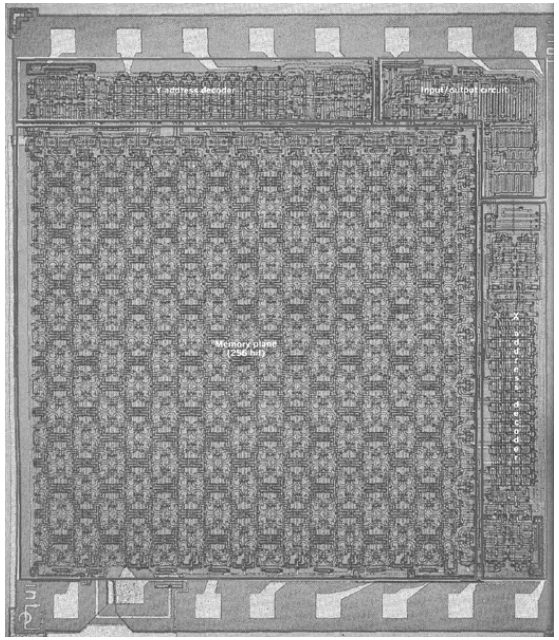
AT&T Archives.
Reprinted with
permission.

Transistor Types

- Bipolar transistors
  - npn or pnp silicon structure
  - Small current into very thin base layer controls large currents between emitter and collector
  - Base currents limit integration density
- Metal Oxide Semiconductor Field Effect Transistors
  - nMOS and pMOS MOSFETS
  - Voltage applied to insulated gate controls current between source and drain
  - Low power allows very high integration

MOS Integrated Circuits

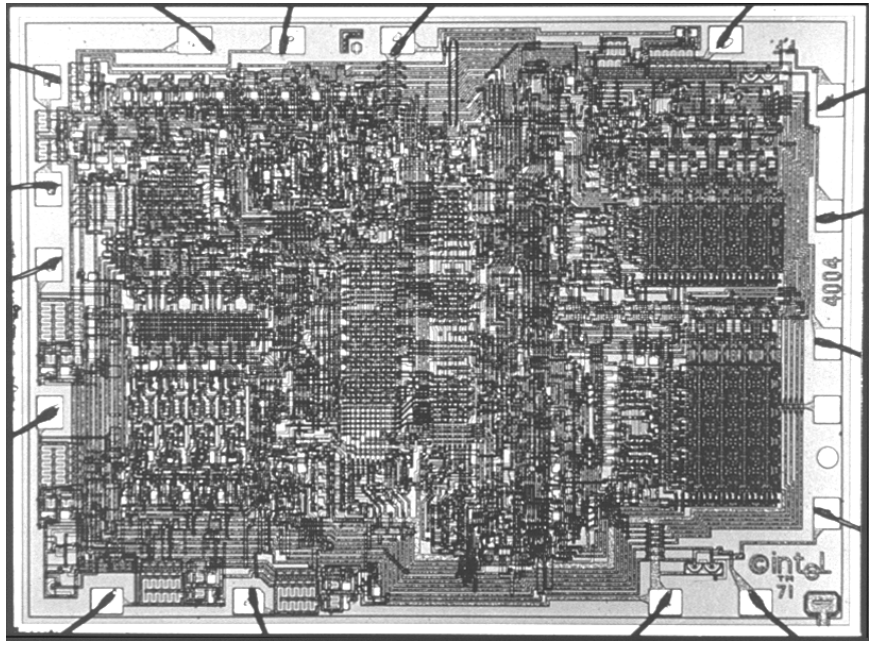
- 1970's processes usually had only nMOS transistors
  - Inexpensive, but consume power while idle



[Vadasz69]

© 1969 IEEE.

Intel 1101 256-bit SRAM



Intel Museum.

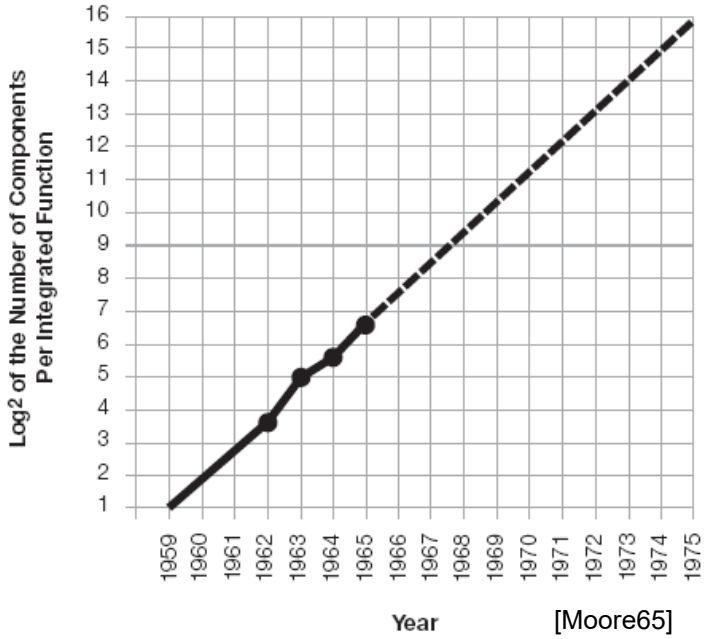
Reprinted with permission.

Intel 4004 4-bit μProc

- 1980s-present: CMOS processes for low idle power

Moore's Law: Then

- 1965: Gordon Moore plotted transistor on each chip
  - Fit straight line on semilog scale
  - Transistor counts have doubled every 26 months



[Moore65]

Electronics Magazine

Integration Levels

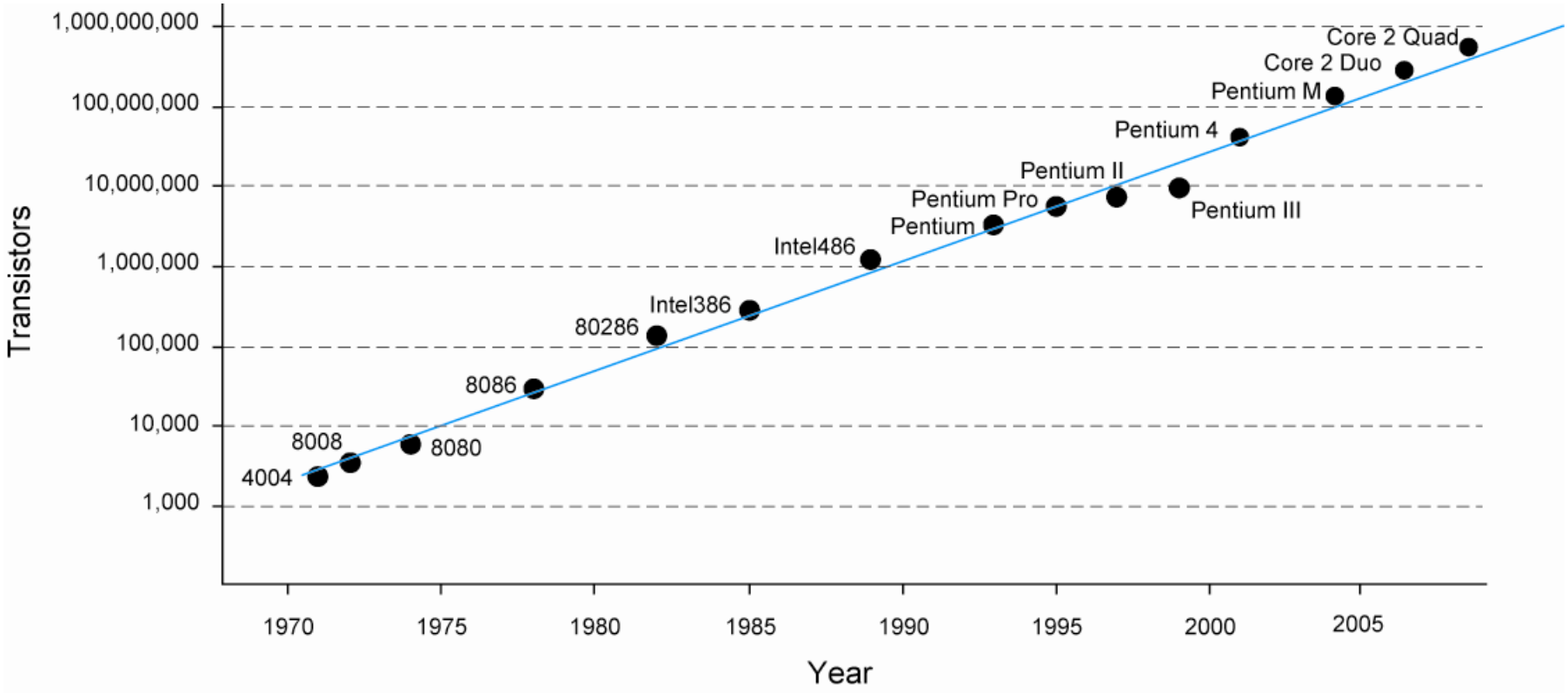
SSI: 10 gates

MSI: 1000 gates

LSI: 10,000 gates

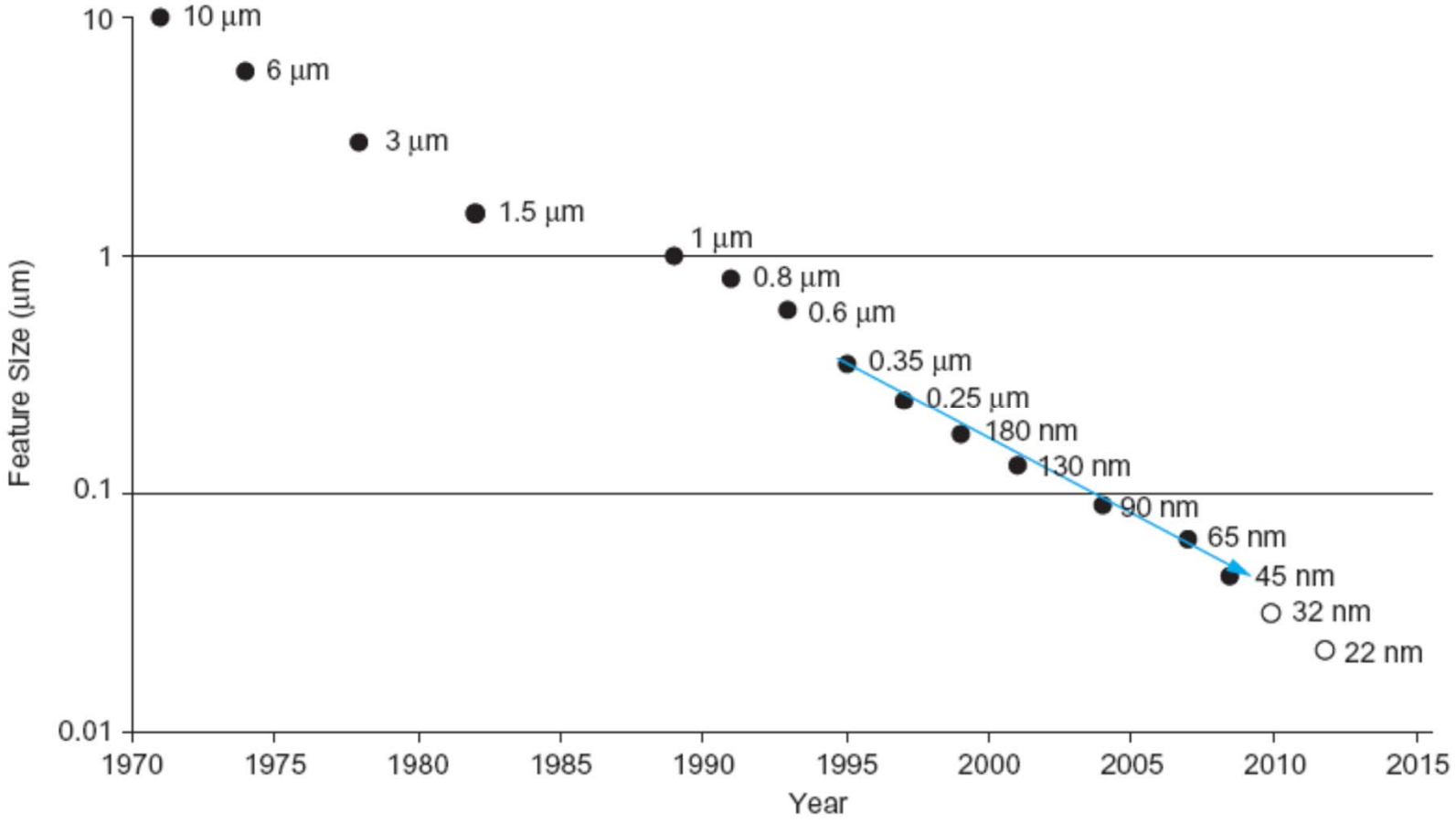
VLSI: > 10k gates

And Now...


1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
Transistors
1970
1975
1980
1985
1990
1995
2000
2005
Year
4004
8008
8080
8086
80286
Intel386
Intel486
Pentium
Pentium Pro
Pentium II
Pentium III
Pentium 4
Pentium M
Core 2 Duo
Core 2 Quad

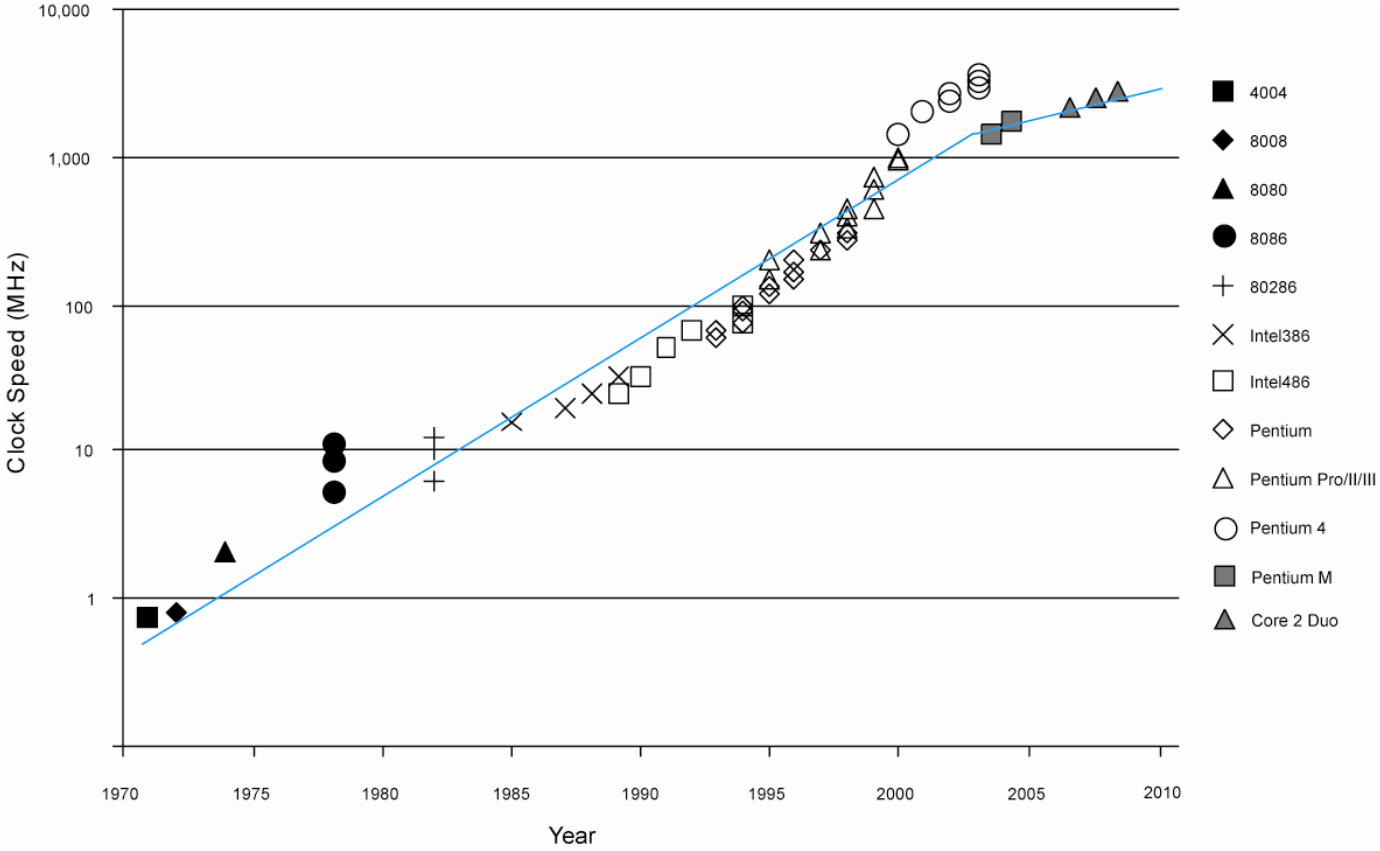
Feature Size

- Minimum feature size shrinking 30% every 2-3 years



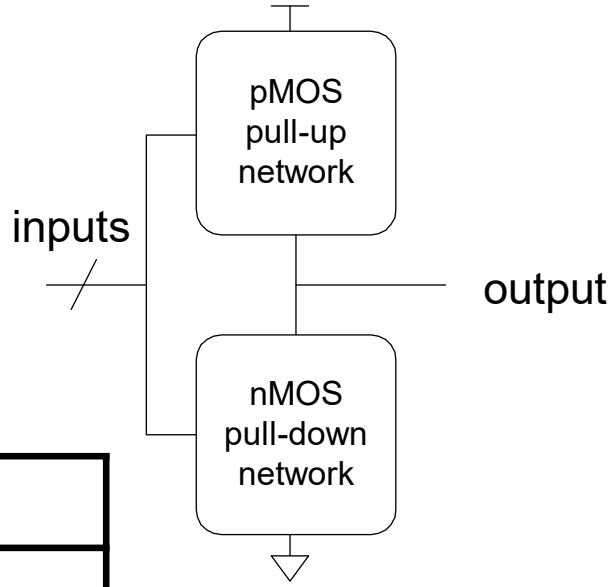
Corollaries

- Many other factors grow exponentially
  - Ex: clock frequency, processor performance



Complementary CMOS

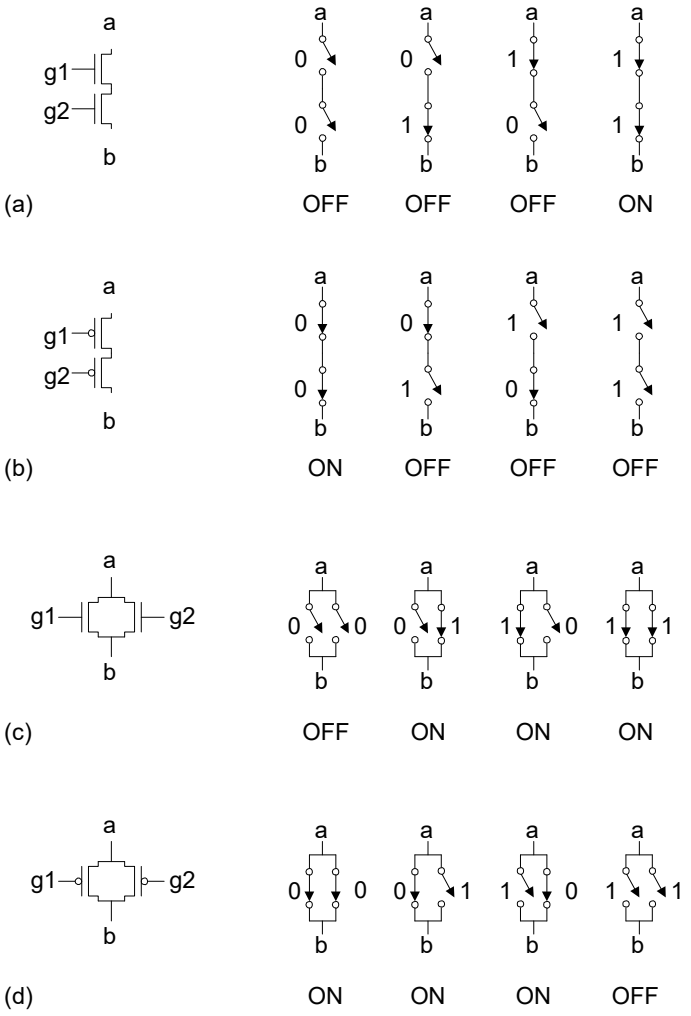
- Complementary CMOS logic gates
  - nMOS *pull-down network*
  - pMOS *pull-up network*
  - a.k.a. static CMOS



	Pull-up OFF	Pull-up ON
Pull-down OFF	Z (float)	1
Pull-down ON	0	X (crowbar)

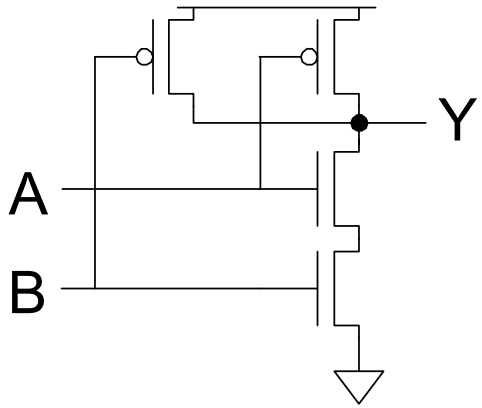
Series and Parallel

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



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

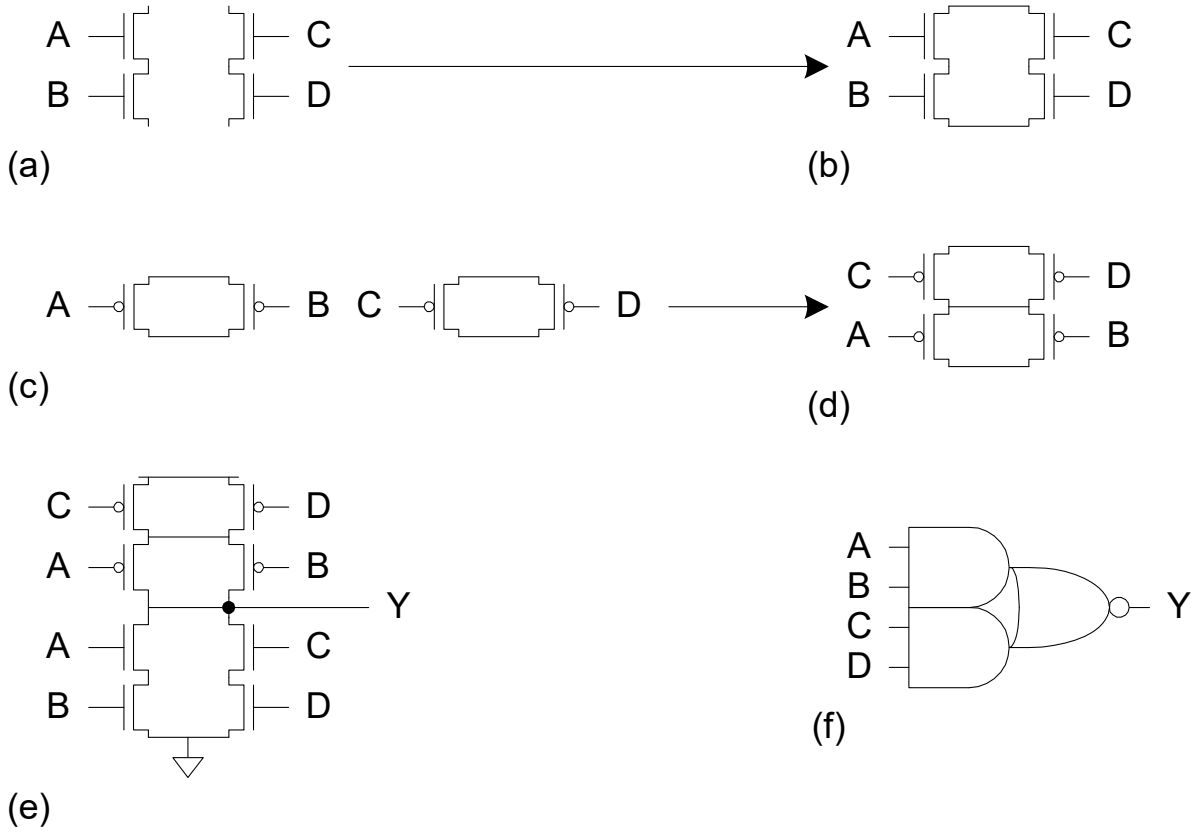


- Rule of *Conduction Complements*
  - Pull-up network is a complement of pull-down
  - Parallel -> series, series -> parallel

Compound Gates

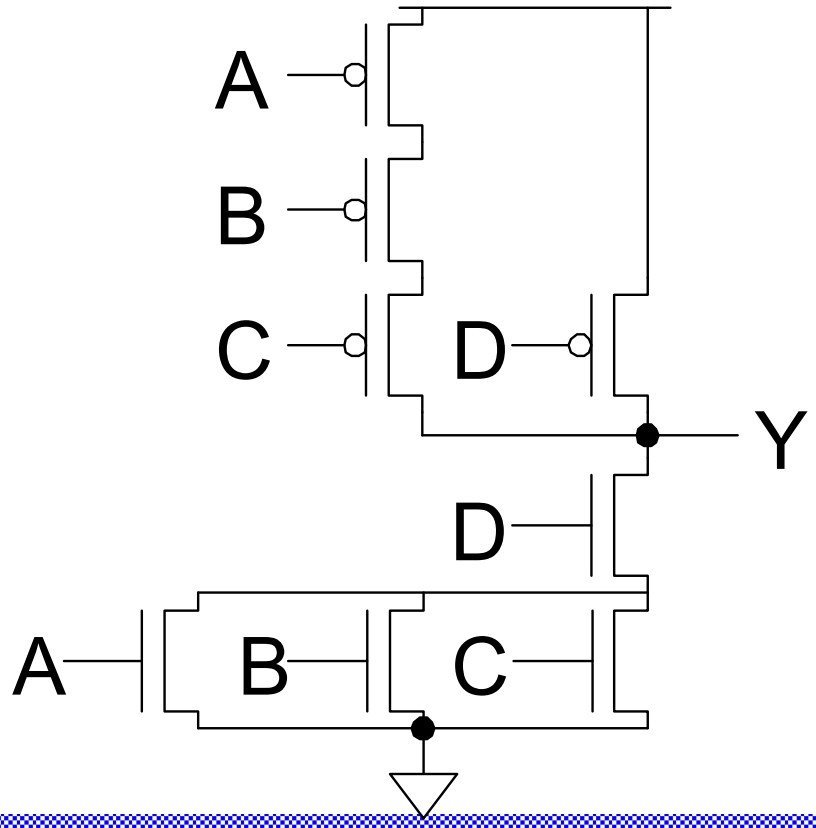
- *Compound gates* can do any inverting function

$$Y = \overline{A.B + C.D} \quad (AND - AND - OR - INVERT, AOI22)$$



Example: O3AI

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

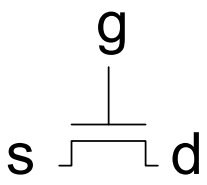


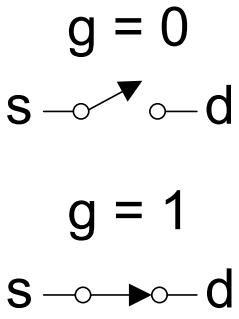
Signal Strength

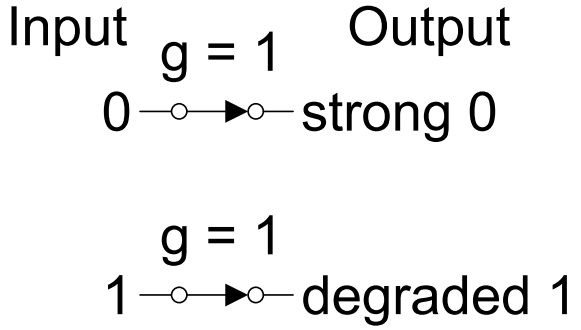
- *Strength* of signal
  - How close it approximates ideal voltage source
- V_{DD} and GND rails are strongest 1 and 0
- nMOS pass strong 0
  - But degraded or weak 1
- pMOS pass strong 1
  - But degraded or weak 0
- Thus, nMOS are best for pull-down network

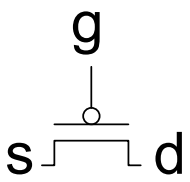
Pass Transistors

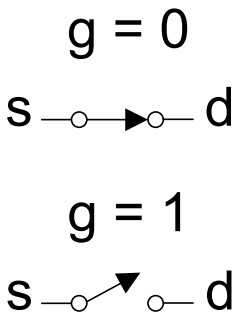
- Transistors can be used as switches

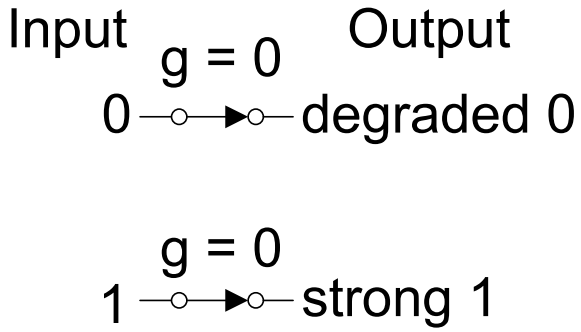






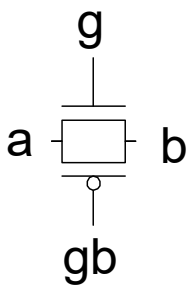


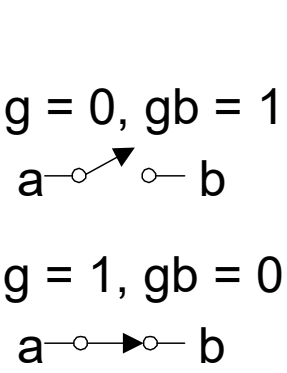


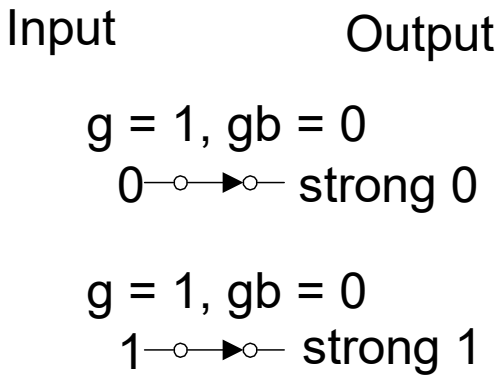


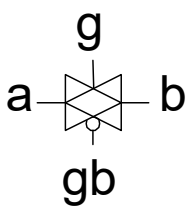
Transmission Gates

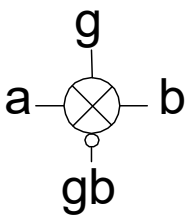
- Pass transistors produce degraded outputs
- *Transmission gates* pass both 0 and 1 well

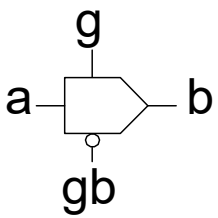








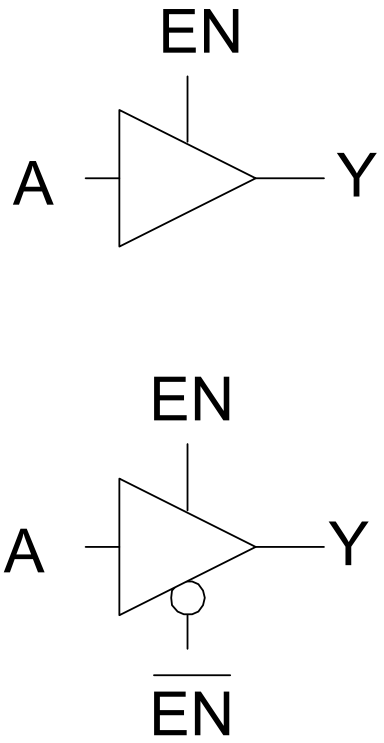




Tristates

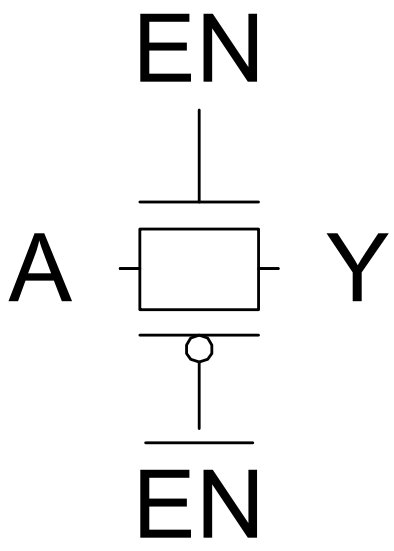
- *Tristate buffer* produces Z when not enabled

EN	A	Y
0	0	
0	1	
1	0	
1	1	



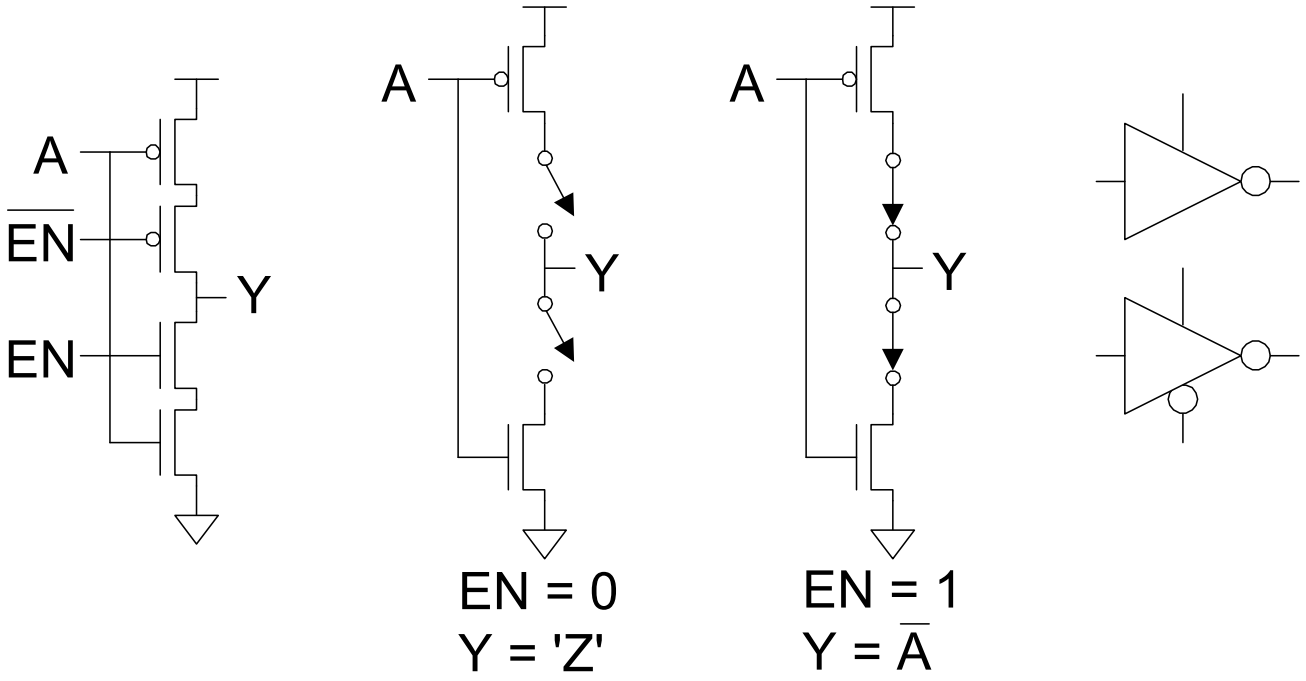
Nonrestoring Tristate

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



Tristate Inverter

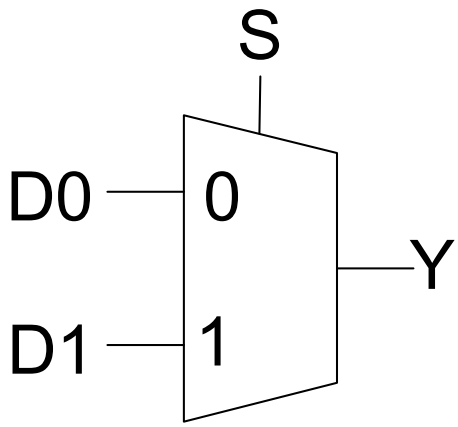
- Tristate inverter produces restored output
  - Violates conduction complement rule
  - Because we want a Z output



Multiplexers

- 2:1 multiplexer chooses between two inputs

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

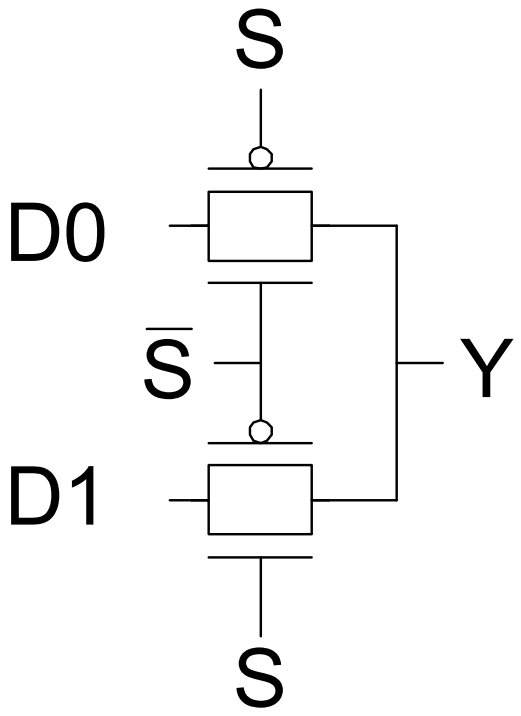


Gate-Level Mux Design

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

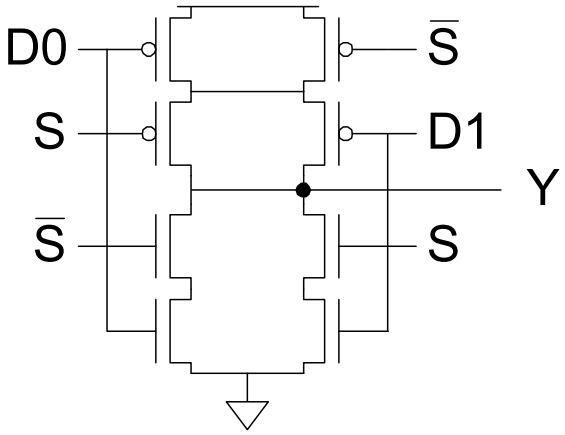
Transmission Gate Mux

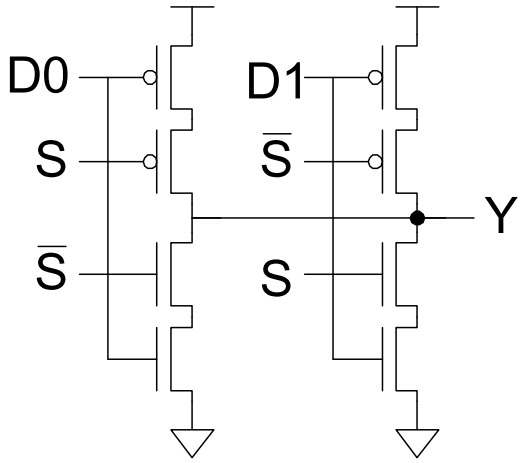
- Nonrestoring mux uses two transmission gates
  - Only 4 transistors

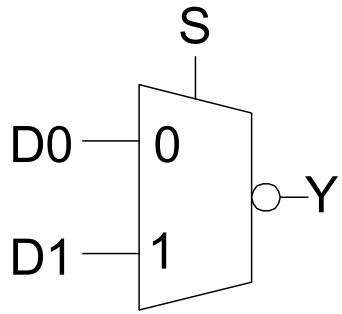


Inverting Mux

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

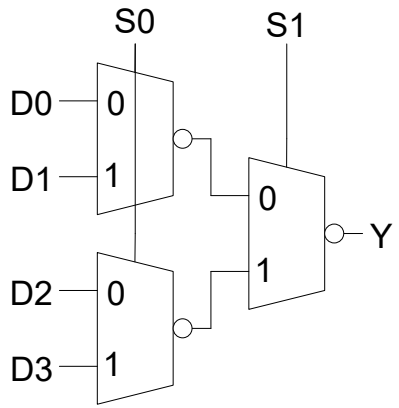


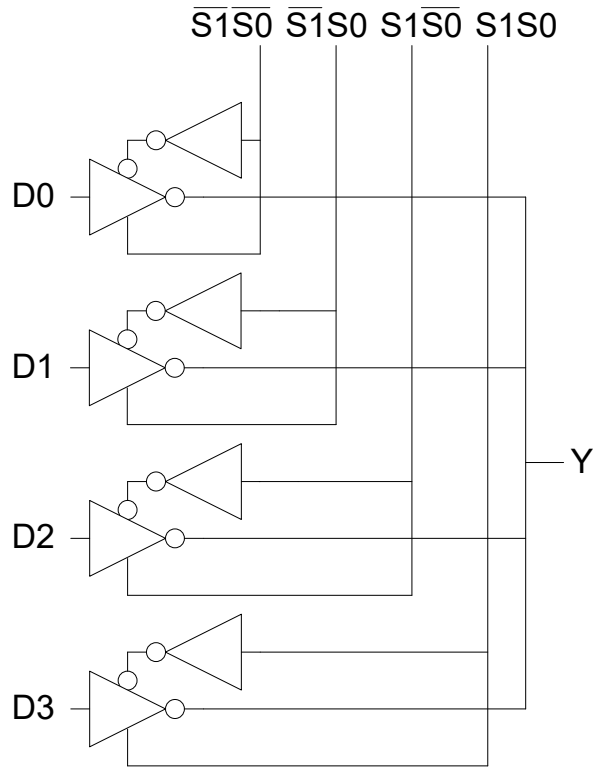


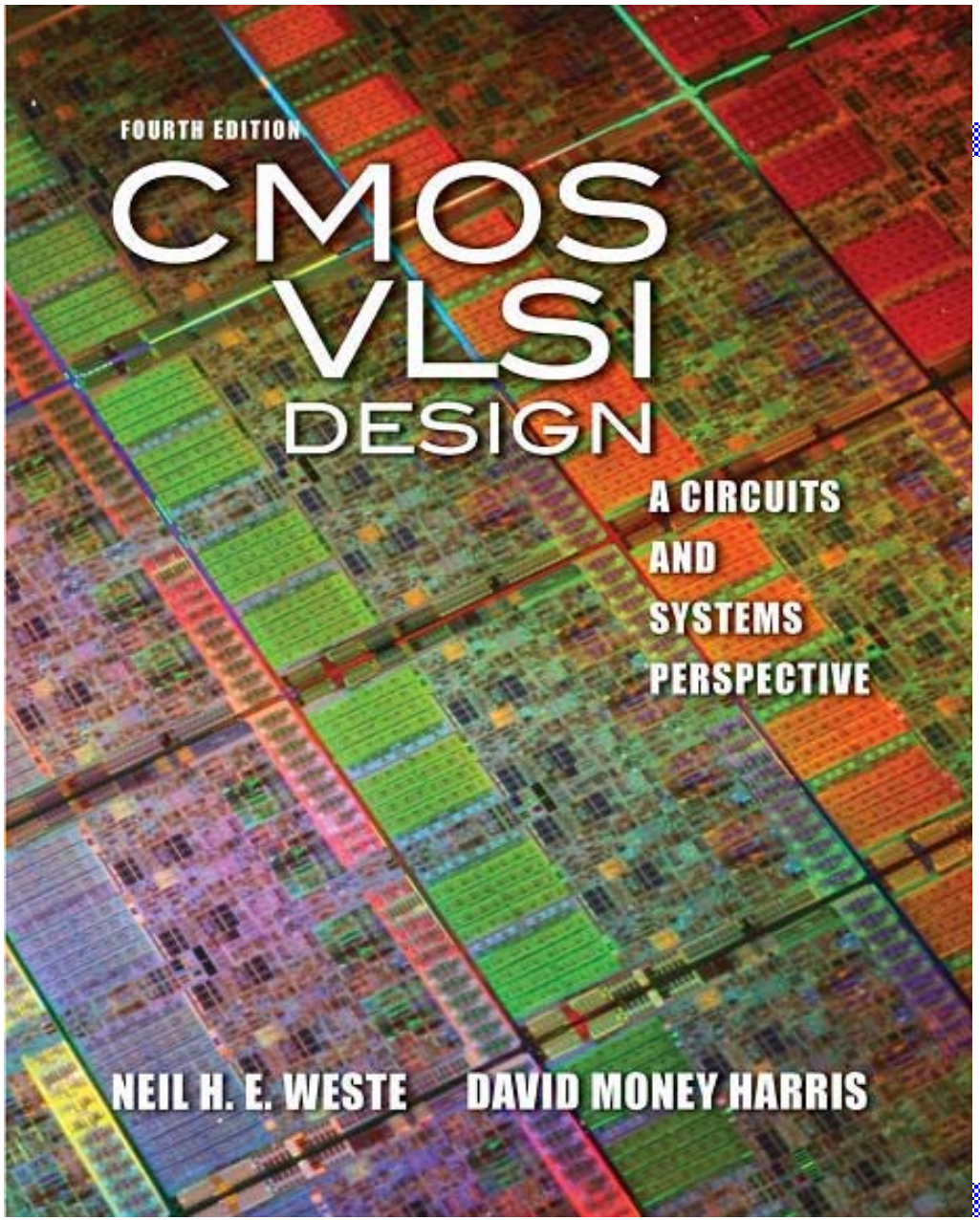


4:1 Multiplexer

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



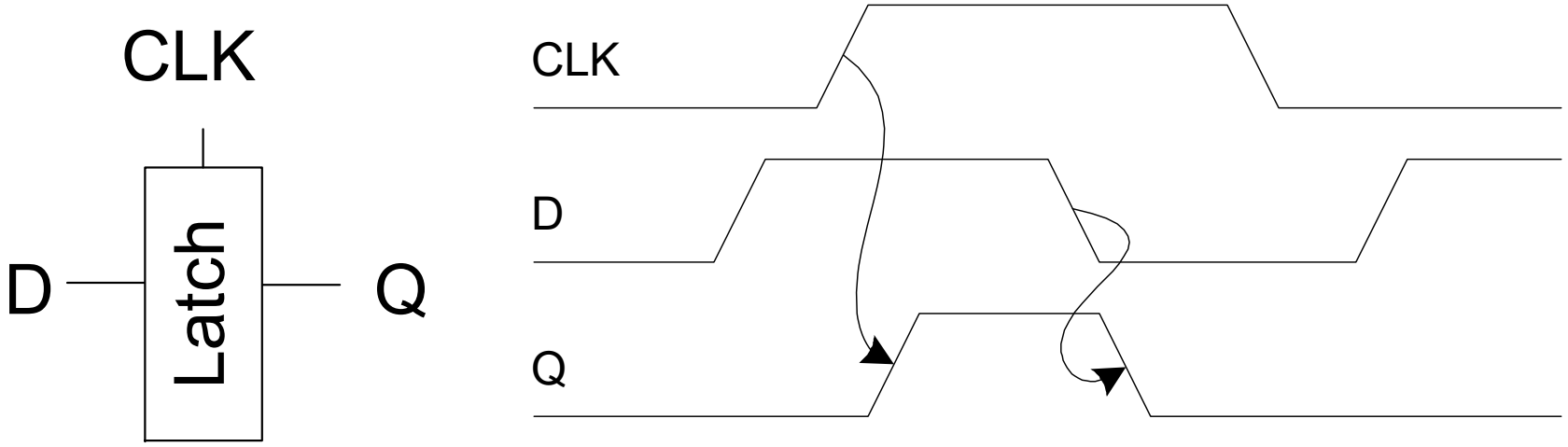




Lecture_4 Circuits & Layout

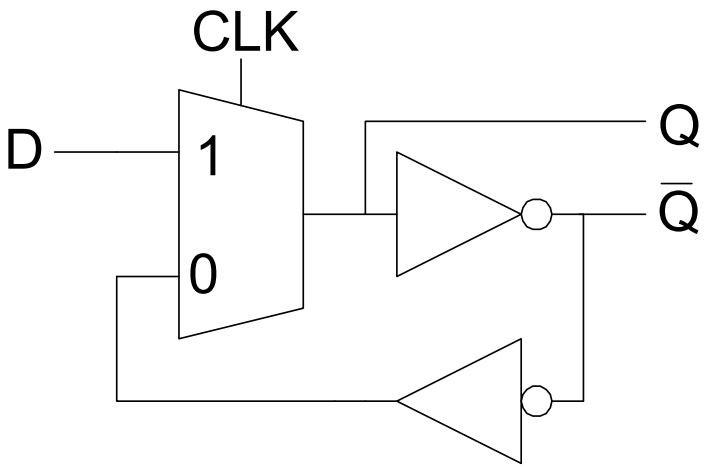
D Latch

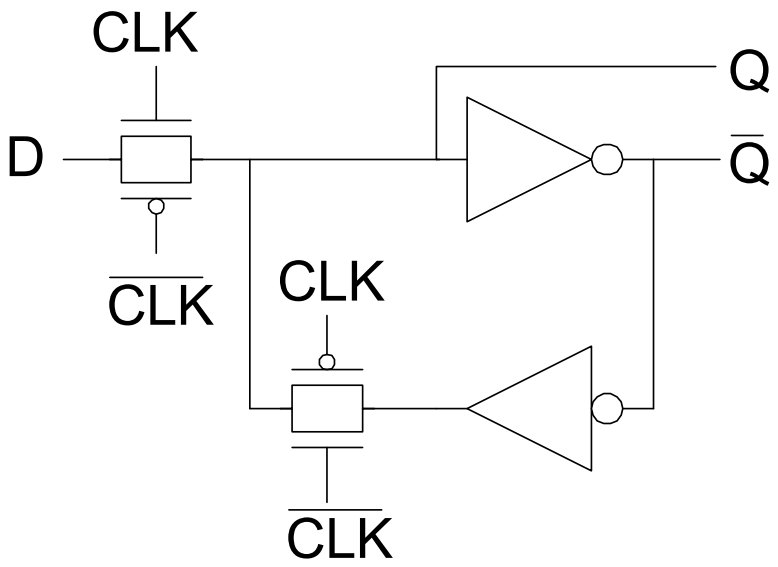
- When CLK = 1, latch is *transparent*
  - D flows through to Q like a buffer
- When CLK = 0, the latch is *opaque*
  - Q holds its old value independent of D
- a.k.a. *transparent latch* or *level-sensitive latch*



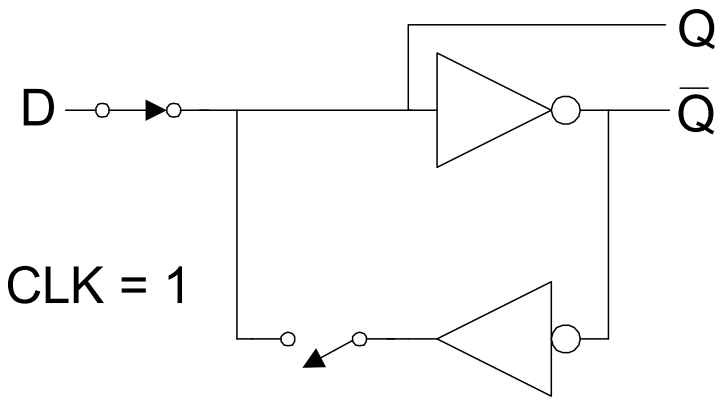
D Latch Design

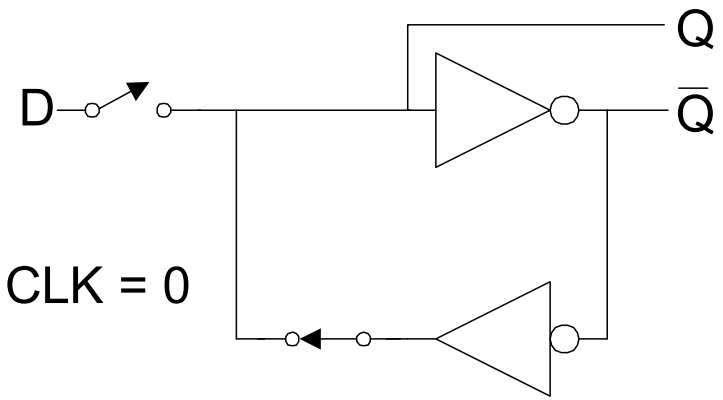
- Multiplexer chooses D or old Q

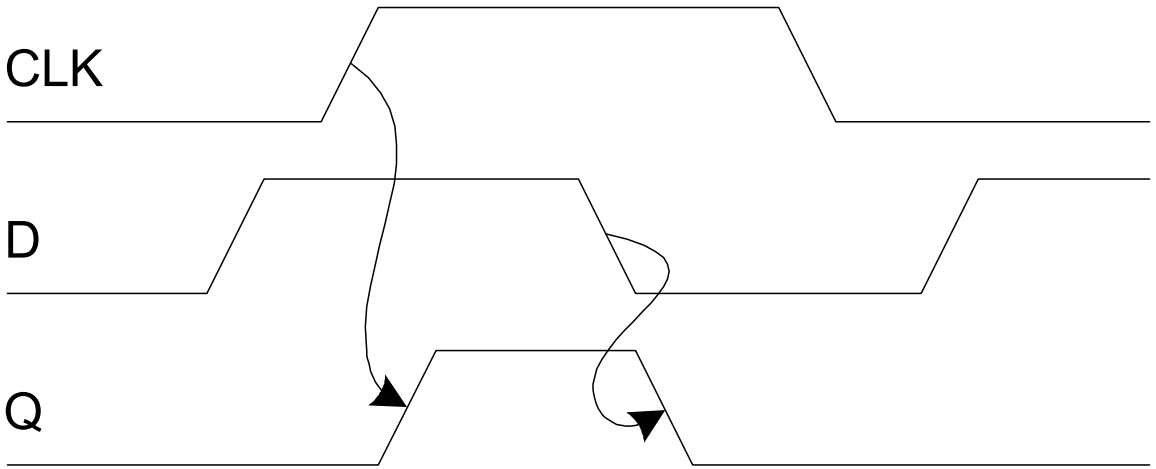




D Latch Operation

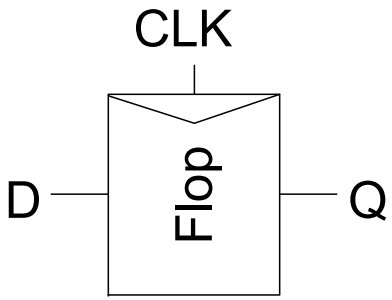


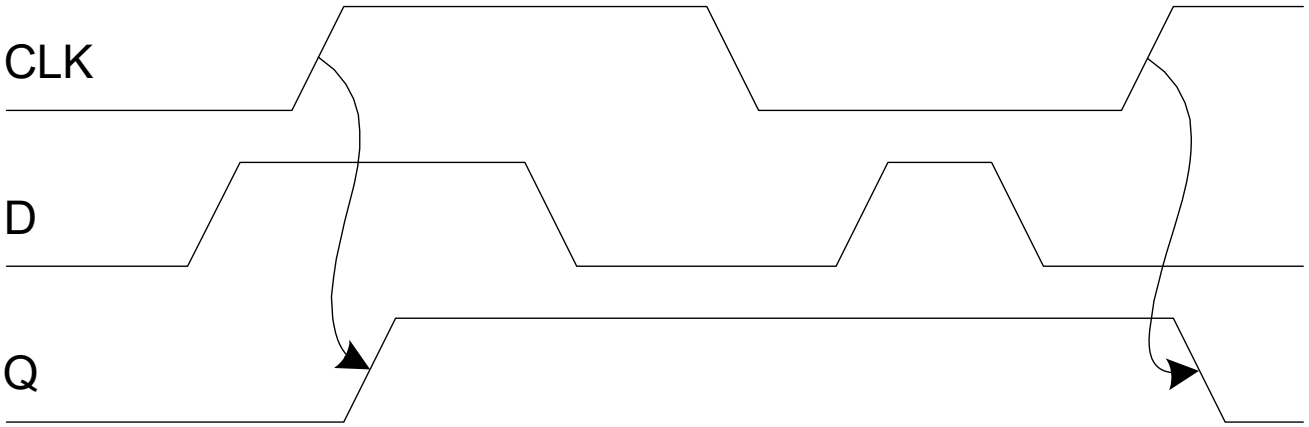




D Flip-flop

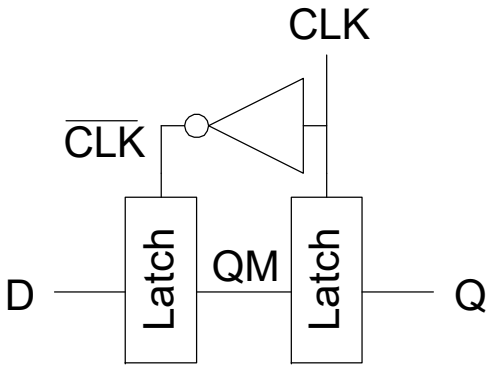
- When CLK rises, D is copied to Q
- At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop*, *master-slave flip-flop*

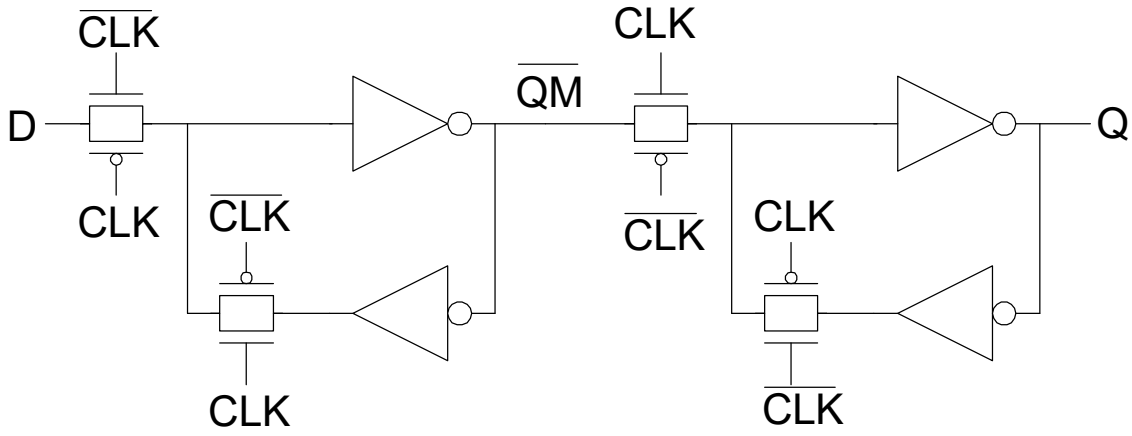




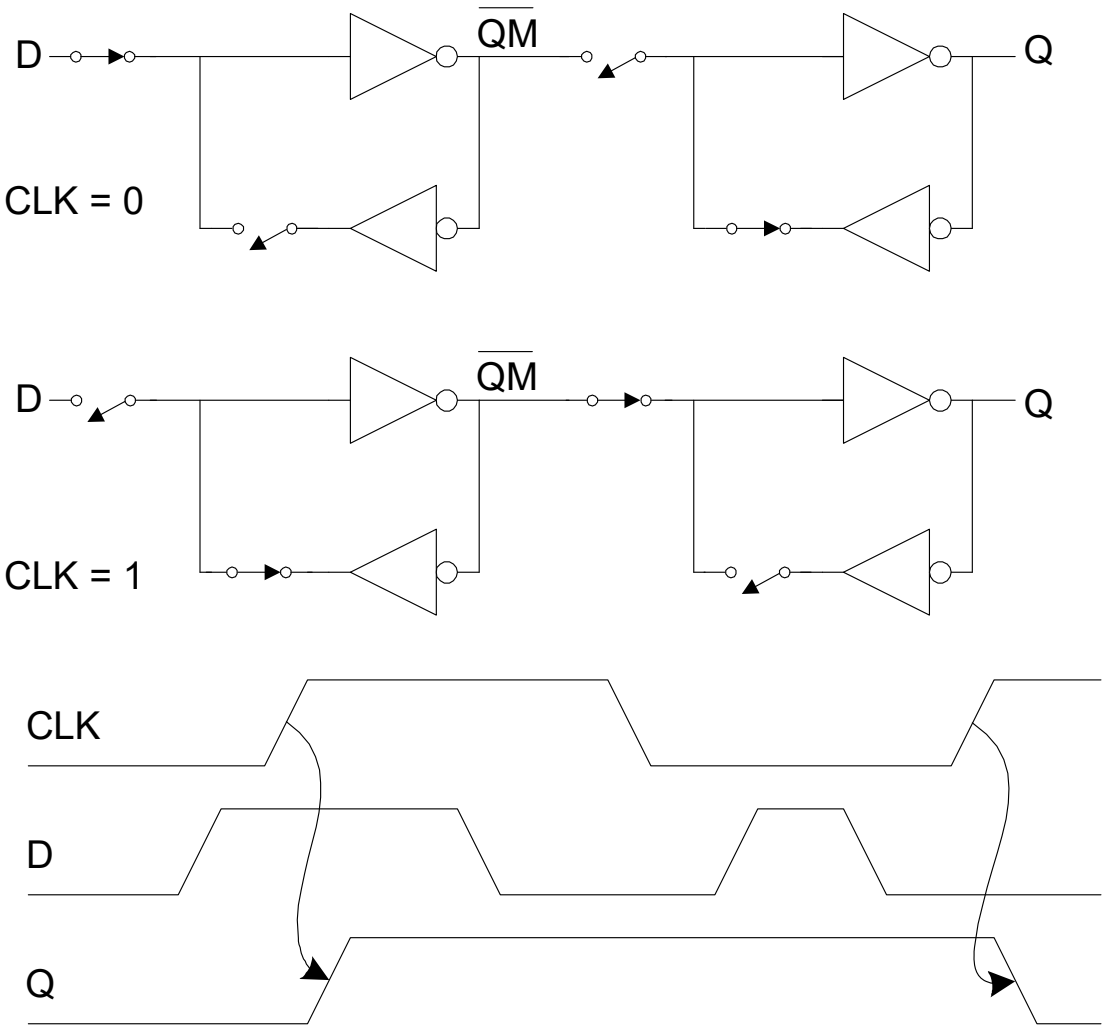
D Flip-flop Design

- Built from master and slave D latches



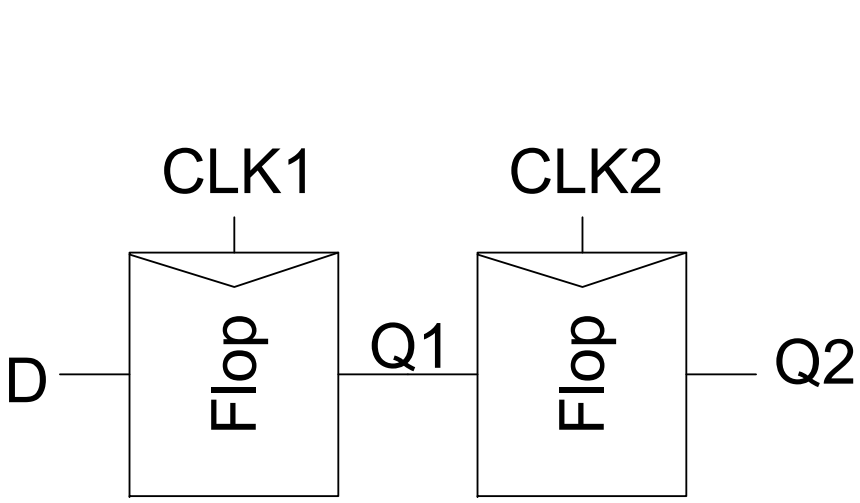


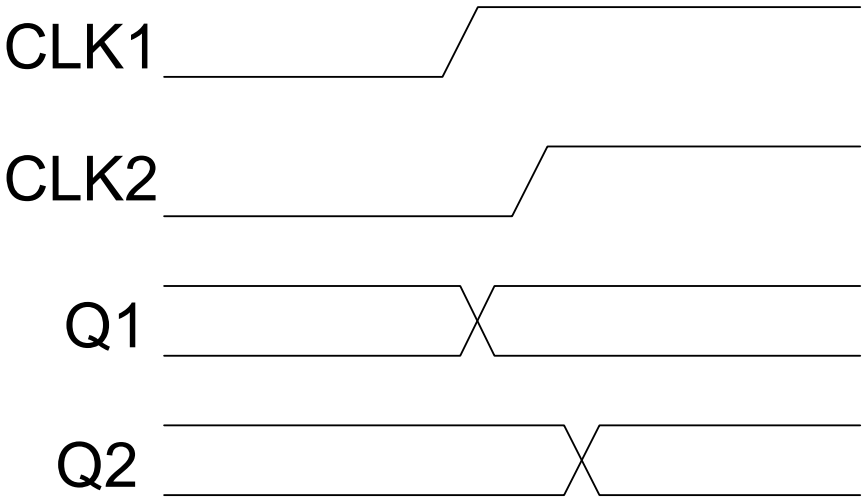
D Flip-flop Operation



Race Condition

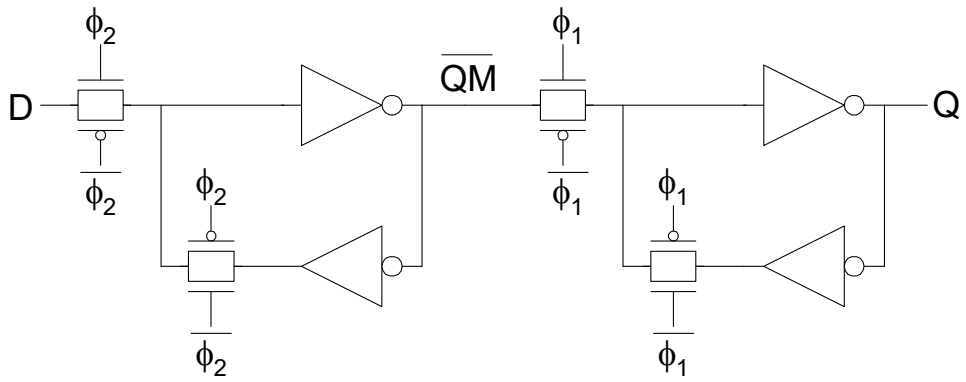
- Back-to-back flops can malfunction from clock skew
  - Second flip-flop fires late
  - Sees first flip-flop change and captures its result
  - Called *hold-time failure* or *race condition*





Nonoverlapping Clocks

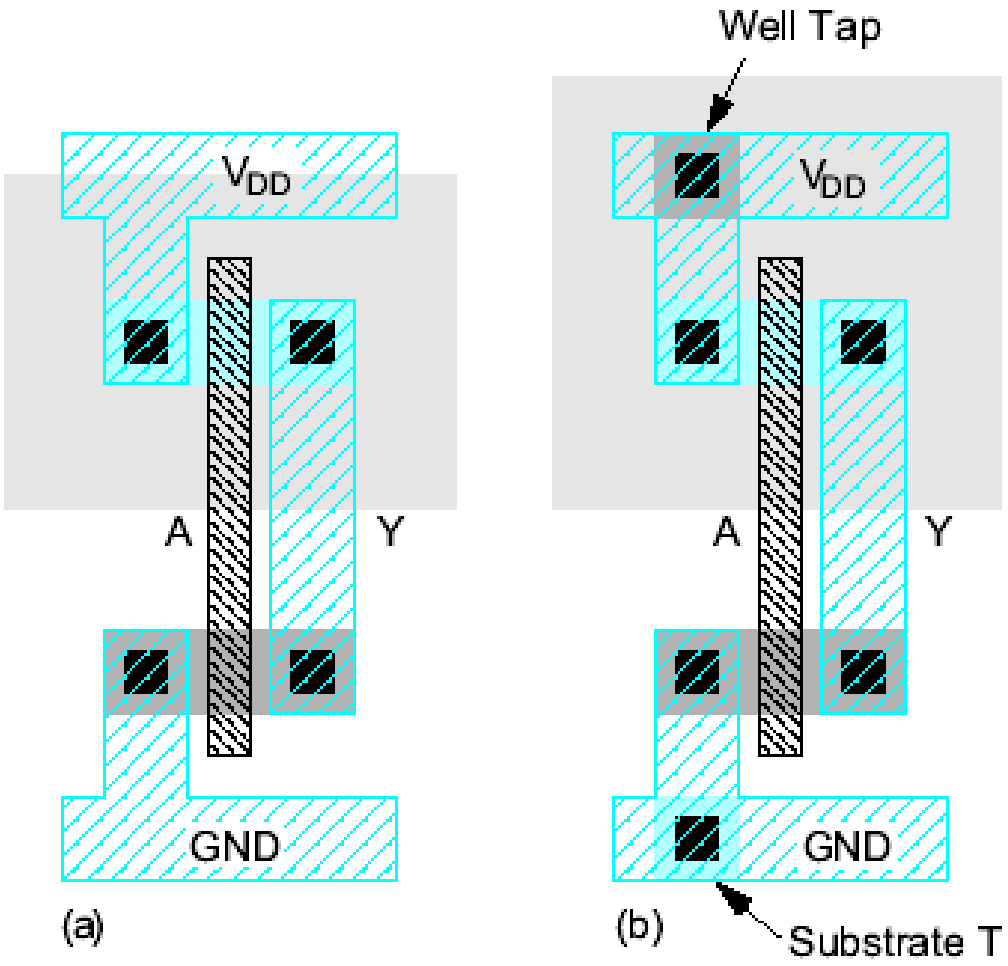
- Nonoverlapping clocks can prevent races
  - As long as nonoverlap exceeds clock skew
- We will use them in this class for safe design
  - Industry manages skew more carefully instead



Gate Layout

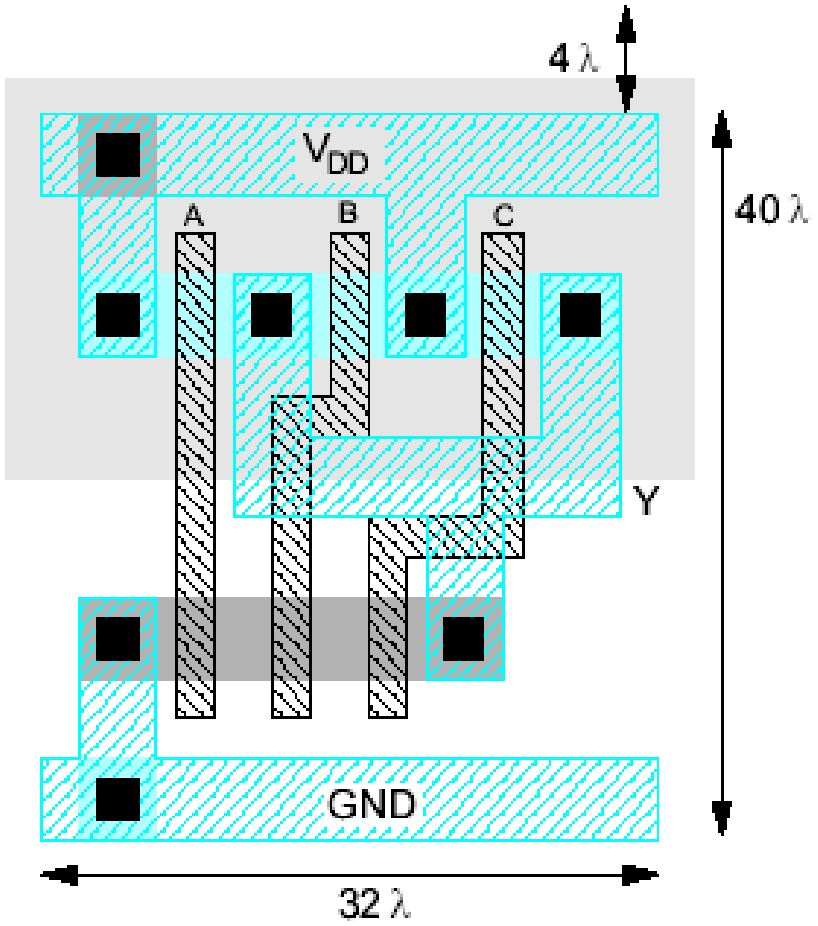
- Layout can be very time consuming
  - Design gates to fit together nicely
  - Build a library of standard cells
- Standard cell design methodology
  - V_{DD} and GND should abut (standard height)
  - Adjacent gates should satisfy design rules
  - nMOS at bottom and pMOS at top
  - All gates include well and substrate contacts

Example: Inverter



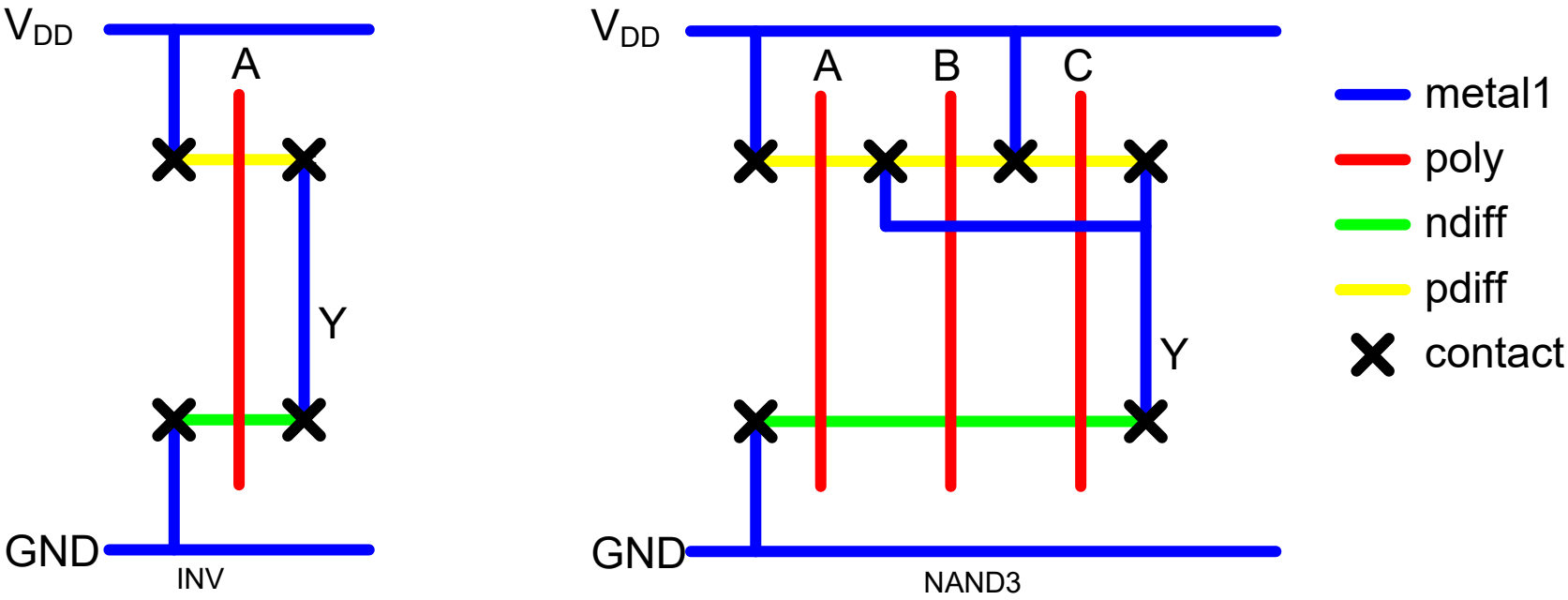
Example: NAND3

- Horizontal N-diffusion and p-diffusion strips
- Vertical polysilicon gates
- Metal1 V_{DD} rail at top
- Metal1 GND rail at bottom
- 32 λ by 40 λ



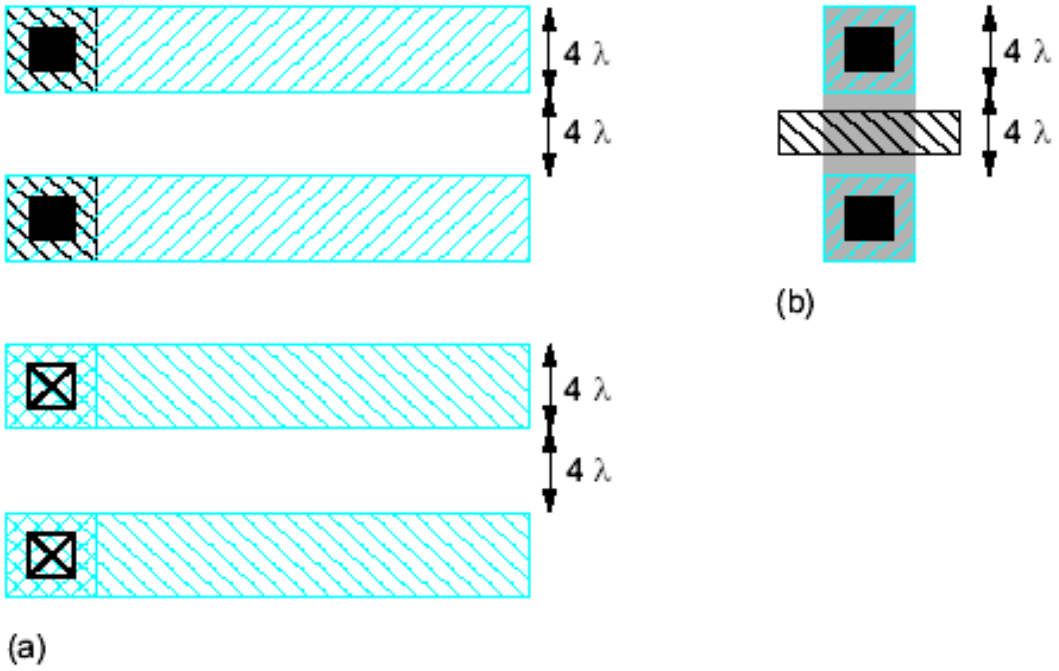
Stick Diagrams

- *Stick diagrams* help plan layout quickly
  - Need not be to scale
  - Draw with color pencils or dry-erase markers



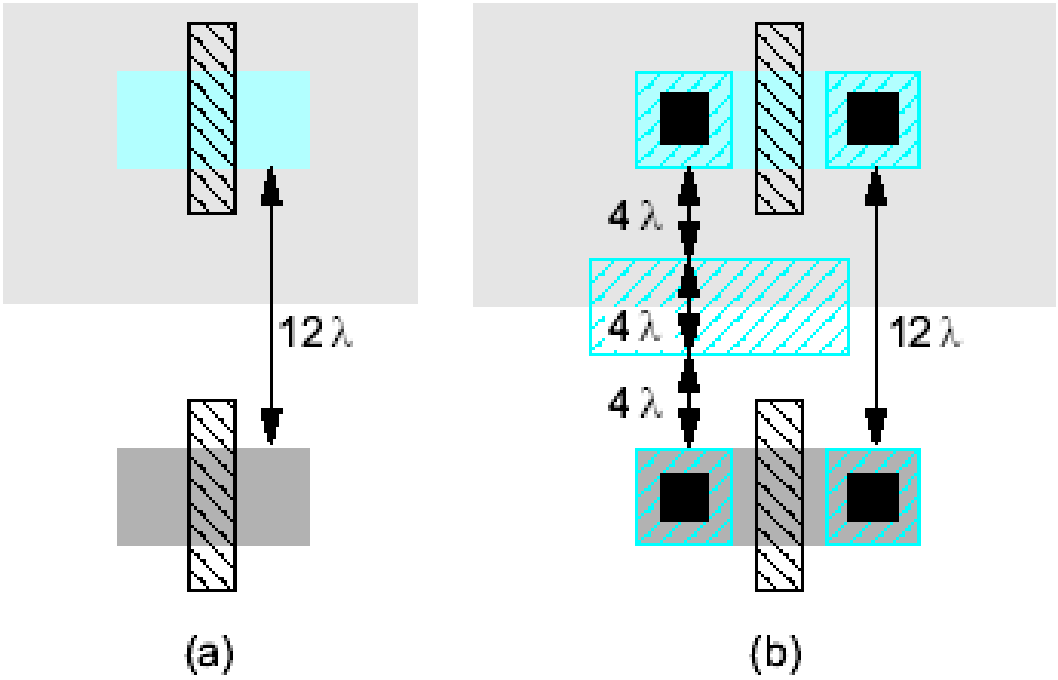
Wiring Tracks

- A *wiring track* is the space required for a wire
  - 4 λ width, 4 λ spacing from neighbor = 8 λ pitch
- Transistors also consume one wiring track



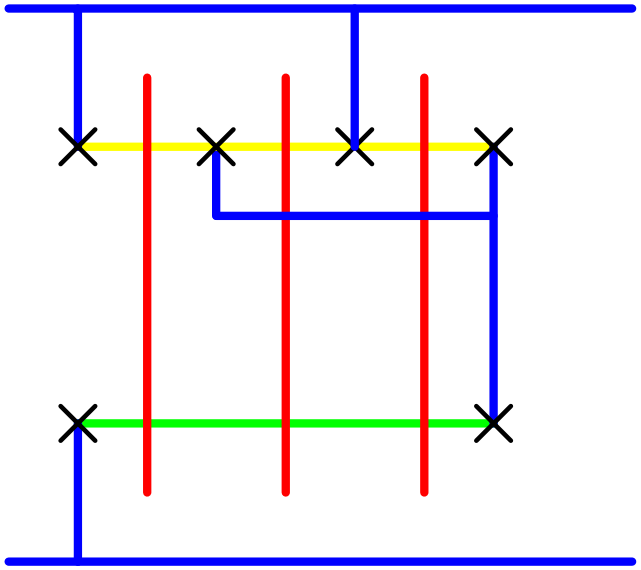
Well spacing

- Wells must surround transistors by 6 λ
  - Implies 12 λ between opposite transistor flavors
  - Leaves room for one wire track



Area Estimation

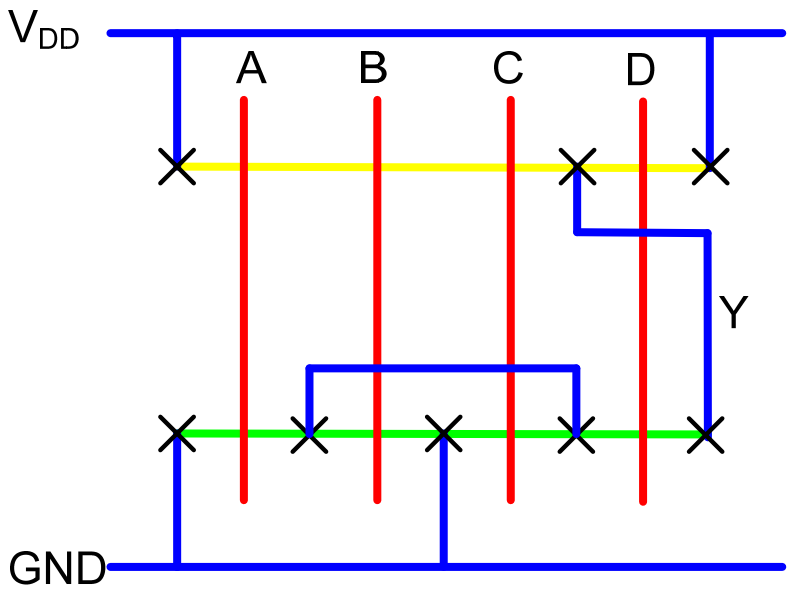
- Estimate area by counting wiring tracks
  - Multiply by 8 to express in λ



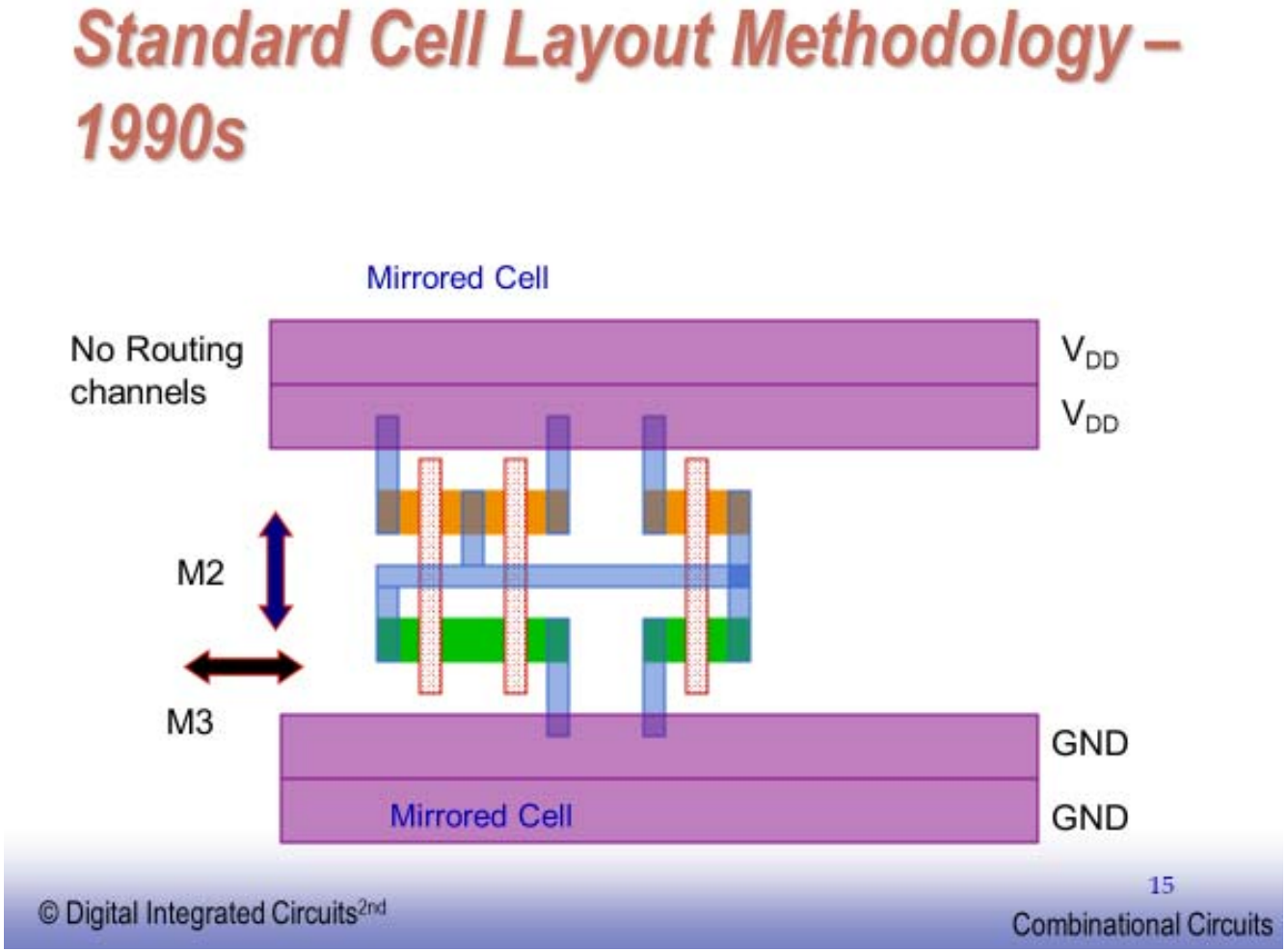
Example: O3AI

- Sketch a stick diagram for O3AI and estimate area

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

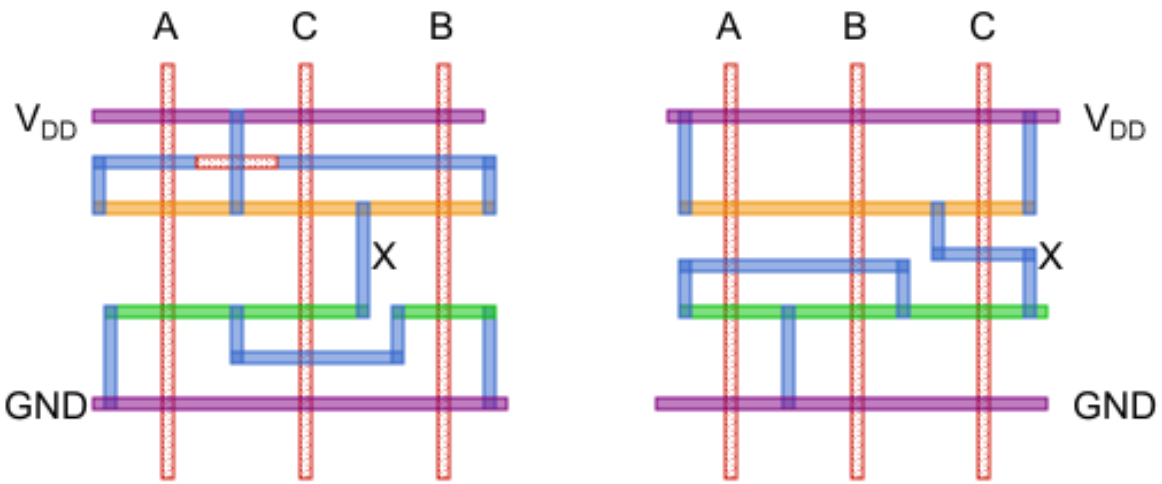


Euler's Path

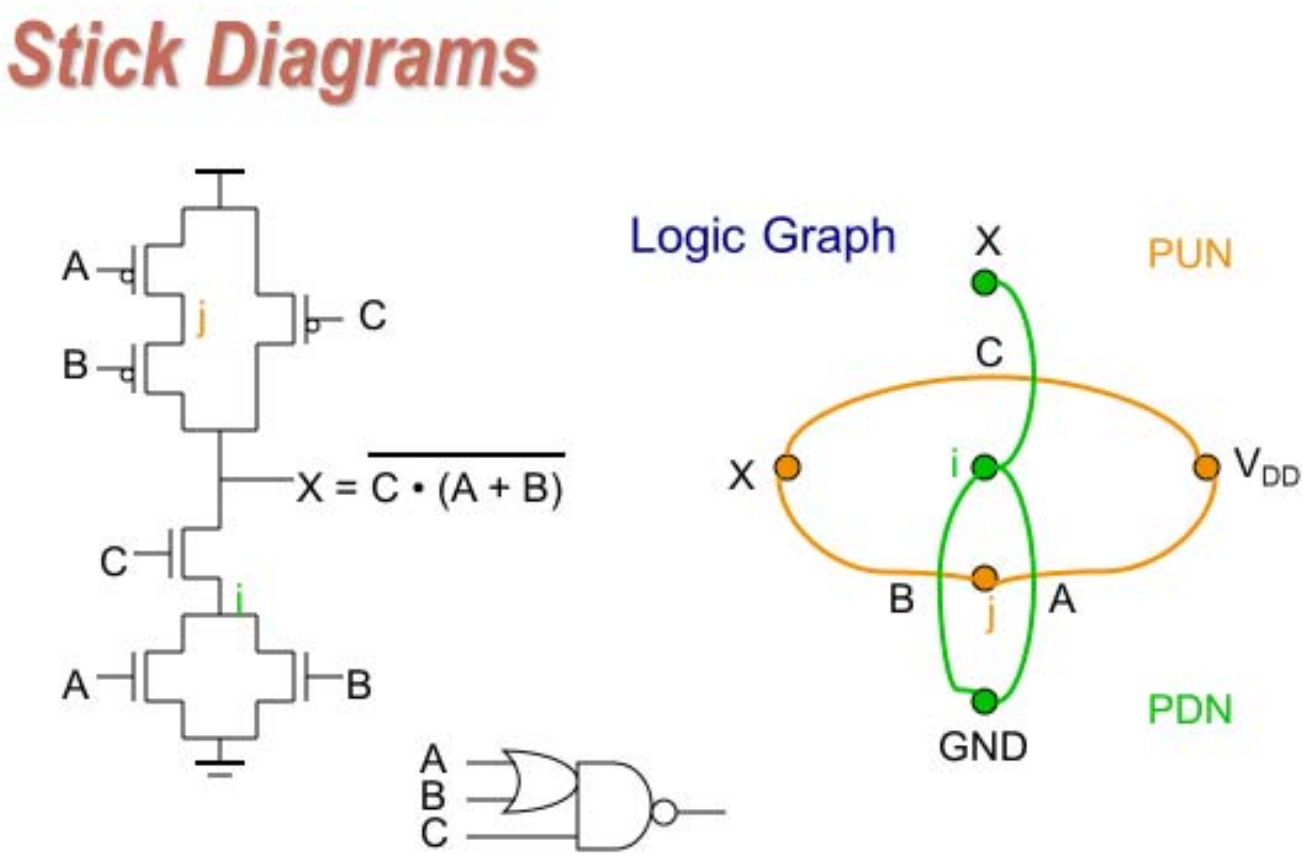

Standard Cell Layout Methodology – 1990s
Mirrored Cell
No Routing channels
V_{DD}
V_{DD}
M2
M3
GND
Mirrored Cell
GND
© Digital Integrated Circuits2nd
15
Combinational Circuits

Euler's Path

Two Versions of $\overline{C \cdot (A + B)}$



Euler's Path

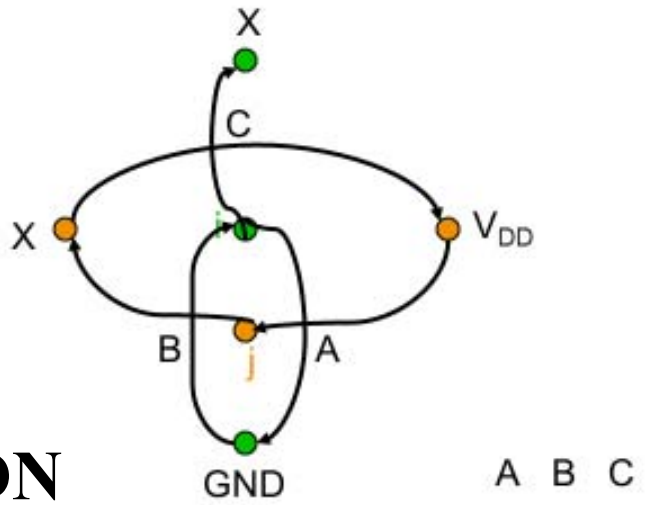

Stick Diagrams
A
B
C
j
i
X = C • (A + B)
Logic Graph
X
PUN
C
i
V_{DD}
B
A
j
PDN
GND

Euler's Path

Consistent Euler Path

A B C
Has a PUN and PDN

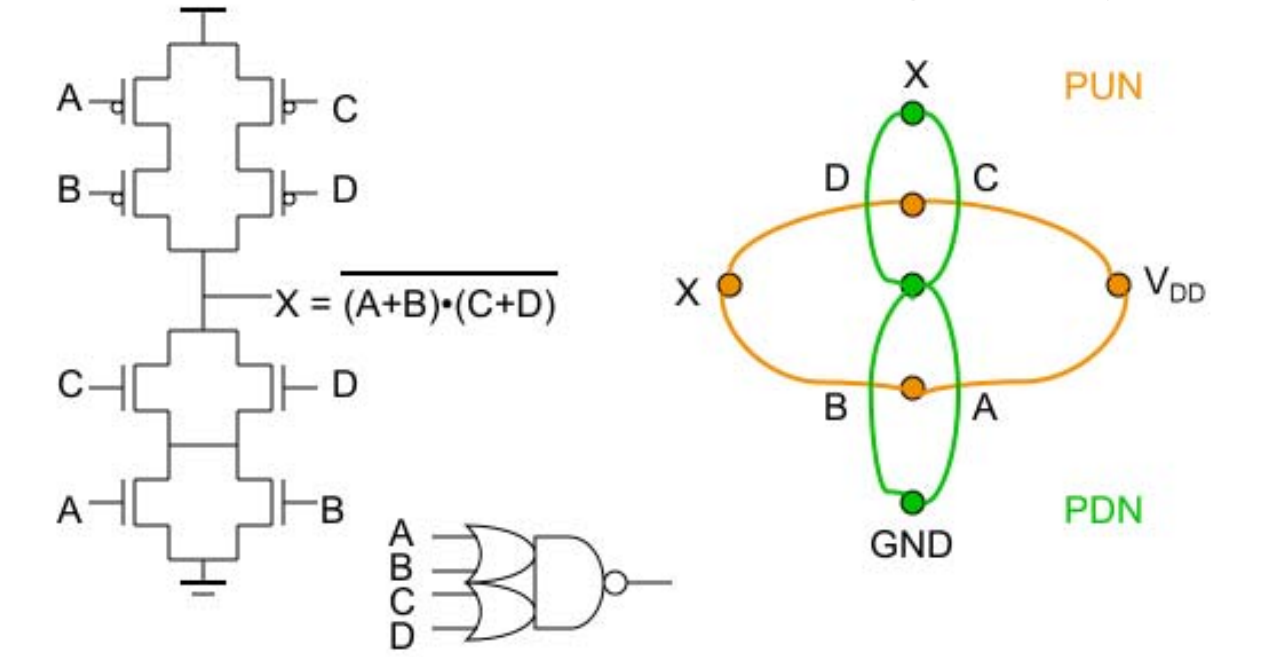
B C A
Has a PUN but no PDN



OAI22 Logic Graph

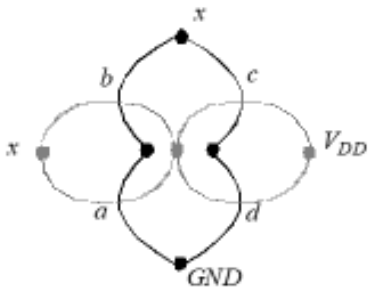
A B C D PDN bot not PUN

A B D C PDN and PUN

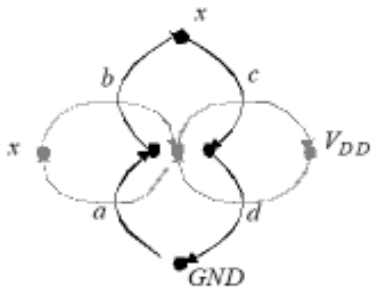


Example: x = ab+cd

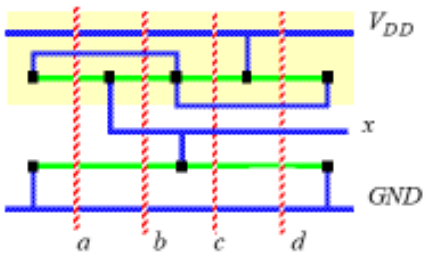
Example: x = ab+cd



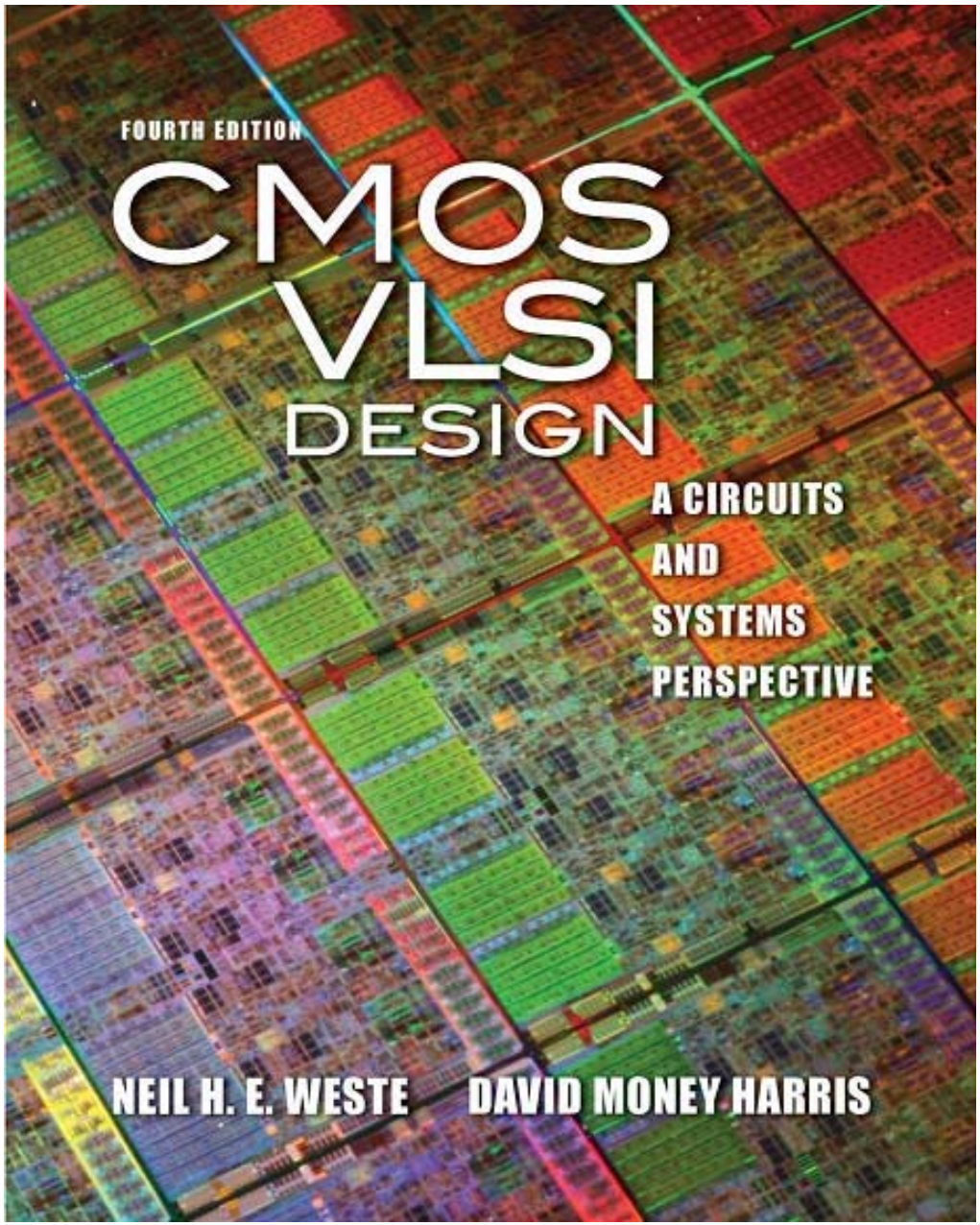
(a) Logic graphs for $\overline{(ab+cd)}$



(b) Euler Paths {a b c d}



(c) stick diagram for ordering {a b c d}



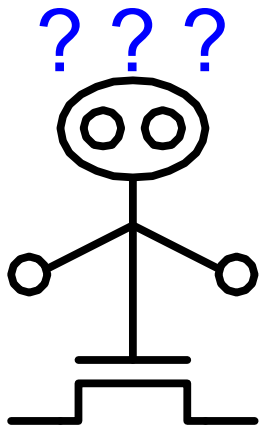
Lecture_5: Logical Effort

Outline

- Logical Effort
- Delay in a Logic Gate
- Multistage Logic Networks
- Choosing the Best Number of Stages
- Example
- Summary

Introduction

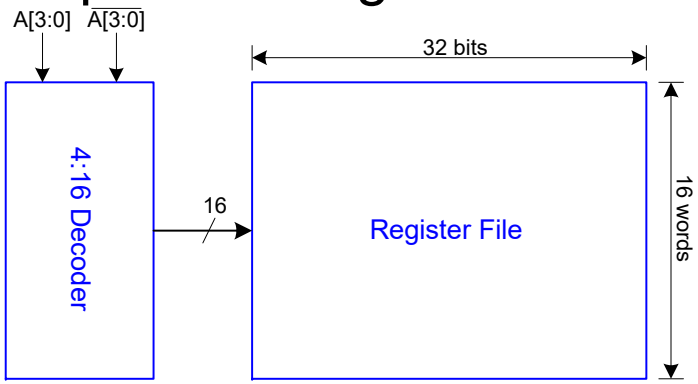
- Chip designers face a bewildering array of choices
  - What is the best circuit topology for a function?
  - How many stages of logic give least delay?
  - How wide should the transistors be?



- Logical effort is a method to make these decisions
  - Uses a simple model of delay
  - Allows back-of-the-envelope calculations
  - Helps make rapid comparisons between alternatives
  - Emphasizes remarkable symmetries

Example

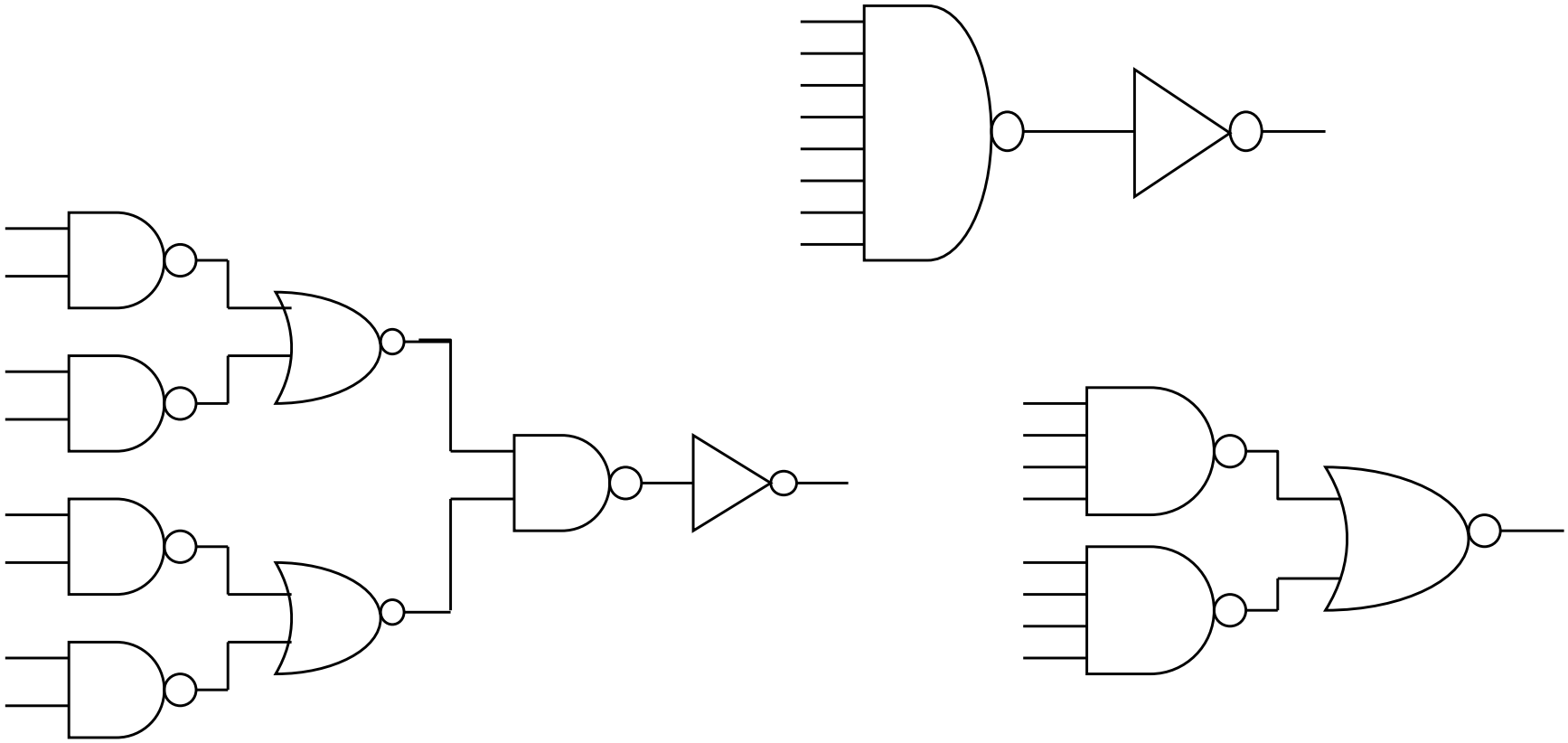
- Ben Bitdiddle is the memory designer for the Motoroil 68W86, an embedded automotive processor. Help Ben design the decoder for a register file.



- Decoder specifications:
  - 16 word register file
  - Each word is 32 bits wide
  - Each bit presents load of 3 unit-sized transistors
  - True and complementary address inputs A[3:0]
  - Each input may drive 10 unit-sized transistors
- Ben needs to decide:
  - How many stages to use?
  - How large should each gate be?
  - How fast can decoder operate?

Alternative Logic Structures

F=ABCDEFGH



Delay in a Logic Gate

- Express delays in process-independent unit
- Delay has two components: d = f + p
- *f*: *effort delay* = *gh* (a.k.a. stage effort)
  - Again, has two components
- *g*: *logical effort*
  - Measures relative ability of gate to deliver current
  - $g \equiv 1$ for inverter
- *h*: *electrical effort* = C_{out} / C_{in}
  - Ratio of output to input capacitance
  - Sometimes called fanout
- *p*: parasitic delay
  - Represents delay of gate driving no load
  - Set by internal parasitic capacitance

$$d = \frac{d_{abs}}{\tau}$$

$\tau =$ 3RC

$\approx$ 3 ps in 65 nm process

60 ps in 0.6 μm process

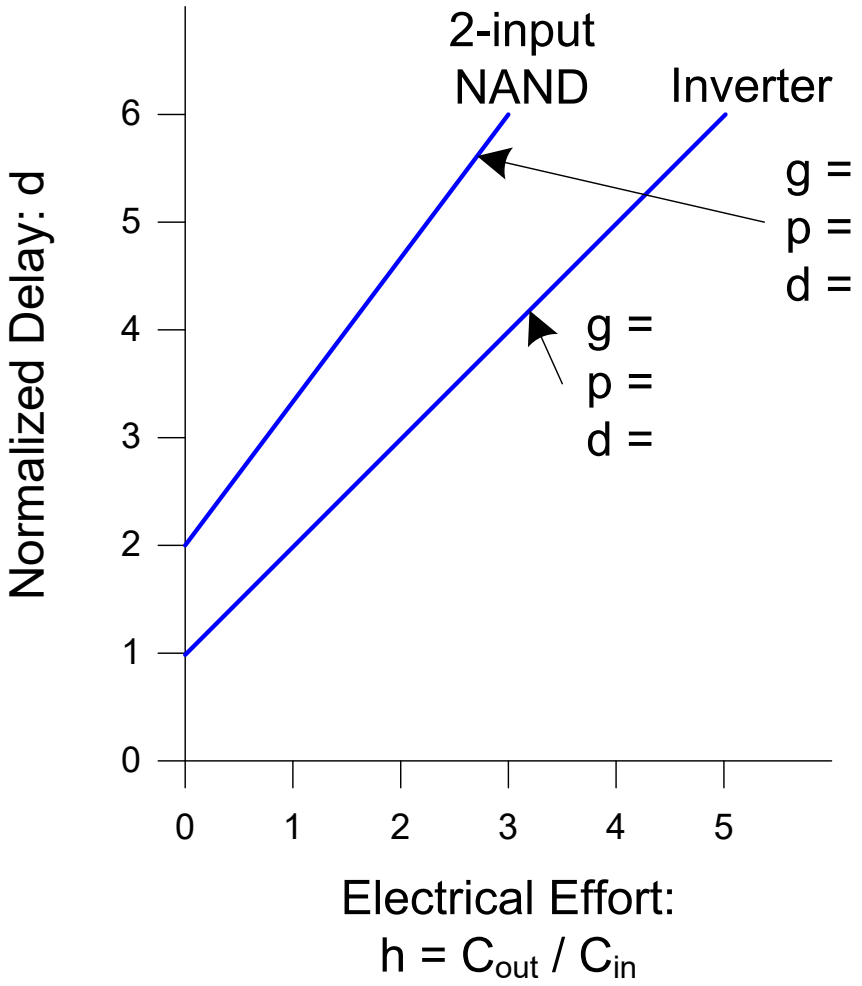
$$p = \text{fan-in}$$

$$g = \frac{C_{gatenorm}}{C_{invnorm}} = \frac{C_{gate}}{3}$$

Delay Plots

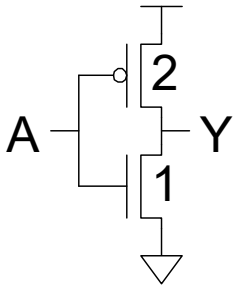
$d = f + p$

$= gh + p$

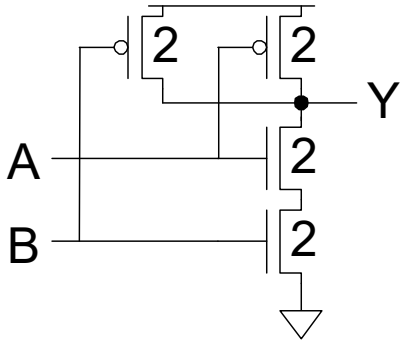


Computing Logical Effort

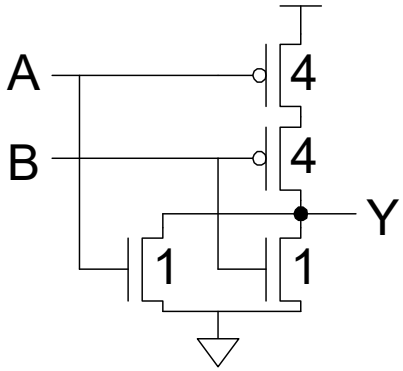
- DEF: *Logical effort is the ratio of the input capacitance of a gate to the input capacitance of an inverter delivering the same output current*.
- Measure from delay vs. fanout plots
- Or estimate by counting transistor widths



C_{in} = 3
g = 3/3



C_{in} = 4
g = 4/3



C_{in} = 5
g = 5/3

Catalog of Gates

- Logical effort of common gates

Gate type	Number of inputs				
	1	2	3	4	n
Inverter	1				
NAND		4/3	5/3	6/3	(n+2)/3
NOR		5/3	7/3	9/3	(2n+1)/3
Tristate / mux	2	2	2	2	2
XOR, XNOR		4, 4	6, 12, 6	8, 16, 16, 8	

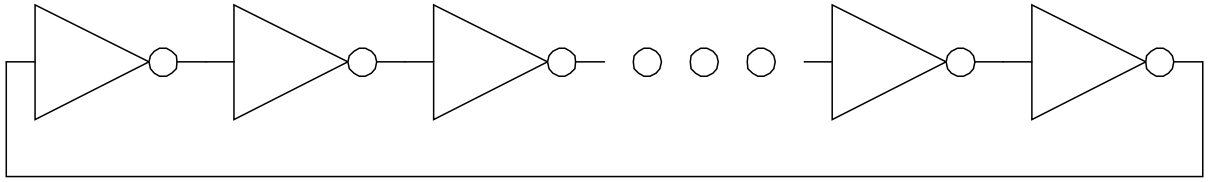
Catalog of Gates

- Parasitic delay of common gates
  - In multiples of p_{inv} (≈ 1)

Gate type	Number of inputs				
	1	2	3	4	n
Inverter	1				
NAND		2	3	4	n
NOR		2	3	4	n
Tristate / mux	2	4	6	8	2n
XOR, XNOR		4	6	8	

Example: Ring Oscillator

- Estimate the frequency of an N-stage ring oscillator



Logical Effort: g =

Electrical Effort: h =

Parasitic Delay: p =

Stage Delay: d =

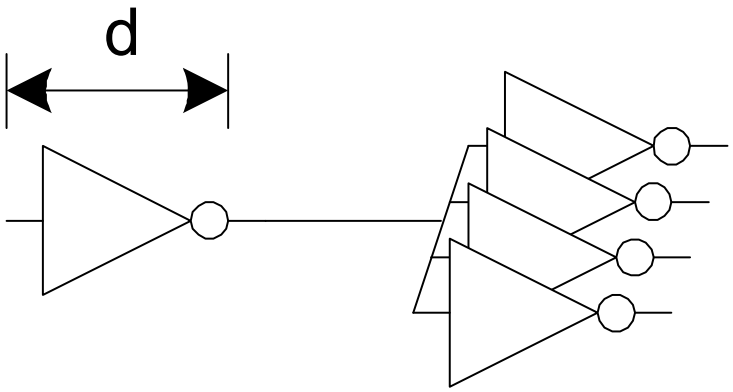
Frequency: f_{osc} =

31 stage ring oscillator in 0.6 μm process has frequency of ~ 200 MHz

$$f_{osc} = \frac{1}{4Nt_{inv}} Hz$$

Example: FO4 Inverter

- Estimate the delay of a fanout-of-4 (FO4) inverter



Logical Effort: g =

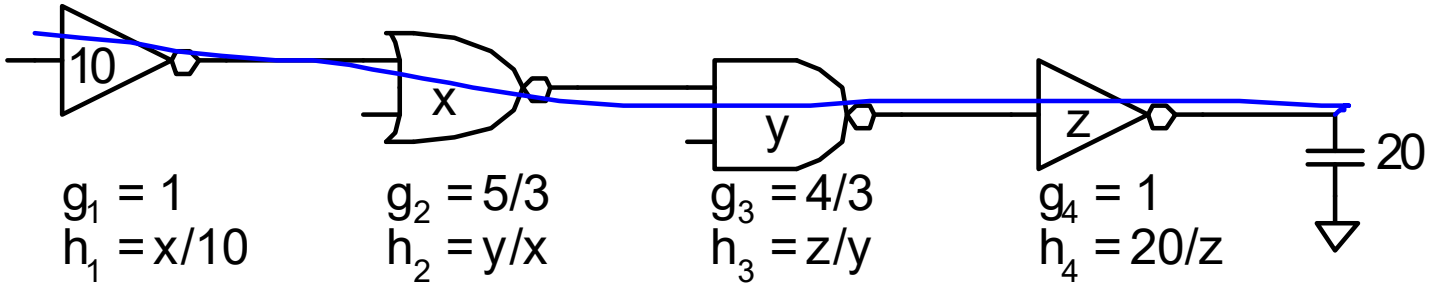
Electrical Effort: h =

Parasitic Delay: p =

Stage Delay: d =

Multistage Logic Networks

- Logical effort generalizes to multistage networks
- *Path Logical Effort* $G = \prod g_i$
- *Path Electrical Effort* $H = \frac{C_{\text{out-path}}}{C_{\text{in-path}}}$
- *Path Effort Delay* $F = \prod f_i = \prod g_i h_i$

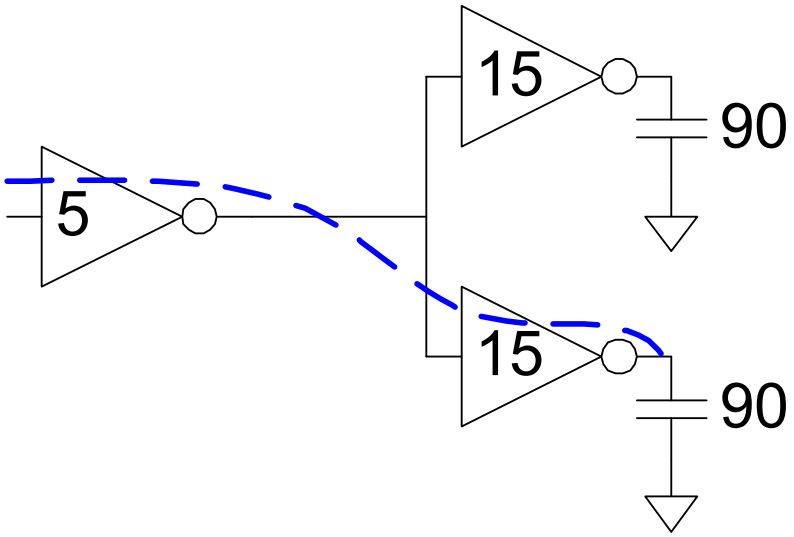


Multistage Logic Networks

- Logical effort generalizes to multistage networks
- *Path Logical Effort* $G = \prod g_i$
- *Path Electrical Effort* $H = \frac{C_{out-path}}{C_{in-path}}$
- *Path Effort Delay* $F = \prod f_i = \prod g_i h_i$
- Can we write F = GH?

Paths that Branch

- No! Consider paths that branch:



G =

H =

GH =

h_1 =

h_2 =

F =

Branching Effort

- Introduce *branching effort*
  - Accounts for branching between stages in path

$$b = \frac{C_{\text{on path}} + C_{\text{off path}}}{C_{\text{on path}}}$$

$$B = \prod b_i$$

Note:

$$\prod h_i = BH$$

- Now we compute the path effort delay.
  - F = GBH

Multistage Delays

- Path Effort Delay $D_F = \sum f_i$
- Path Parasitic Delay $P = \sum p_i$
- Path Delay $D = \sum d_i = D_F + P$

Designing Fast Circuits

$$D = \sum d_i = D_F + P$$

- Delay is smallest when each stage bears same effort

$$\hat{f} = g_i h_i = F^{\frac{1}{N}}$$

- Thus minimum delay of N stage path is

- This is a key result of logical effort
  - Find fastest possible delay
  - Doesn't require calculating gate sizes

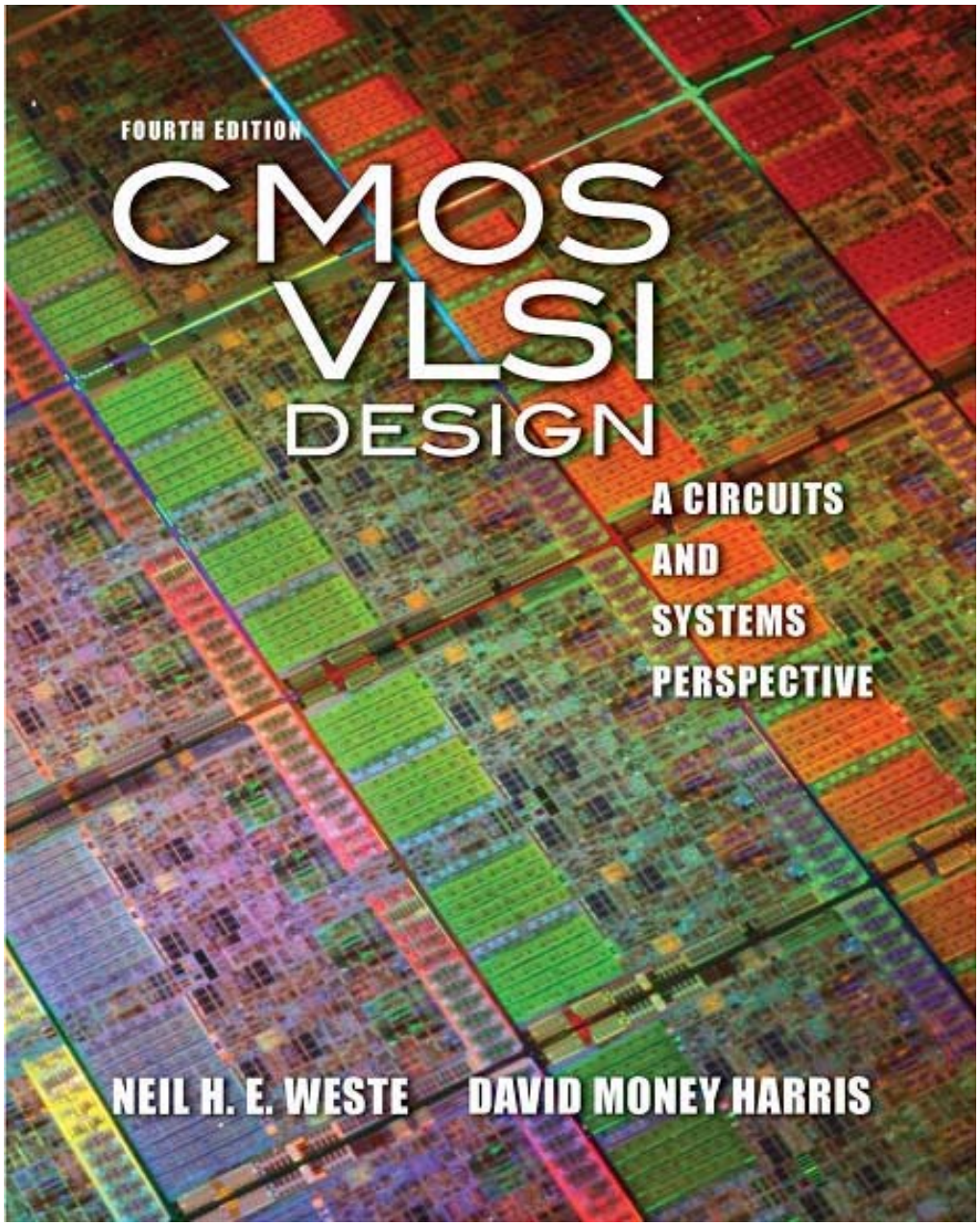
Gate Sizes

- How wide should the gates be for least delay?

$$\hat{f} = gh = g\frac{C_{out}}{C_{in}}$$

$$\Rightarrow C_{in_i} = \frac{g_i C_{out_i}}{\hat{f}}$$

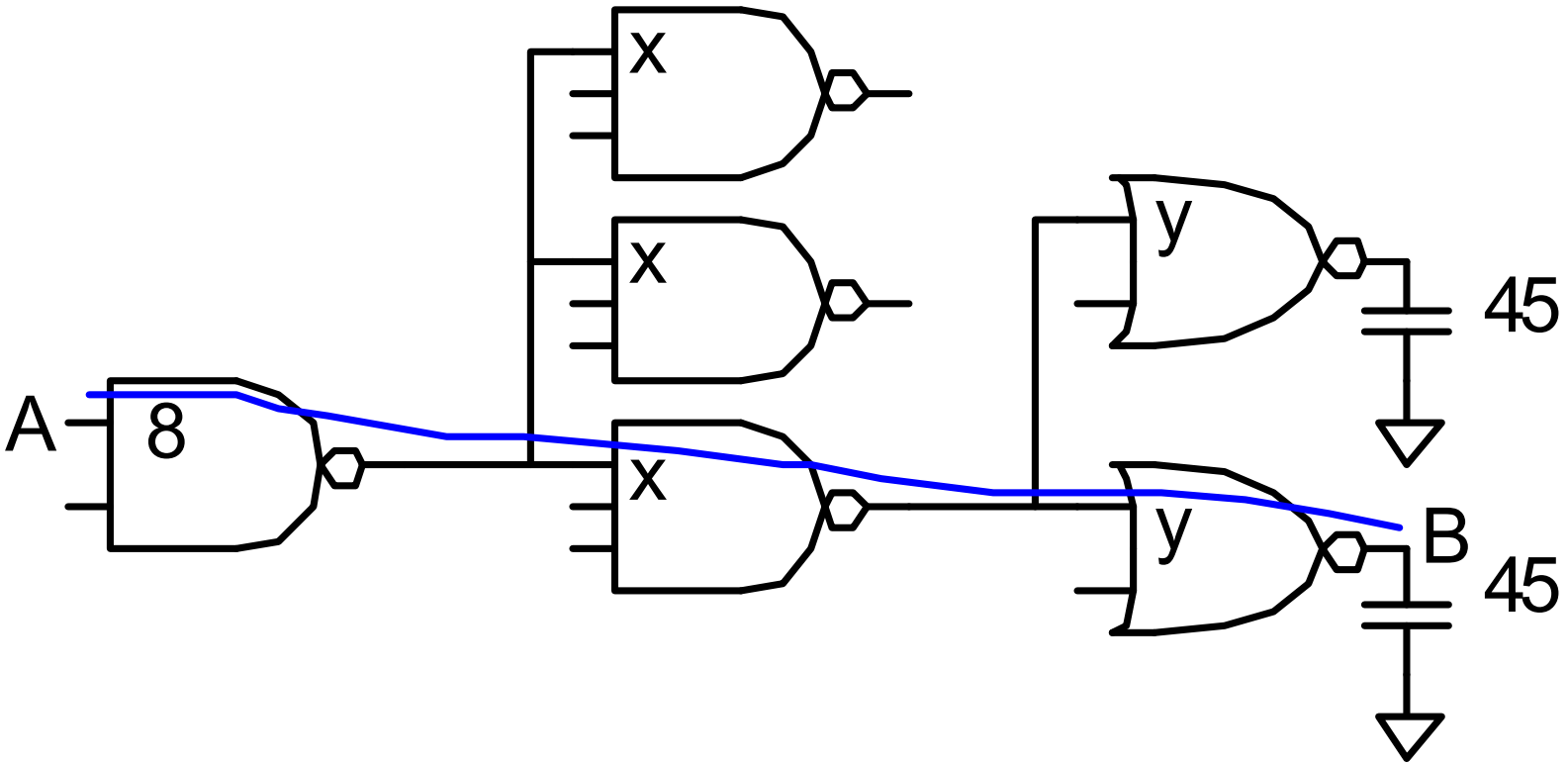
- Working backward, apply capacitance transformation to find input capacitance of each gate given the load it drives.
- Check work by verifying input cap spec is met.



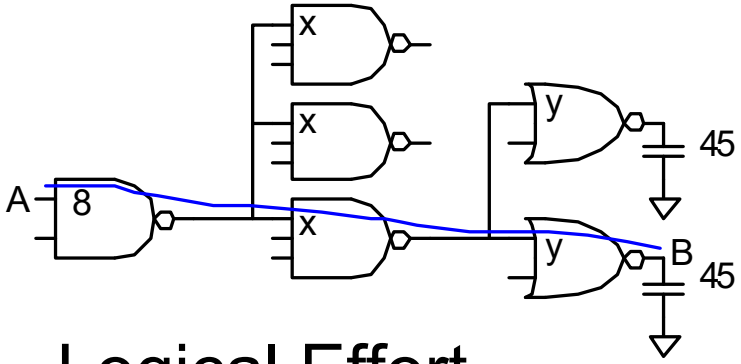
Lecture_6: Logical Effort

Example: 3-stage path

- Select gate sizes x and y for least delay from A to B



Example: 3-stage path



Logical Effort G =

Electrical Effort H =

Branching Effort B =

Path Effort F =

Best Stage Effort $\hat{f} =$

Parasitic Delay P =

Delay D =

Example: 3-stage path

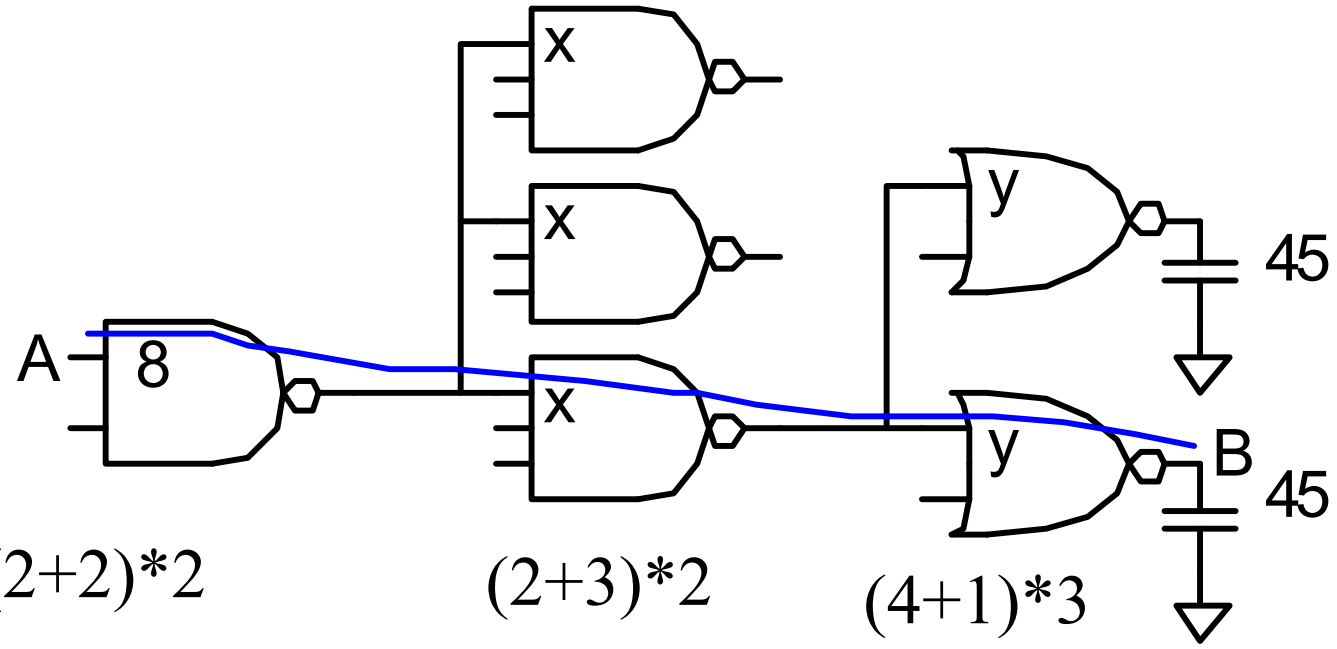
- Work backward for sizes

  y =

  x =

2y/x for branching

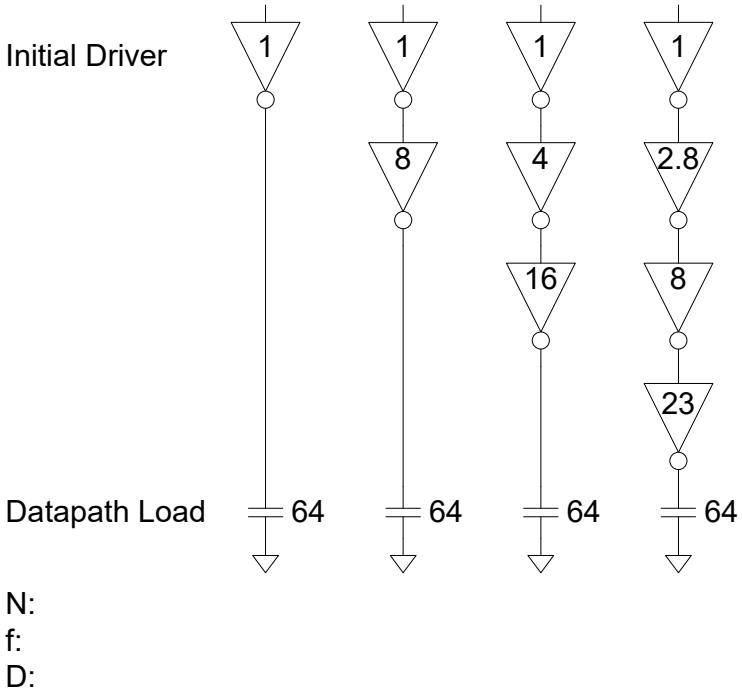
(3x/Cin).4/3 = 5 gives Cin=8



Best Number of Stages

- How many stages should a path use?
  - Minimizing number of stages is not always fastest
- Example: drive 64-bit datapath with unit inverter

D =

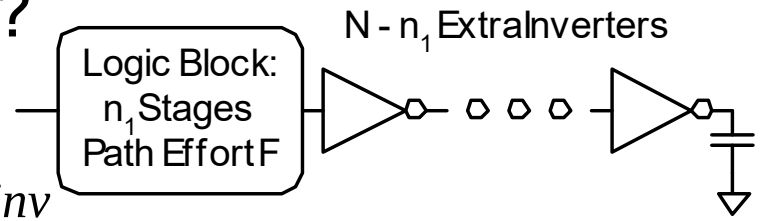


$g_i = 1$ *then* h_i *is always*

Equal to $F^{\frac{1}{N}} = f$

Derivation

- Consider adding inverters to end of path
  - How many give least delay?



$$D = NF^{\frac{1}{N}} + \sum_{i=1}^{n_1} p_i + \left(N - n_1\right) p_{inv}$$

$$\frac{\partial D}{\partial N} = -\frac{1}{N} F^{\frac{1}{N}} \ ln\mathrm{F} + F^{\frac{1}{N}} + p_{inv} = 0$$

- Define best stage effort $\rho = F^{\frac{1}{N}}$

$$p_{inv} + \rho\left(1 - \ln \rho\right) = 0$$

$$\frac{d}{dx}\left(a^{\frac{1}{x}}\right) = -\frac{a^{\frac{1}{x}} ln a}{x^2}$$

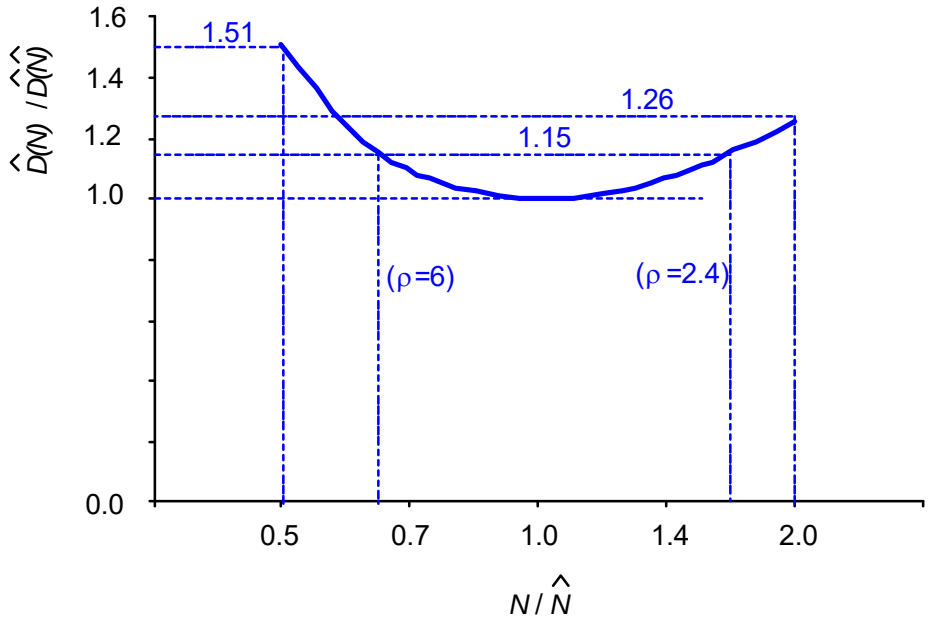
Best Stage Effort

- $p_{inv} + \rho(1 - \ln \rho) = 0$ has no closed-form solution

- Neglecting parasitic (p_{inv} = 0), we find ρ = 2.718 (e)
- For p_{inv} = 1, solve numerically for ρ = 3.59

Sensitivity Analysis

- How sensitive is delay to using exactly the best number of stages?

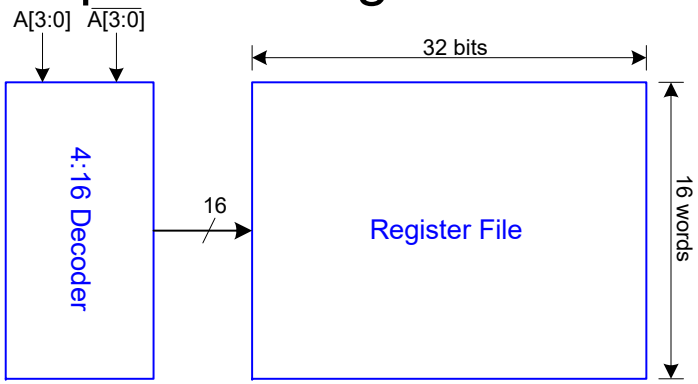


- 2.4 < ρ < 6 gives delay within 15% of optimal
  - We can be sloppy!
  - I like ρ = 4

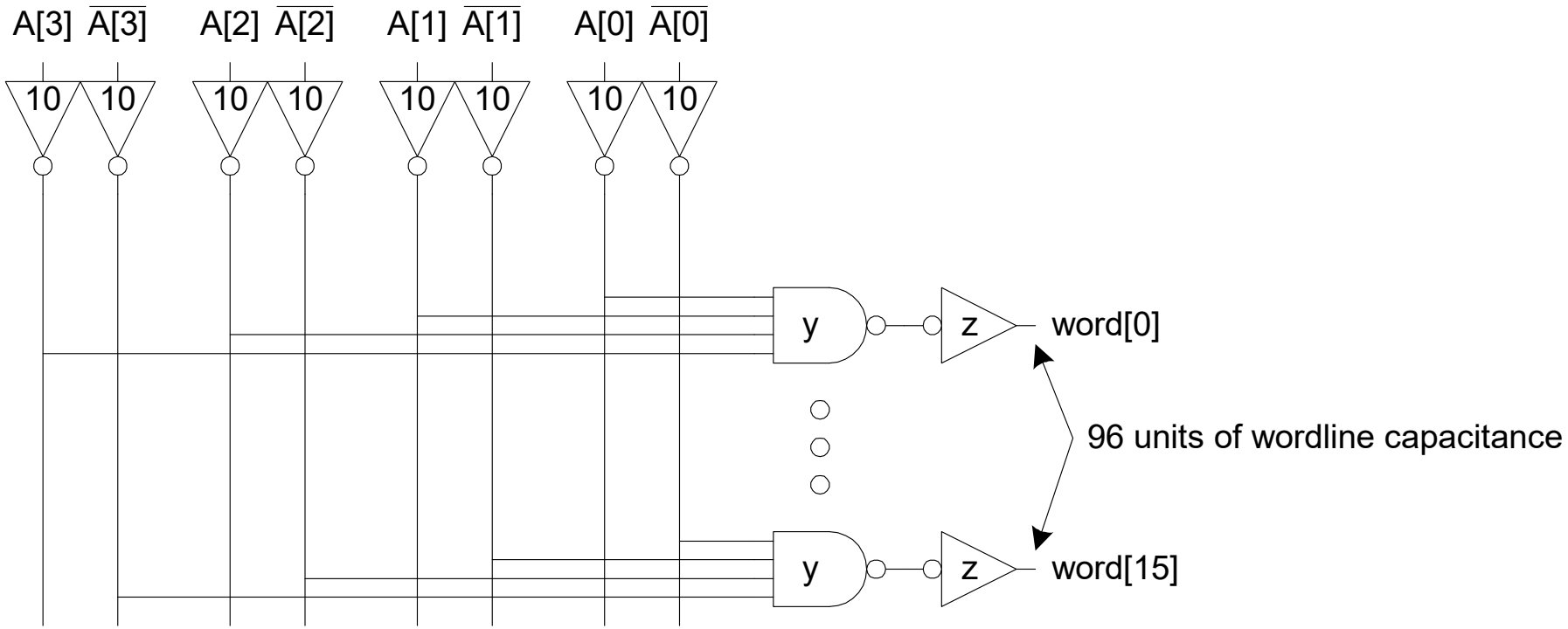
$$\rho = 4 = F^{\frac{1}{N}} \Rightarrow N = log_4 F$$

Example, Revisited

- Ben Bitdiddle is the memory designer for the Motoroil 68W86, an embedded automotive processor. Help Ben design the decoder for a register file.



- Decoder specifications:
  - 16 word register file
  - Each word is 32 bits wide
  - Each bit presents load of 3 unit-sized transistors
  - True and complementary address inputs A[3:0]
  - Each input may drive 10 unit-sized transistors
- Ben needs to decide:
  - How many stages to use?
  - How large should each gate be?
  - How fast can decoder operate?


A[3]
$\overline{A[3]}$
A[2]
$\overline{A[2]}$
A[1]
$\overline{A[1]}$
A[0]
$\overline{A[0]}$
10
y
z
word[0]
96 units of wordline capacitance
word[15]

Number of Stages

- Decoder effort is mainly electrical and branching

  Electrical Effort: H =

  Branching Effort: B =

- If we neglect logical effort (assume G = 1)

  Path Effort: F =

  Number of Stages: N =

- Try a -stage design

Gate Sizes & Delay

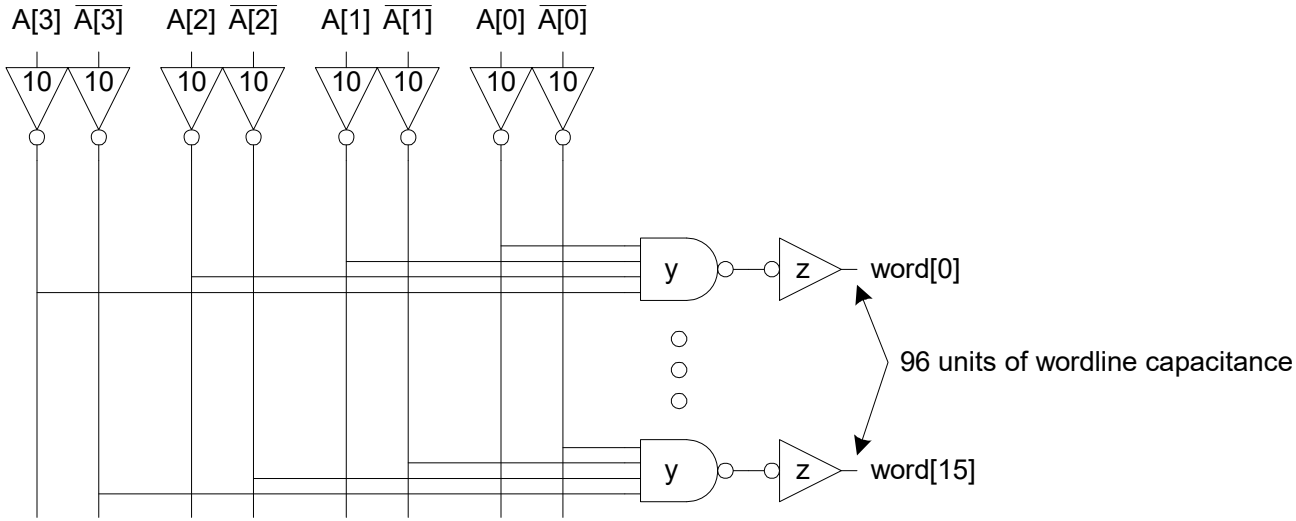
Logical Effort: G =

Path Effort: F =

Stage Effort: $\hat{f} =$

Path Delay: $D =$

Gate sizes: z = y =



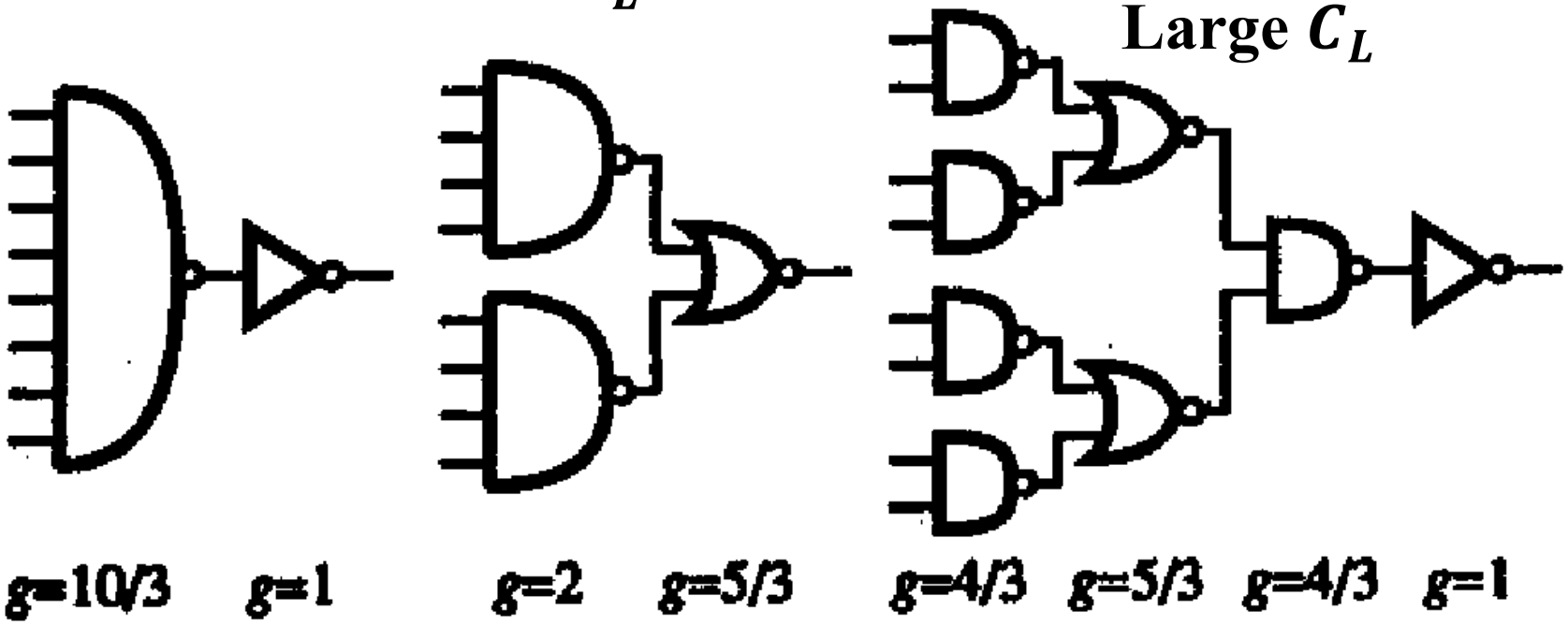
G ,H and B Calculations

- G = 1(INV10) * 6/3 (NAND4) * 1(INVz) = 6/3 = 2
- H = 3*32/10 = 9.6
- B, each input is connected to 8 words because the input variables A[0-3] and their complements are available.
  -So, path branching is (1+7)/1 one ON path and seven OFF paths.
  - So, B is equal to 8
- Then F = GHB = 6/3*9.6*8 = 153.6 ~ 154

Which is the best!!

Low C_L

Large C_L



G = 10/3 G = 10/3 G = 80/27

P = 8 +1 = 9 P = 4 + 2 = 6 P = 2 + 2 + 2 + 1 = 7

Comparison

- Compare many alternatives with a spreadsheet
- $D = N(76.8\ G)^{1/N} + P$

Design	N	G	P	D
NOR4	1	3	4	234
NAND4-INV	2	2	5	29.8
NAND2-NOR2	2	20/9	4	30.1
INV-NAND4-INV	3	2	6	22.1
NAND4-INV-INV-INV	4	2	7	21.1
NAND2-NOR2-INV-INV	4	20/9	6	20.5
NAND2-INV-NAND2-INV	4	16/9	6	19.7
INV-NAND2-INV-NAND2-INV	5	16/9	7	20.4
NAND2-INV-NAND2-INV-INV-INV	6	16/9	8	21.6

Review of Definitions

Term	Stage	Path
number of stages	1	N
logical effort	g	$G = \prod g_i$
electrical effort	$h = \frac{C_{out}}{C_{in}}$	$H = \frac{C_{out\text{-}path}}{C_{in\text{-}path}}$
branching effort	$b = \frac{C_{on\text{-}path} + C_{off\text{-}path}}{C_{on\text{-}path}}$	$B = \prod b_i$
effort	$f = gh$	$F = GBH$
effort delay	f	$D_F = \sum f_i$
parasitic delay	p	$P = \sum p_i$
delay	$d = f + p$	$D = \sum d_i = D_F + P$

Method of Logical Effort

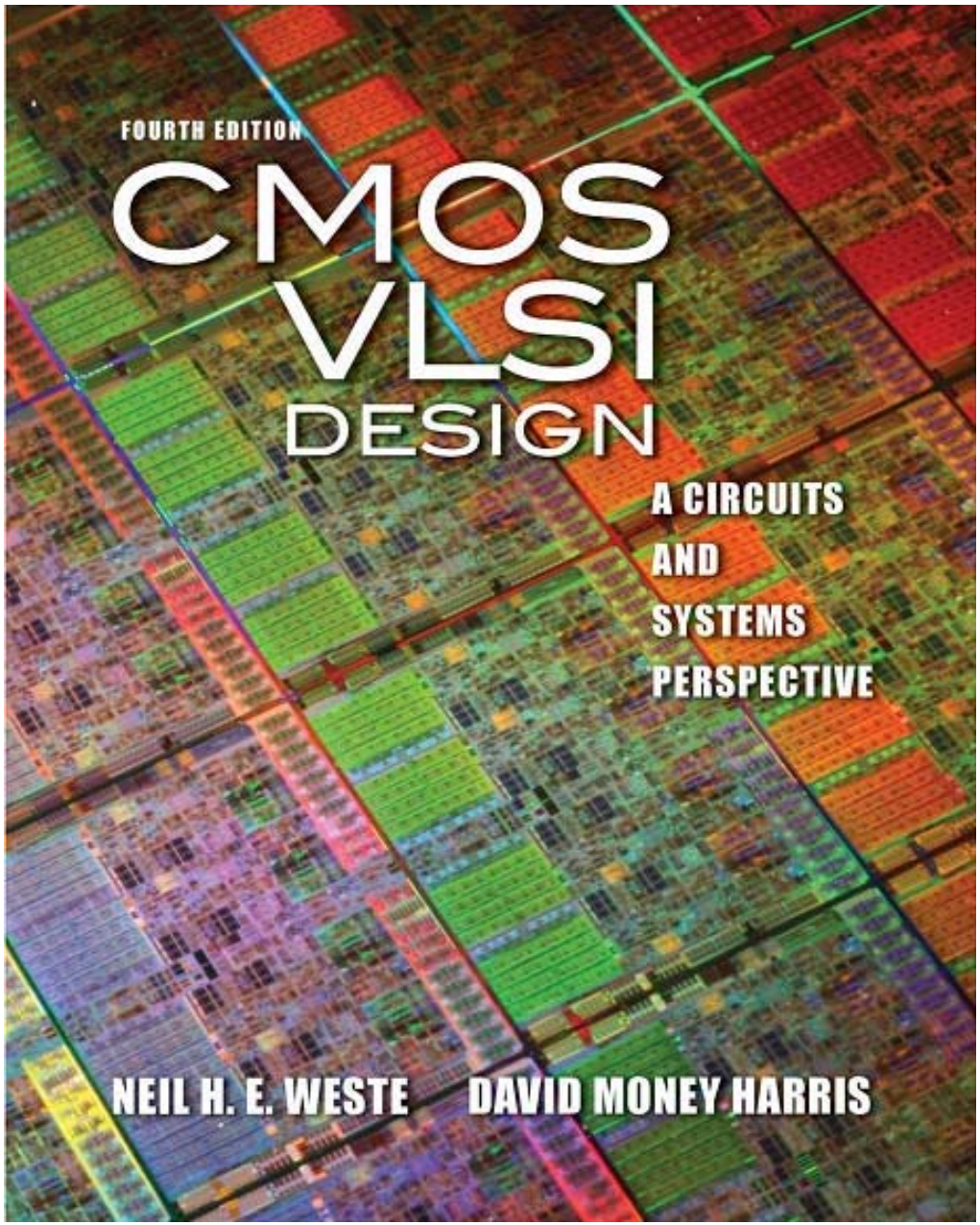
1) Compute path effort $F = GBH$
2) Estimate best number of stages $N = \log_4 F$
3) Sketch path with N stages
4) Estimate least delay $D = NF^{\frac{1}{N}} + P$
5) Determine best stage effort $\hat{f} = F^{\frac{1}{N}}$
6) Find gate sizes $C_{in_i} = \frac{g_i C_{out_i}}{\hat{f}}$

Limits of Logical Effort

- Chicken and egg problem
  - Need path to compute G
  - But don't know number of stages without G
- Simplistic delay model
  - Neglects input rise time effects
- Interconnect
  - Iteration required in designs with wire
- Maximum speed only
  - Not minimum area/power for constrained delay

Summary

- Logical effort is useful for thinking of delay in circuits
  - Numeric logical effort characterizes gates
  - NANDs are faster than NORs in CMOS
  - Paths are fastest when effort delays are ~4
  - Path delay is weakly sensitive to stages, sizes
  - But using fewer stages doesn't mean faster paths
  - Delay of path is about $\log_4 F$ FO4 inverter delays
  - Inverters and NAND2 best for driving large caps
- Provides language for discussing fast circuits
  - But requires practice to master



Lecture_7: Power

Outline

- Power and Energy
- Dynamic Power
- Static Power

Power and Energy

- Power is drawn from a voltage source attached to the V_{DD} pin(s) of a chip.
- Instantaneous Power: $P(t) =$
- Energy: $E =$
- Average Power: $P_{avg} =$

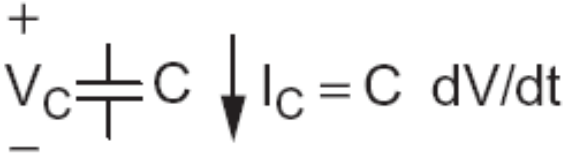
Power in Circuit Elements

$$P_{VDD}(t) = I_{DD}(t) V_{DD}$$

$$P_R(t) = \frac{V_R^2(t)}{R} = I_R^2(t) R$$

$$E_C = \int_0^\infty I(t) V(t)\, dt = \int_0^\infty C \frac{dV}{dt} V(t)\, dt$$

$$= C \int_0^{V_C} V(t)\, dV = \tfrac{1}{2} C V_C^2$$



Charging a Capacitor

- When the gate output rises
  - Energy stored in capacitor is

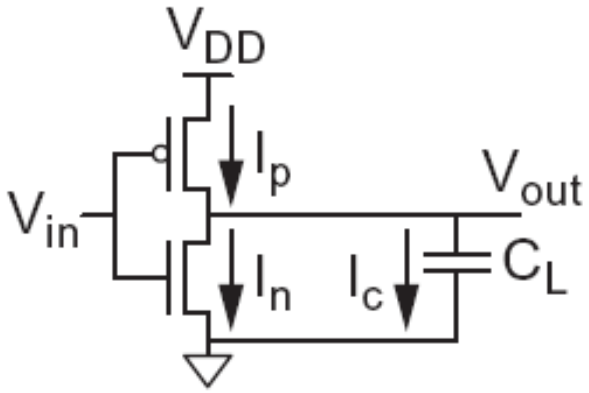
$$E_C = \frac{1}{2} C_L V_{DD}^2$$

  - But energy drawn from the supply is

$$E_{VDD} = \int_0^\infty I(t) V_{DD} dt = \int_0^\infty C_L \frac{dV}{dt} V_{DD} dt$$

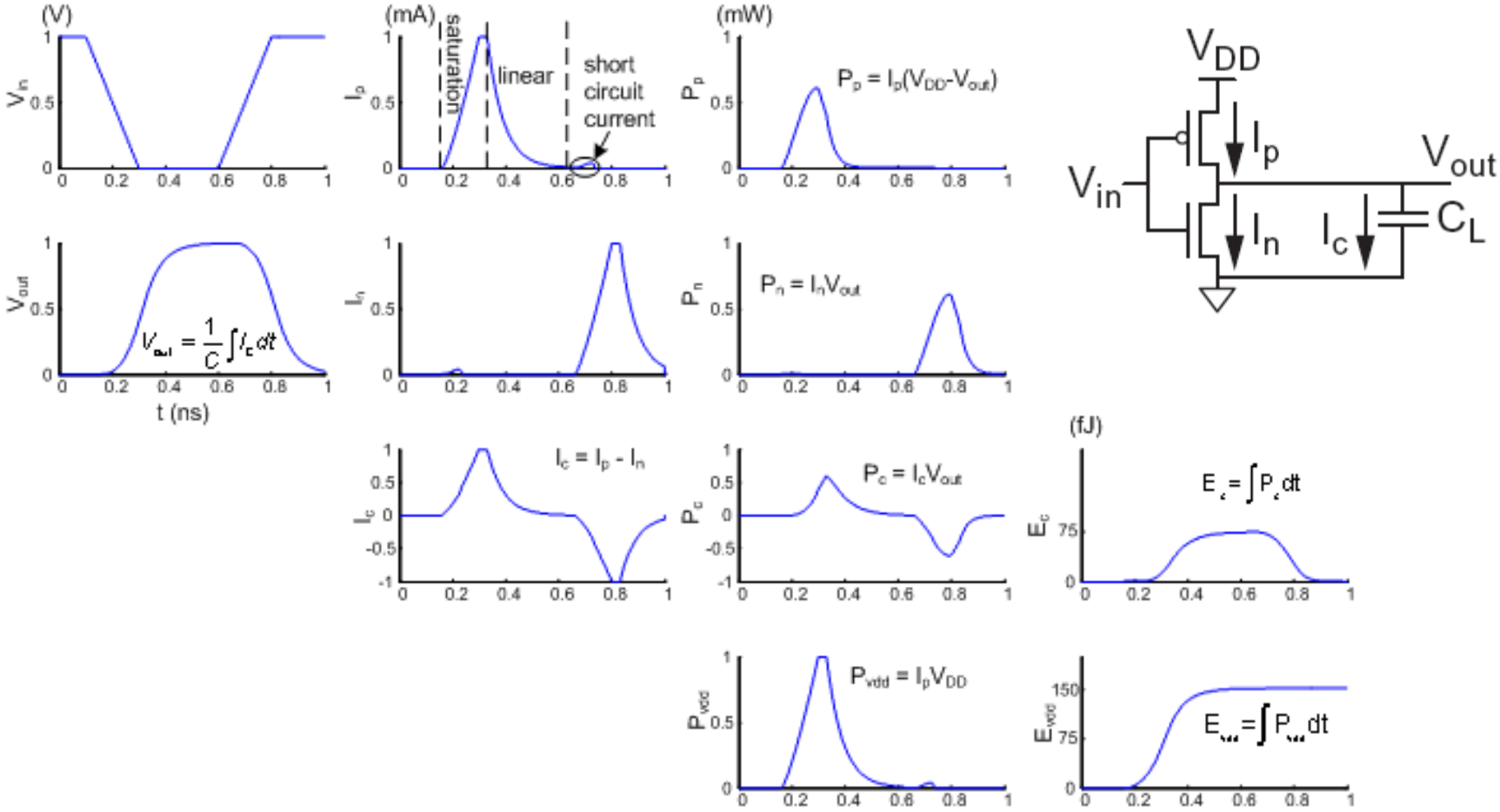
$$= C_L V_{DD} \int_0^{V_{DD}} dV = C_L V_{DD}^2$$

  - Half the energy from V_{DD} is dissipated in the pMOS transistor as heat, other half stored in capacitor
- When the gate output falls
  - Energy in capacitor is dumped to GND
  - Dissipated as heat in the nMOS transistor



Switching Waveforms

- Example: V_{DD} = 1.0 V, C_L = 150 fF, f = 1 GHz



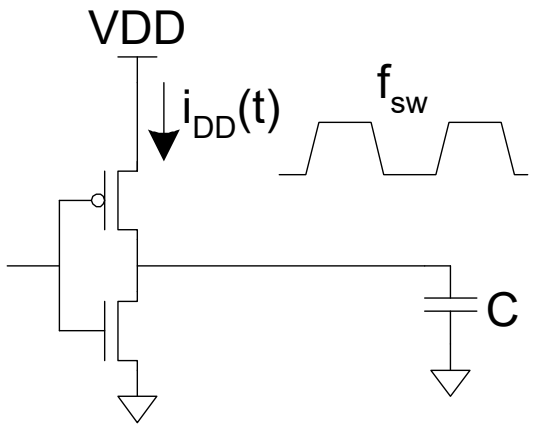
Switching Power

$$P_{\text{switching}} = \frac{1}{T}\int_0^T i_{DD}(t)V_{DD}\,dt$$

$$= \frac{V_{DD}}{T}\int_0^T i_{DD}(t)\,dt$$

$$= \frac{V_{DD}}{T}\left[Tf_{\text{sw}}CV_{DD}\right]$$

$$= CV_{DD}{}^2 f_{\text{sw}}$$



Activity Factor

- Suppose the system clock frequency = f
- Let $f_{sw} = \alpha f$, where α = activity factor
  - If the signal is a clock, $\alpha = 1$
  - If the signal switches once per cycle, $\alpha = ½$
- Dynamic power:

$$P_{\text{switching}} = \alpha C V_{DD}^{\ 2} f$$

Short Circuit Current

- When transistors switch, both nMOS and pMOS networks may be momentarily ON at once
- Leads to a blip of “short circuit” current.
- < 10% of dynamic power if rise/fall times are comparable for input and output
- We will generally ignore this component

Power Dissipation Sources

- $P_{total} = P_{dynamic} + P_{static}$
- Dynamic power: $P_{dynamic} = P_{switching} + P_{shortcircuit}$
  - Switching load capacitances
  - Short-circuit current
- Static power: $P_{static} = (I_{sub} + I_{gate} + I_{junct} + I_{contention})V_{DD}$
  - Subthreshold leakage
  - Gate leakage
  - Junction leakage
  - Contention current

Dynamic Power Example

- 1 billion transistor chip
  - 50M logic transistors
    - Average width: 12 λ
    - Activity factor = 0.1
  - 950M memory transistors
    - Average width: 4 λ
    - Activity factor = 0.02
  - 1.0 V 65 nm process, $L_{eff} = 50nm$
  - C = 1 fF/μm (gate) + 0.8 fF/μm (diffusion)
- Estimate dynamic power consumption @ 1 GHz. Neglect wire capacitance and short-circuit current.

Solution

$$C_{logic} = (50 \times 10^6)(12 \times 0.025\mu m)(1.0 + 0.8)\left(\frac{pF}{\mu m}\right) = 27\text{nF}$$

$$C_{mem} = (950 \times 10^6)(4 \times 0.025\mu m)(1.0 + 0.8)\left(\frac{pF}{\mu m}\right) = 171\text{nF}$$

$$P_{dynamic} = [0.1C_{logic} + 0.02C_{mem}](1.0)^2(1.0Ghz) = 6.1W$$

$$f = 50\text{nm and } \lambda = 25nm = 0.025\mu m$$

Dynamic Power Reduction

- $P_{\text{switching}} = \alpha C V_{DD}^{\ 2} f$
- Try to minimize:
  - Activity factor
  - Capacitance
  - Supply voltage
  - Frequency

Activity Factor Estimation

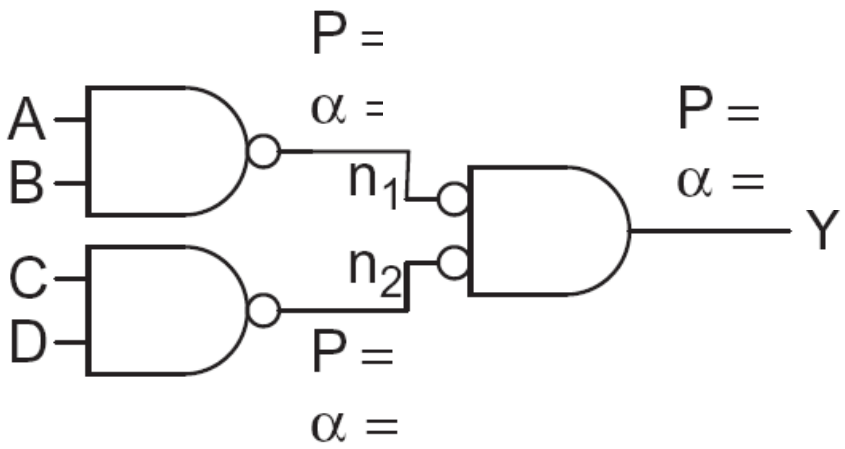
- Let P_i = Prob(node i = 1)
  - $\overline{P}_i = 1 - P$, Prob(node i = 0)
- $\alpha_i = \overline{P}_i \times P_i$
- Completely random data has P = 0.5 and α = 0.25
- Data is often not completely random
  - Structured data, e.g. upper bits of 64-bit unsigned integer representing bank account balances are usually 0
- Data propagating through ANDs and ORs has lower activity factor
  - Depends on design, but typically $\alpha \approx 0.1$

Switching Probability

Gate	P_Y
AND2	$P_A P_B$
AND3	$P_A P_B P_C$
OR2	$1 - \overline{P}_A \overline{P}_B$
NAND2	$1 - P_A P_B$
NOR2	$\overline{P}_A \overline{P}_B$
XOR2	$P_A \overline{P}_B + \overline{P}_A P_B$

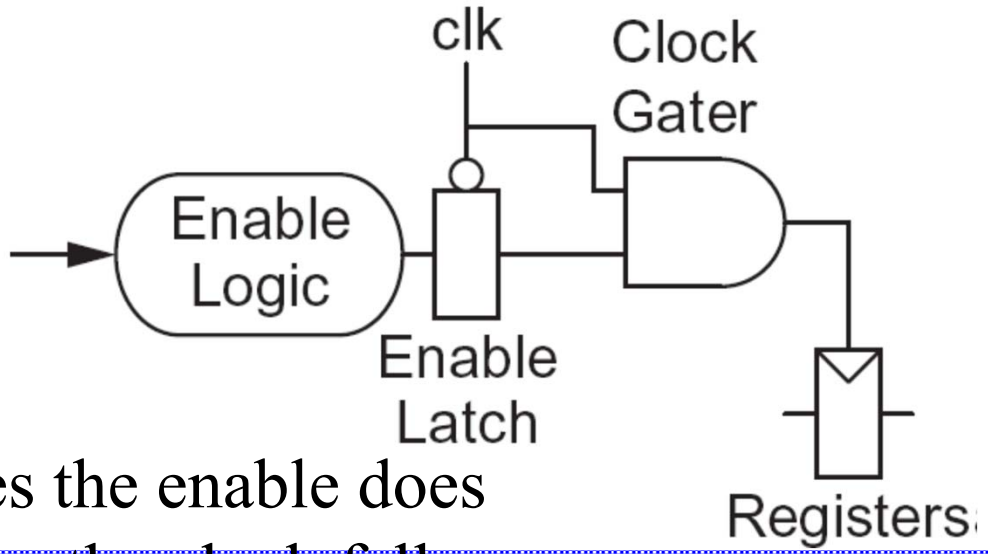
Example

- A 4-input AND is built out of two levels of gates
- Estimate the activity factor at each node if the inputs have P = 0.5

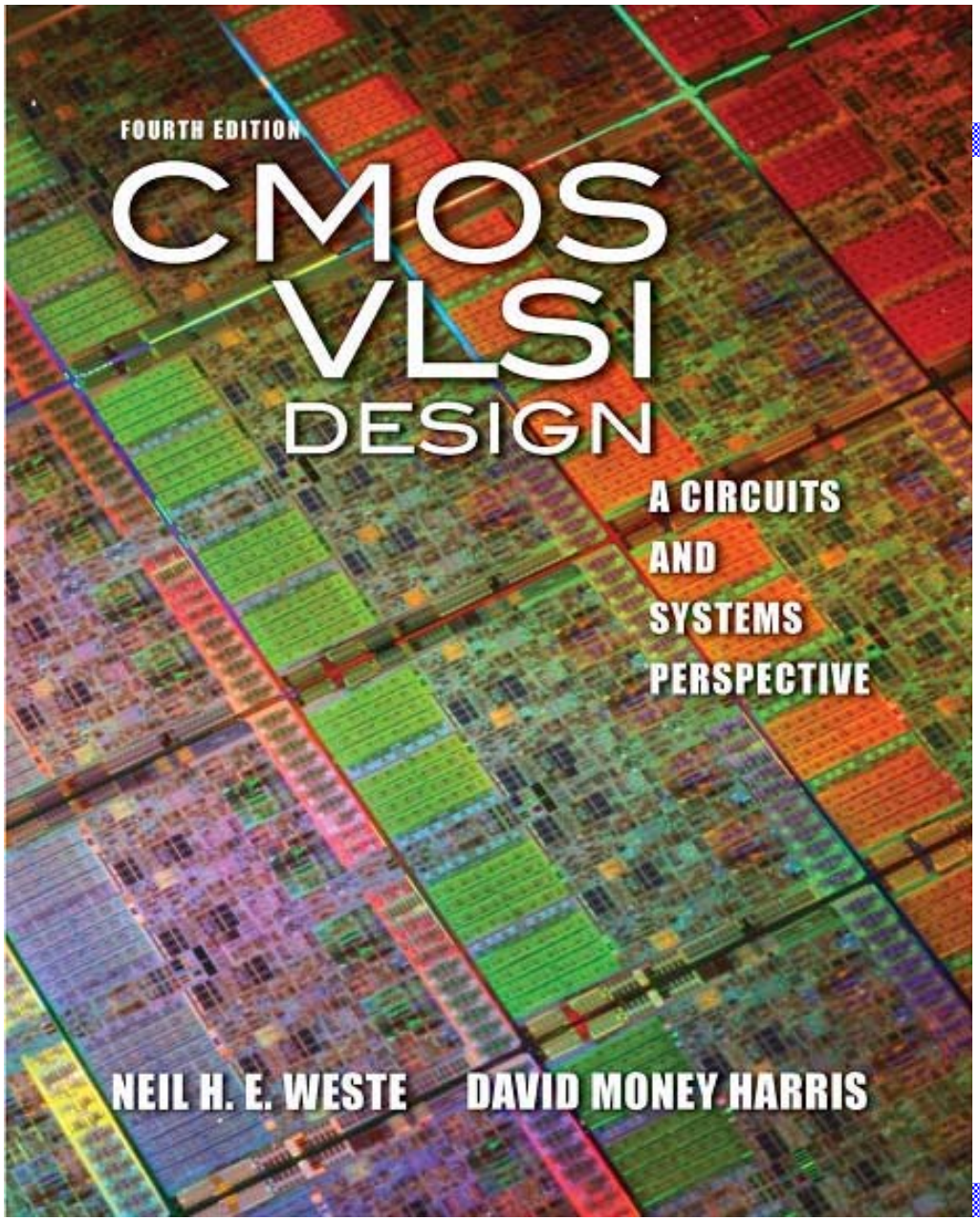


Clock Gating

- The best way to reduce the activity is to turn off the clock to registers in unused blocks
  - Saves clock activity ($\alpha = 1$)
  - Eliminates all switching activity in the block
  - Requires determining if block will be used



The latch ensures the enable does not change before the clock falls



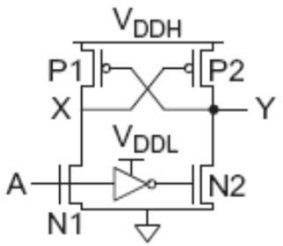
Lecture_8: Power

Capacitance

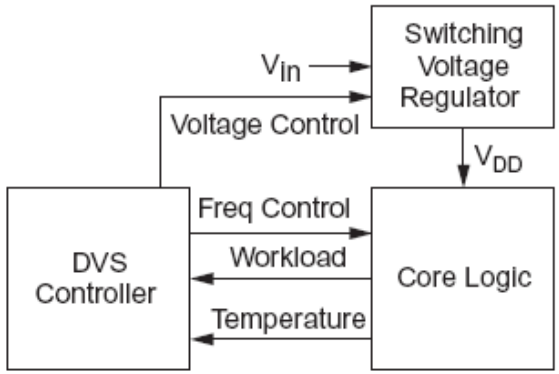
- Gate capacitance
  - Fewer stages of logic
  - Small gate sizes
- Wire capacitance
  - Good floorplanning to keep communicating blocks close to each other
  - Drive long wires with inverters or buffers rather than complex gates

Voltage / Frequency

- Run each block at the lowest possible voltage and frequency that meets performance requirements
- Voltage Domains
  - Provide separate supplies to different blocks
  - Level converters required when crossing from low to high V_{DD} domains



- Dynamic Voltage Scaling
  - Adjust V_{DD} and f according to workload



Static Power

- Static power is consumed even when chip is quiescent.
  - Leakage draws power from nominally OFF devices
  - Ratioed circuits burn power in fight between ON transistors

Static Power Example

- Revisit power estimation for 1 billion transistor chip
- Estimate static power consumption
  - Subthreshold leakage
    - Normal V_t: 100 nA/μm
    - High V_t: 10 nA/μm
    - High Vt used in all memories and in 95% of logic gates
  - Gate leakage 5 nA/μm
  - Junction leakage negligible

Solution

$$W_{\text{normal-V}_t} = \left(50\times10^6\right)\left(12\lambda\right)\left(0.025\mu\text{m}/\lambda\right)\left(0.05\right) = 0.75\times10^6\ \mu\text{m}$$

$$W_{\text{high-V}_t} = \left[\left(50\times10^6\right)\left(12\lambda\right)\left(0.95\right)+\left(950\times10^6\right)\left(4\lambda\right)\right]\left(0.025\mu\text{m}/\lambda\right) = 109.25\times10^6\ \mu\text{m}$$

$$I_{sub} = \left[W_{\text{normal-V}_t}\times100\ \text{nA}/\mu\text{m}+W_{\text{high-V}_t}\times10\ \text{nA}/\mu\text{m}\right]/2 = 584\ \text{mA}$$

$$I_{gate} = \left[\left(W_{\text{normal-V}_t}+W_{\text{high-V}_t}\right)\times5\ \text{nA}/\mu\text{m}\right]/2 = 275\ \text{mA}$$

$$\text{P}_{static} = \left(584\ \text{mA}+275\ \text{mA}\right)\left(1.0\ \text{V}\right) = 859\ \text{mW}$$

Subthreshold Leakage

- For V_{ds} > 50 mV

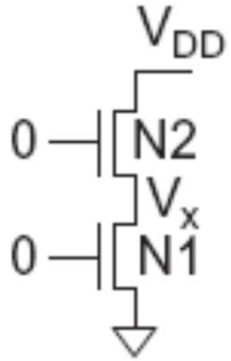
$$I_{sub} \approx I_{off} 10^{\frac{V_{gs} + \eta(V_{ds} - V_{DD}) - k_\gamma V_{sb}}{S}}$$

- I_{off} = leakage at V_{gs} = 0, V_{ds} = V_{DD}

Typical values in 65 nm
I_{off} = 100 nA/μm @ V_t = 0.3 V
I_{off} = 10 nA/μm @ V_t = 0.4 V
I_{off} = 1 nA/μm @ V_t = 0.5 V
η = 0.1
k_γ = 0.1
S = 100 mV/decade

Stack Effect

- Series OFF transistors have less leakage
  - $V_x > 0$, so N2 has negative V_{gs}



$$I_{sub} = \underbrace{I_{off}\,10^{\frac{\eta(V_x - V_{DD})}{S}}}_{N1} = \underbrace{I_{off}\,10^{\frac{-V_x + \eta((V_{DD} - V_x) - V_{DD}) - k_\gamma V_x}{S}}}_{N2}$$

$$V_x = \frac{\eta V_{DD}}{1 + 2\eta + k_\gamma}$$

$$I_{sub} = I_{off}\,10^{\frac{-\eta V_{DD}\left(\frac{1+\eta+k_\gamma}{1+2\eta+k_\gamma}\right)}{S}} \approx I_{off}\,10^{\frac{-\eta V_{DD}}{S}}$$

  - Leakage through 2-stack reduces ~10x
  - Leakage through 3-stack reduces further

Leakage Control

- Leakage and delay trade off
  - Aim for low leakage in sleep and low delay in active mode
- To reduce leakage:
  - Increase V_t: *multiple V_t*
    - Use low V_t only in critical circuits
  - Increase V_s: *stack effect*
    - *Input vector control* in sleep
  - Decrease V_b
    - *Reverse body bias* in sleep
    - Or forward body bias in active mode

Gate Leakage

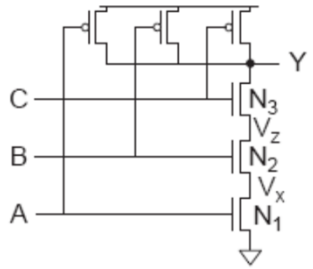
- Extremely strong function of t_{ox} and V_{gs}
  - Negligible for older processes
  - Approaches subthreshold leakage at 65 nm and below in some processes
- An order of magnitude less for pMOS than nMOS
- Control leakage in the process using t_{ox} > 10.5 Å
  - High-k gate dielectrics help
  - Some processes provide multiple t_{ox}
    - e.g. thicker oxide for 3.3 V I/O transistors
- Control leakage in circuits by limiting V_{DD}

NAND3 Leakage Example

- 100 nm process

I_{gn} = 6.3 nA I_{gp} = 0

I_{offn} = 5.63 nA I_{offp} = 9.3 nA



Input State (ABC)	I_{sub}	I_{gate}	I_{total}	V_x	V_z
000	0.4	0	0.4	stack effect	stack effect
001	0.7	0	0.7	stack effect	$V_{DD} - V_t$
010	0.7	1.3	2.0	intermediate	intermediate
011	3.8	0	3.8	$V_{DD} - V_t$	$V_{DD} - V_t$
100	0.7	6.3	7.0	0	stack effect
101	3.8	6.3	10.1	0	$V_{DD} - V_t$
110	5.6	12.6	18.2	0	0
111	28	18.9	46.9	0	0

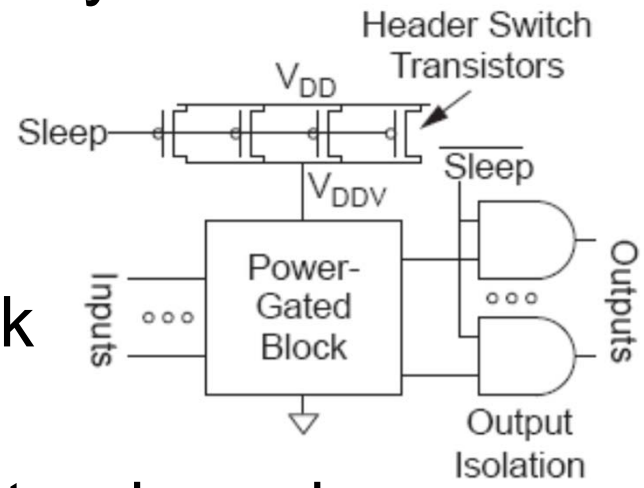
Data from [Lee03]

Junction Leakage

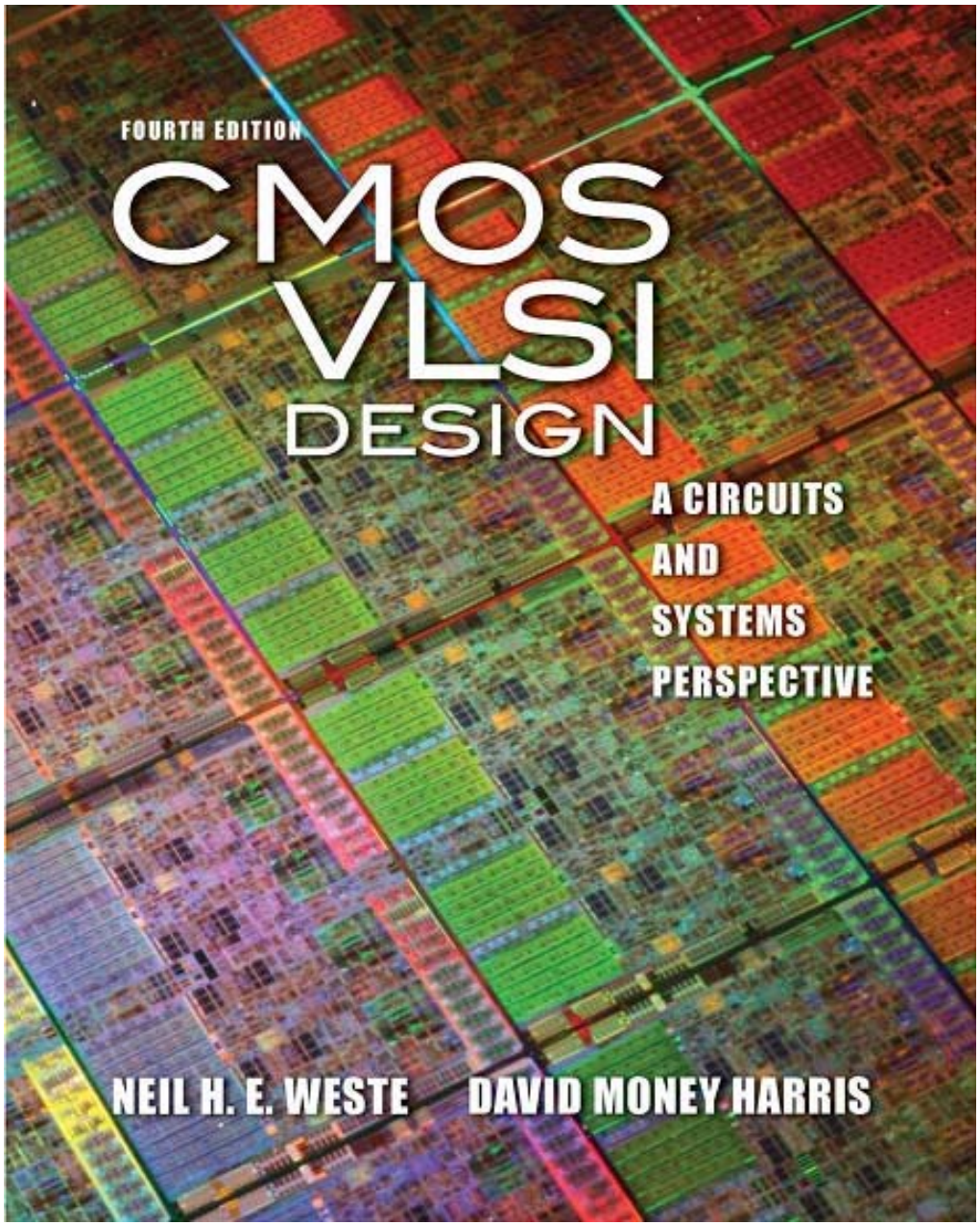
- From reverse-biased p-n junctions
  - Between diffusion and substrate or well
- Ordinary diode leakage is negligible
- Band-to-band tunneling (BTBT) can be significant
  - Especially in high-V_t transistors where other leakage is small
  - Worst at $V_{db} = V_{DD}$
- Gate-induced drain leakage (GIDL) exacerbates
  - Worst for $V_{gd} = -V_{DD}$ (or more negative)

Power Gating

- Turn OFF power to blocks when they are idle to save leakage
  - Use virtual V_{DD} (V_{DDV})
  - Gate outputs to prevent invalid logic levels to next block



- Voltage drop across sleep transistor degrades performance during normal operation
  - Size the transistor wide enough to minimize impact
- Switching wide sleep transistor costs dynamic power
  - Only justified when circuit sleeps long enough



Lecture_9:
Combinational Circuit Design

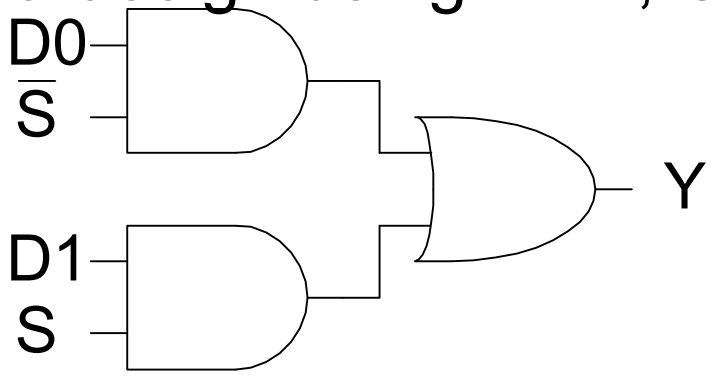
Outline

- Bubble Pushing
- Compound Gates
- Logical Effort Example
- Input Ordering
- Asymmetric Gates
- Skewed Gates
- Best P/N ratio

Example 1

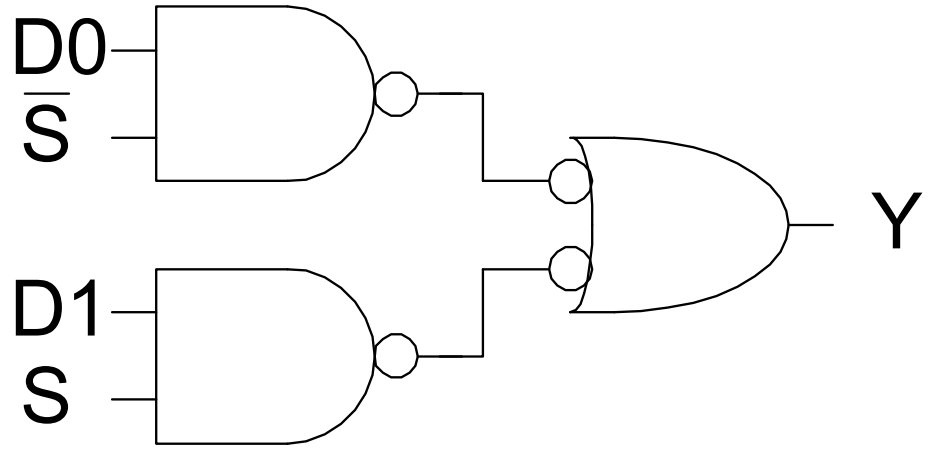
```
module mux(input  s, d0, d1,
           output y);

   assign y = s ? d1 : d0;  //Ternary Operator. If s is
                            //true y = d1 else y = d0
Endmodule
```

1) Sketch a design using AND, OR, and NOT gates.



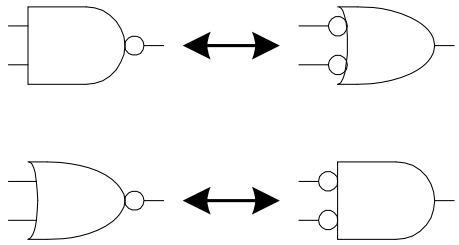
Example 2

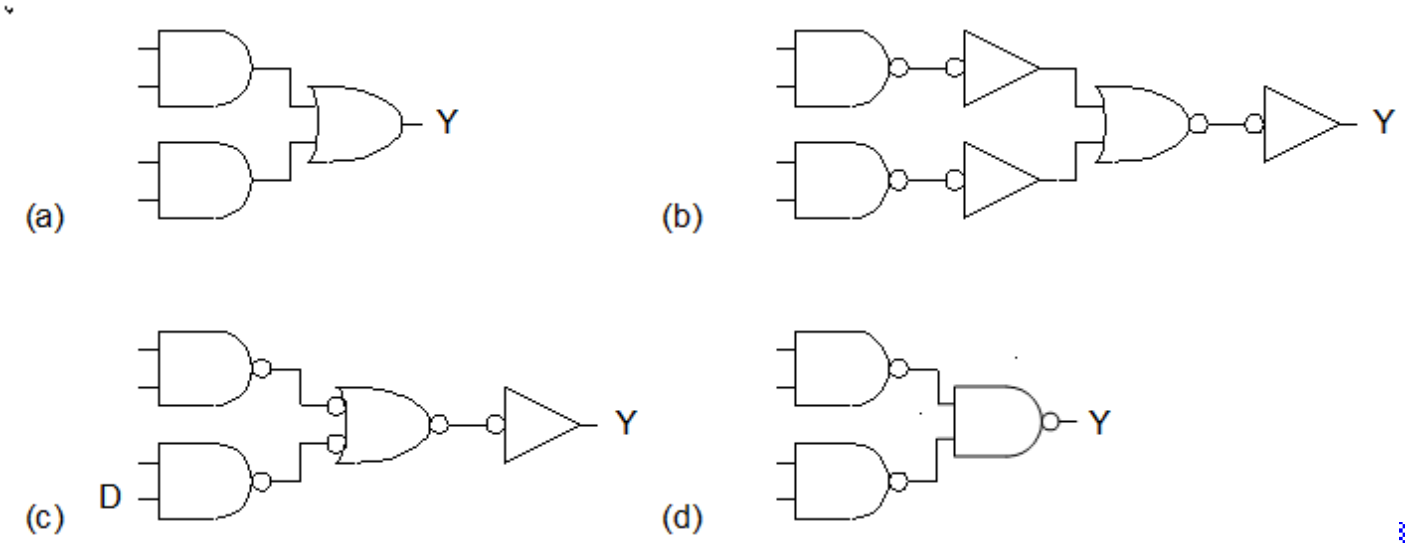
2) Sketch a design using NAND, NOR, and NOT gates.
Assume ~S is available.



Bubble Pushing

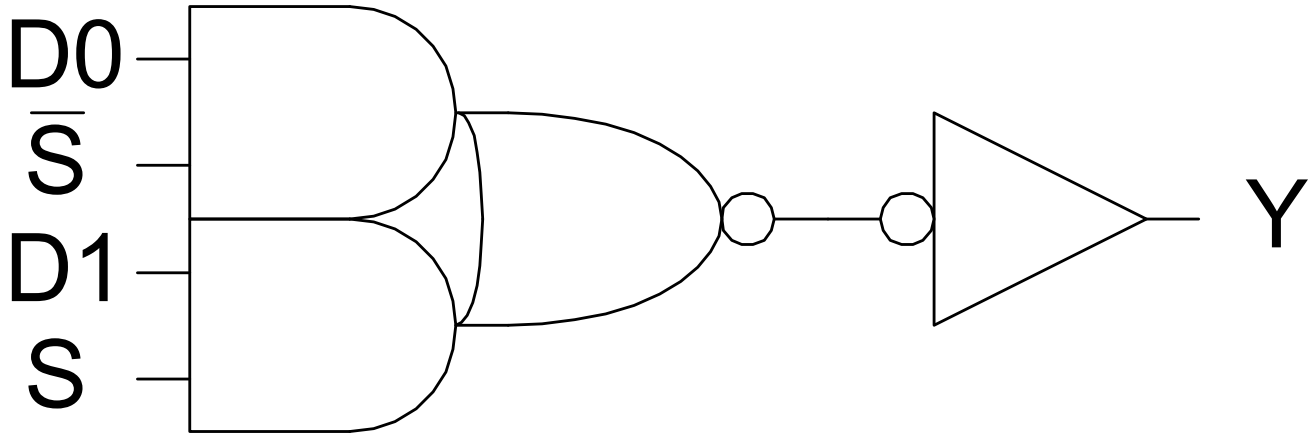
- Start with network of AND / OR gates
- Convert to NAND / NOR + inverters
- Push bubbles around to simplify logic
  - Remember DeMorgan's Law





Example 3

3) Sketch a design using one compound gate and one NOT gate. Assume ~S is available.

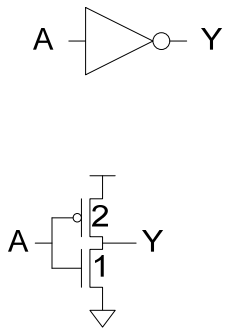


Compound Gates

- Logical Effort of compound gates

unit inverter

$Y = \overline{A}$

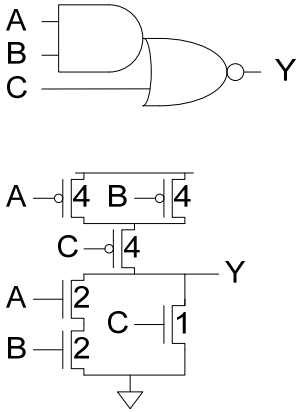


$g_A = 3/3$

$p = 3/3$

AOI21

$Y = \overline{A \cdot B + C}$



$g_A = 6/3$

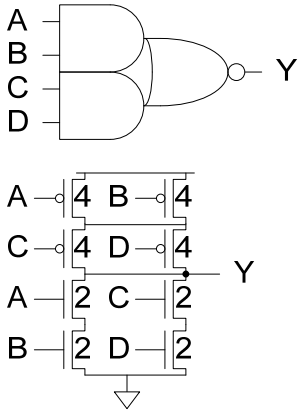
$g_B = 6/3$

$g_C = 5/3$

$p = 7/3$

AOI22

$Y = \overline{A \cdot B + C \cdot D}$



$g_A =$

$g_B =$

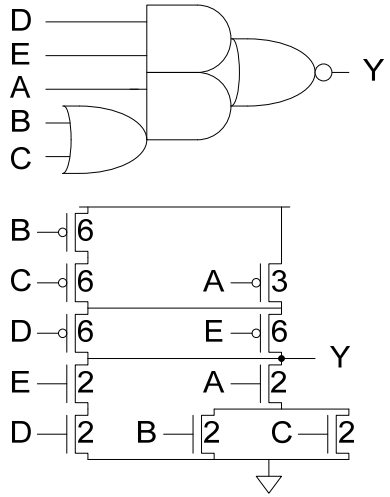
$g_C =$

$g_D =$

$p =$

Complex AOI

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



$g_A =$

$g_B =$

$g_C =$

$g_D =$

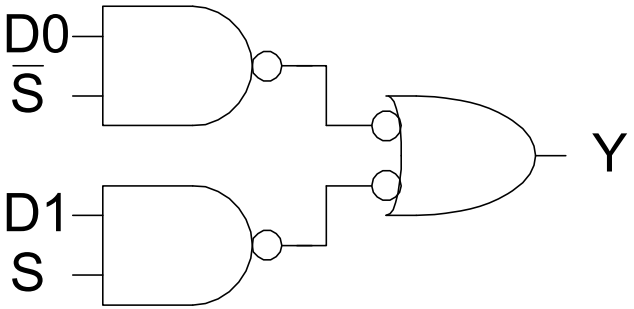
$g_E =$

$p =$

Example 4

- The multiplexer has a maximum input capacitance of 16 units on each input. It must drive a load of 160 units. Estimate the delay of the two designs.

H =



$P =$

$G =$

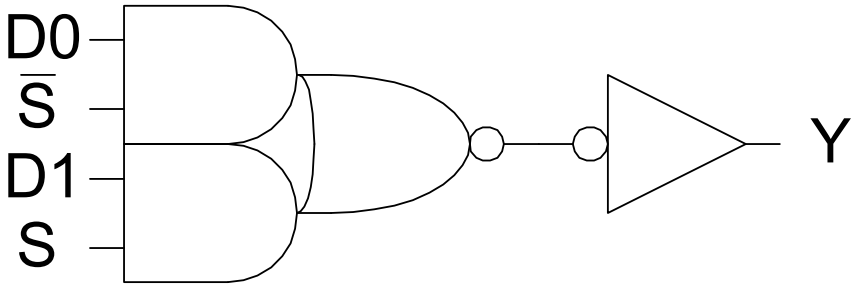
$F =$

$\hat{f} =$

$D =$

B =

N =



$P =$

$G =$

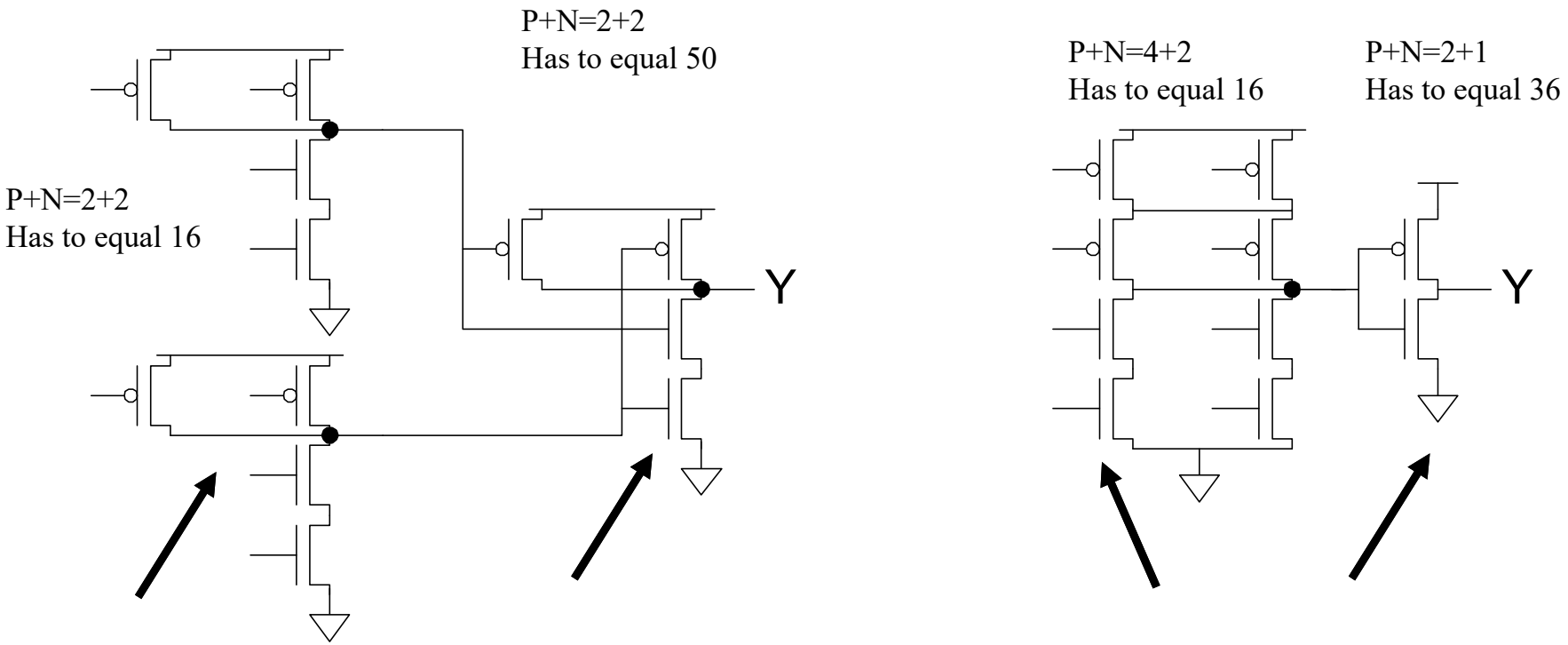
$F =$

$\hat{f} =$

$D =$

Example 5

- Annotate your designs with transistor sizes that achieve this delay.

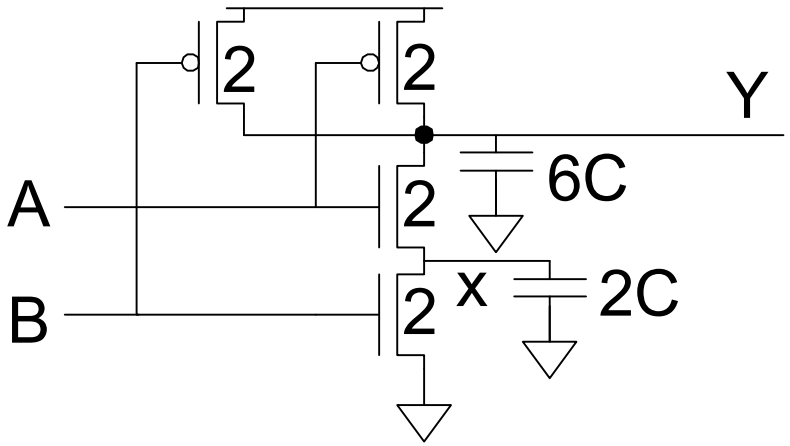


Input Order

- Our parasitic delay model was too simple
  - Calculate parasitic delay for Y falling
    - If A arrives latest?
    - If B arrives latest?

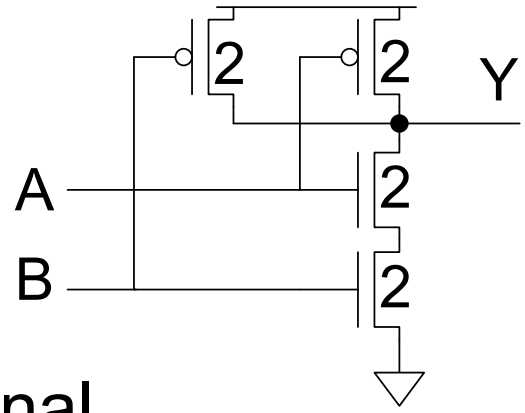
$$t_{pd} = 6C * \left(\frac{R}{2} + \frac{R}{2}\right)/3RC$$

$$t_{pd} = (2C * R/2 + 6C * \left(\frac{R}{2} + \frac{R}{2}\right))/3RC$$



Inner & Outer Inputs

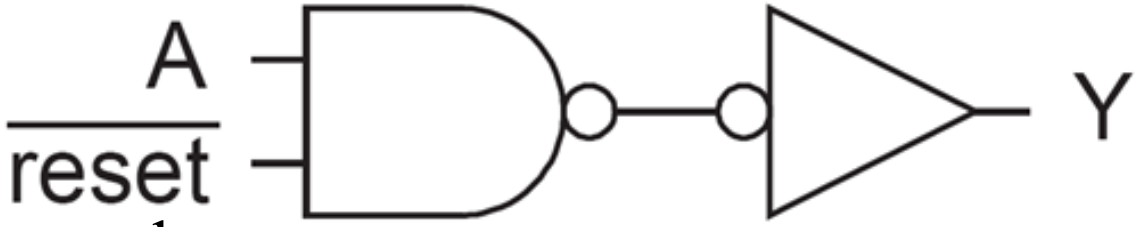
- *Inner* input is closest to output (A)
- *Outer* input is closest to rail (B)



- If input arrival time is known
  - Connect latest input to inner terminal

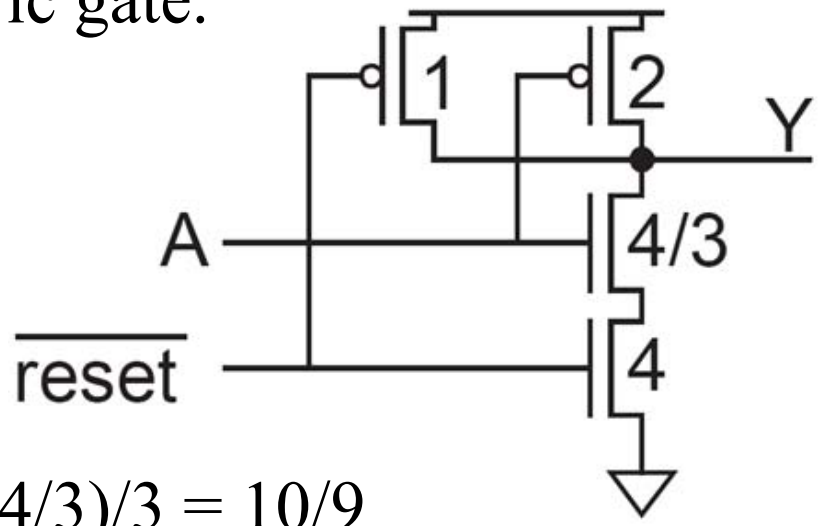
Asymmetric Gates

Buffer
Reset asserted y=0
Required to reset less frequently



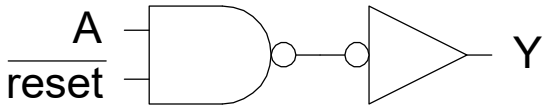
A is most critical, go for Asymmetric gate.

- Make it inner
- Less gate capacitance
- Reset to a wider nMOS, Less R
- Reset narrower pMOS, Less C
- Series nMOS R =unity
- R/4 + R/(4/3) = R and $g_A = (2+4/3)/3 = 10/9$
- As the reset nMOS W gets larger, g_A *becomes closer to unity*



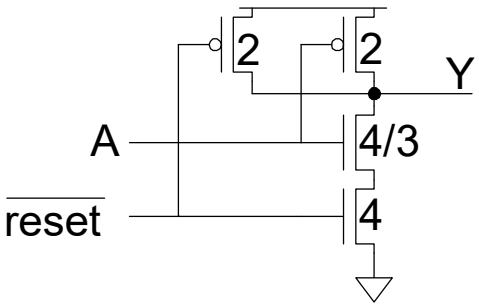
Asymmetric Gates

- Asymmetric gates favor one input over another
- Ex: suppose input A of a NAND gate is most critical
  - Use smaller transistor on A (less capacitance)
  - Boost size of noncritical input
  - So total resistance is same



- g_A =
- g_B =
- $g_{total} = g_A + g_B$ =

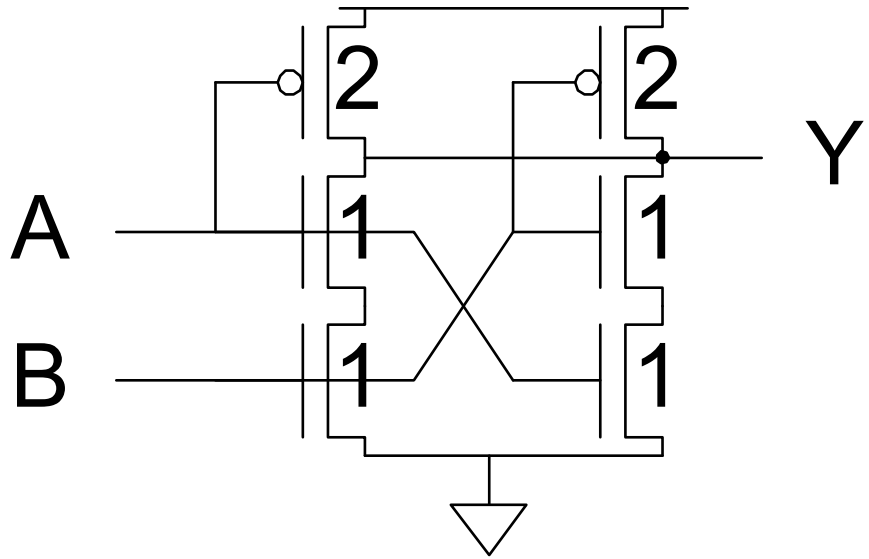
$$R_{PD} = \frac{1}{4} + \frac{3}{4} = 1$$



- Asymmetric gate approaches g = 1 on critical input
- But total logical effort goes up

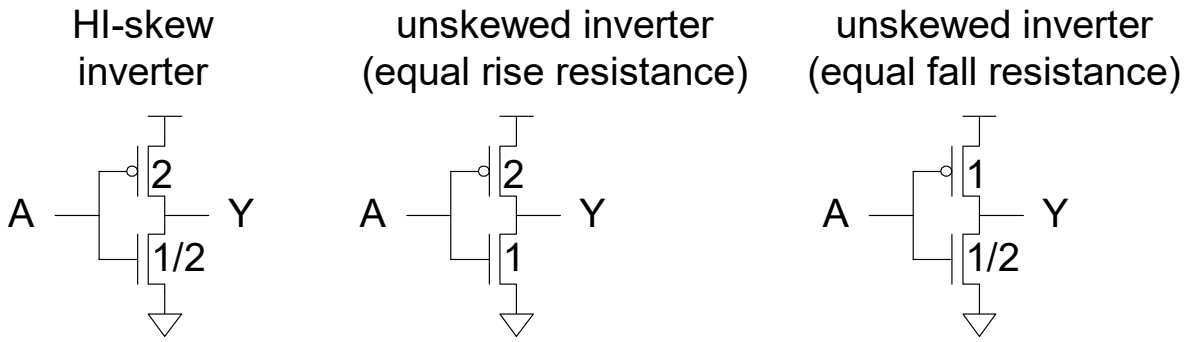
Symmetric Gates

- Inputs can be made perfectly symmetric



Skewed Gates

- Skewed gates favor one edge over another
- Ex: suppose rising output of inverter is most critical
  - Downsize noncritical nMOS transistor



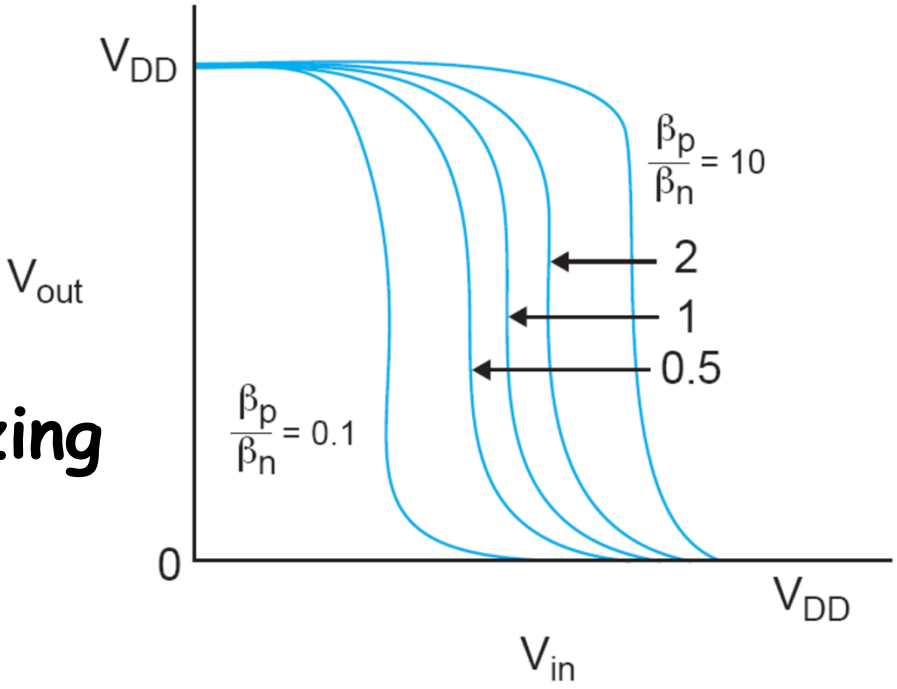
- Calculate logical effort by comparing to unskewed inverter with same effective resistance on that edge.
  - g_u =
  - g_d =

HI- and LO-Skew

$$when\ \frac{\beta_p}{\beta_n} > 1\ HI\ skew$$

Favors rising transition

Done by downsizing nMOS



Skewing is done by downsizing MOSs by a factor of 2

$$when\ \frac{\beta_p}{\beta_n} < 1\ LO\ skew$$

Favors falling transition

Done by downsizing pMOS

$$\beta_n = (\frac{W}{L})_n$$

$$\beta_p = (\frac{W}{L})_p$$

HI- and LO-Skew

- Def: Logical effort of a skewed gate for a particular transition is the ratio of the input capacitance of that gate to the input capacitance of an unskewed inverter delivering the same output current for the same transition.

- Skewed gates reduce size of noncritical transistors
  - HI-skew gates favor rising output (small nMOS)
  - LO-skew gates favor falling output (small pMOS)
- Logical effort is smaller for favored direction
- But larger for the other direction

HI- and LO-Skew

In calculating $g_u \; of \; a \; complex \; gate$:
Draw the unskewed inverter (2:1) whose pull-up resistance is equal to the equivalent resistance of the pull-up network of the skewed gate.

Then $g_u = \dfrac{input \; capacitance \; of \; the \; skewed \; gate}{input \; capacitance \; of \; the \; unskewed \; invrter}$

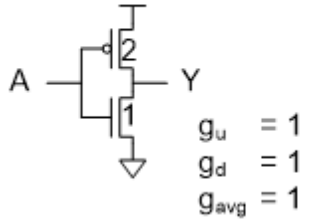
In calculating $g_d \; of \; a \; complex \; gate$:
Draw the unskewed inverter (2:1) whose pull-down resistance is equal to the equivalent resistance of the pull-down network of the skewed gate.

Then $g_d = \dfrac{input \; capacitance \; of \; the \; skewed \; gate}{input \; capacitance \; of \; the \; unskewed \; invrter}$

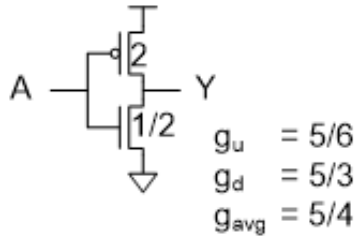
Calculations of $g'_u s$ and $g'_d s$

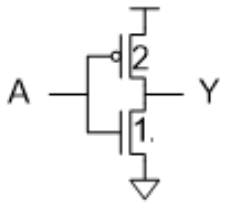
Inverters

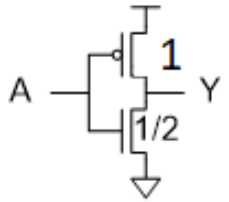
unskewed



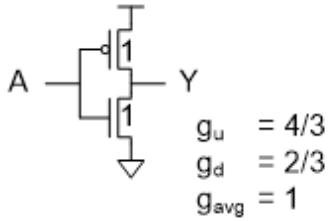
HI-skew

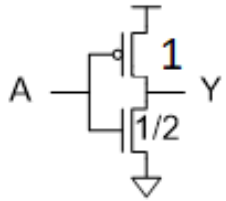




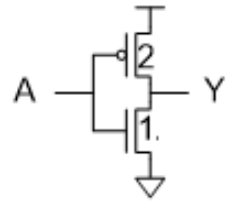


LO-skew





Equal rise time

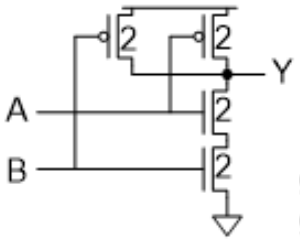


Equal fall time

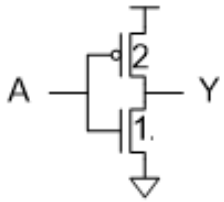
Calculations of $g_u's$ and $g_d's$

NAND gates

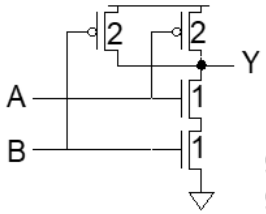
Unskewed



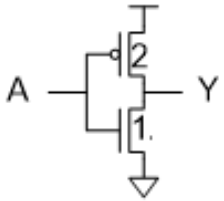
$g_u = 4/3$
$g_d = 4/3$
$g_{avg} = 4/3$

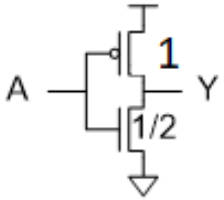


HI-skewed

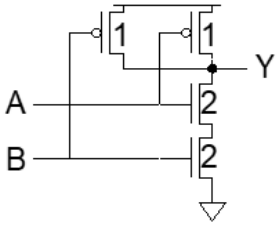


$g_u = 1$
$g_d = 2$
$g_{avg} = 3/2$

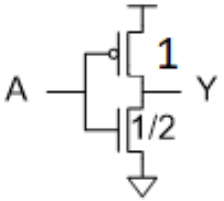




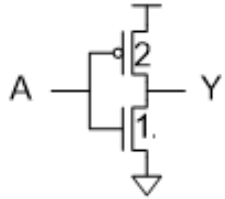
LO-skewed



$g_u = 2$
$g_d = 1$
$g_{avg} = 3/2$



Equal rise time

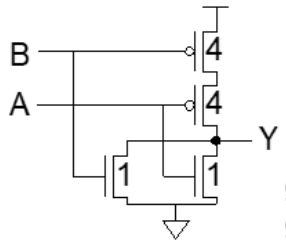


Equal fall time

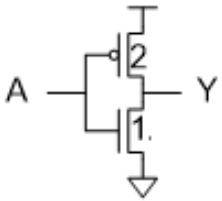
Calculations of $g'_u s$ and $g'_d s$

NOR gates

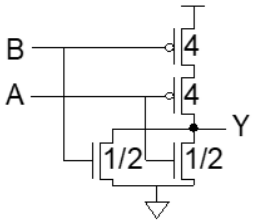
Unskewed



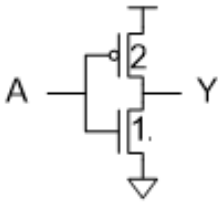
$g_u = 5/3$
$g_d = 5/3$
$g_{avg} = 5/3$

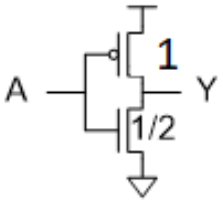


HI-skewed

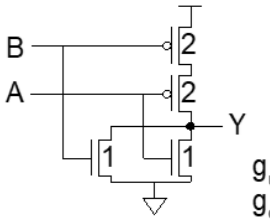


$g_u = 3/2$
$g_d = 3$
$g_{avg} = 9/4$

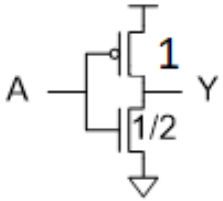


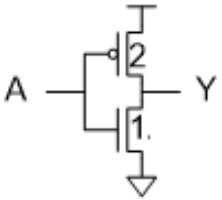


LO-skewed



$g_u = 2$
$g_d = 1$
$g_{avg} = 3/2$

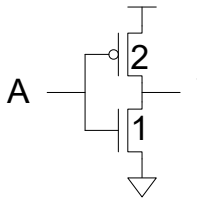
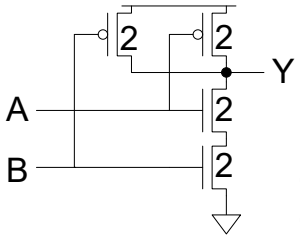
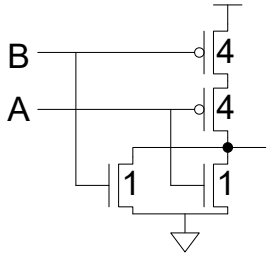
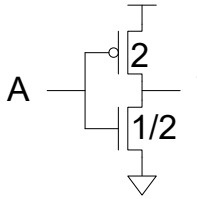
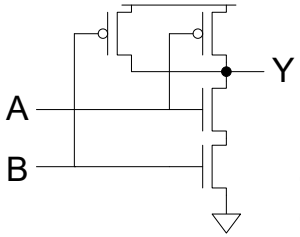
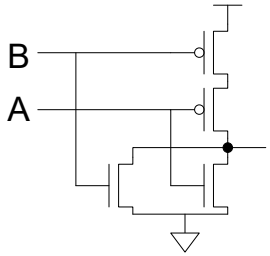
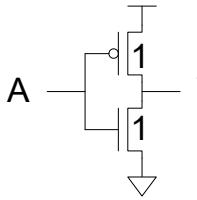
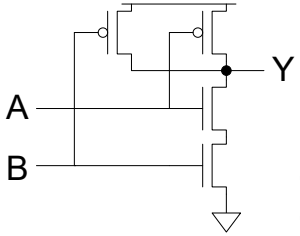
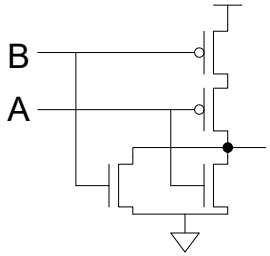




Equal rise time

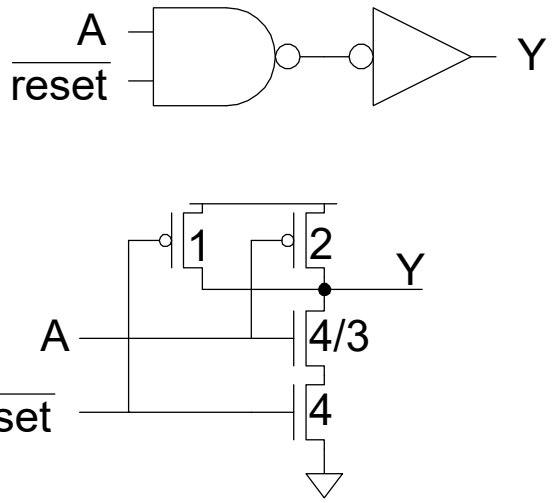
Equal fall time

Catalog of Skewed Gates

	Inverter	NAND2	NOR2
unskewed	 $g_u = 1$ $g_d = 1$ $g_{avg} = 1$	 $g_u = 4/3$ $g_d = 4/3$ $g_{avg} = 4/3$	 $g_u = 5/3$ $g_d = 5/3$ $g_{avg} = 5/3$
HI-skew	 $g_u = 5/6$ $g_d = 5/3$ $g_{avg} = 5/4$	 $g_u =$ $g_d =$ $g_{avg} =$	 $g_u =$ $g_d =$ $g_{avg} =$
LO-skew	 $g_u = 4/3$ $g_d = 2/3$ $g_{avg} = 1$	 $g_u =$ $g_d =$ $g_{avg} =$	 $g_u =$ $g_d =$ $g_{avg} =$

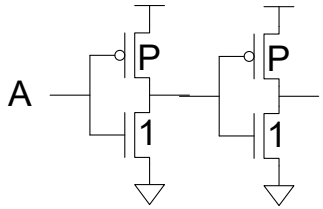
Asymmetric Skew

- Combine asymmetric and skewed gates
  - Downsize noncritical transistor on unimportant input
  - Reduces parasitic delay for critical input



Best P/N Ratio

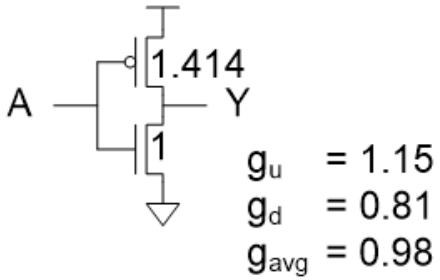
- We have selected P/N ratio for unit rise and fall resistance (μ = 2-3 for an inverter). $\mu = \frac{\mu_n}{\mu_p} = 2$
- Alternative: choose ratio for least average delay
  - Ex: inverter
    - Delay driving identical inverter

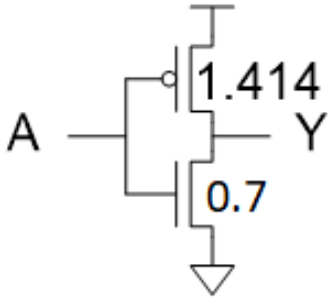


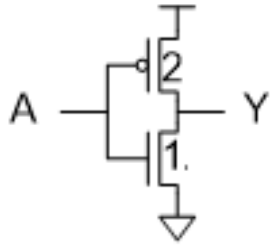
- $t_{pdf} = 2C(P+1).\ R$
- $t_{pdr} = 2C(P+1).\ R(\mu/P)$
- $t_{pd} = 1/2(t_{pdf} + t_{pdr}) = 1/2[2CR(P+1)(1+\mu/P)] = (P + 1 + \mu + \mu/P)CR$
- $dt_{pd} / dP = (1 - \mu/P^2) = 0$
- Least delay for $P = \sqrt{\mu}$

Best P/N Ratio

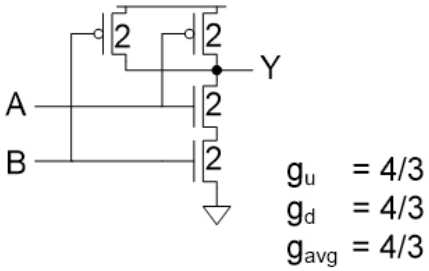
Inverters

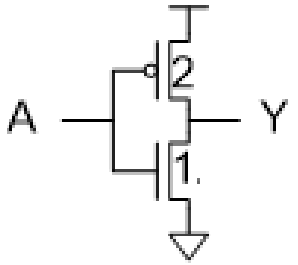


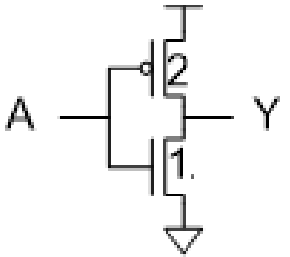




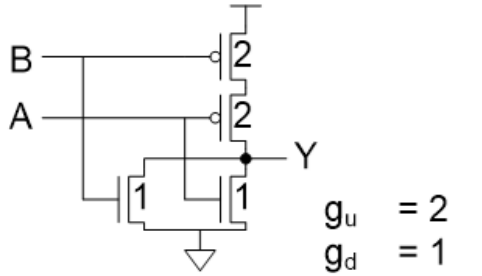
NAND gate

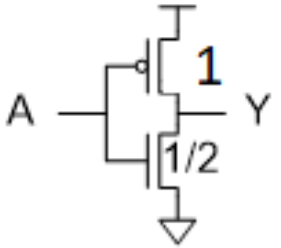


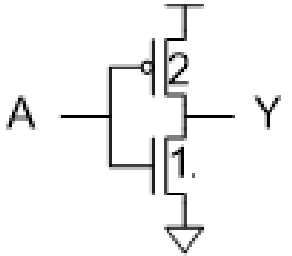




NOR gate







Equal rise time

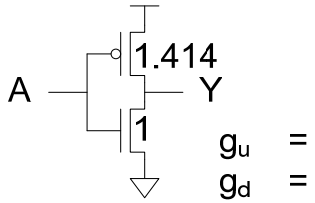
Equal fall time

P/N Ratios

- In general, best P/N ratio is sqrt of equal delay ratio.
  - Only improves average delay slightly for inverters
  - But significantly decreases area and power

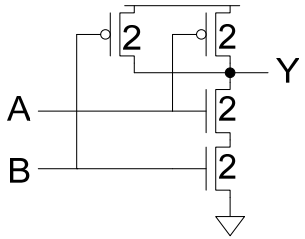
fastest P/N ratio

Inverter



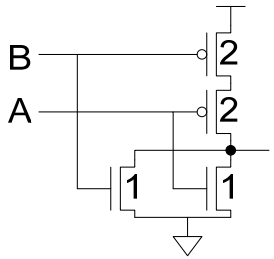
g_u =
g_d =
g_{avg} =

NAND2



g_u =
g_d =
g_{avg} =

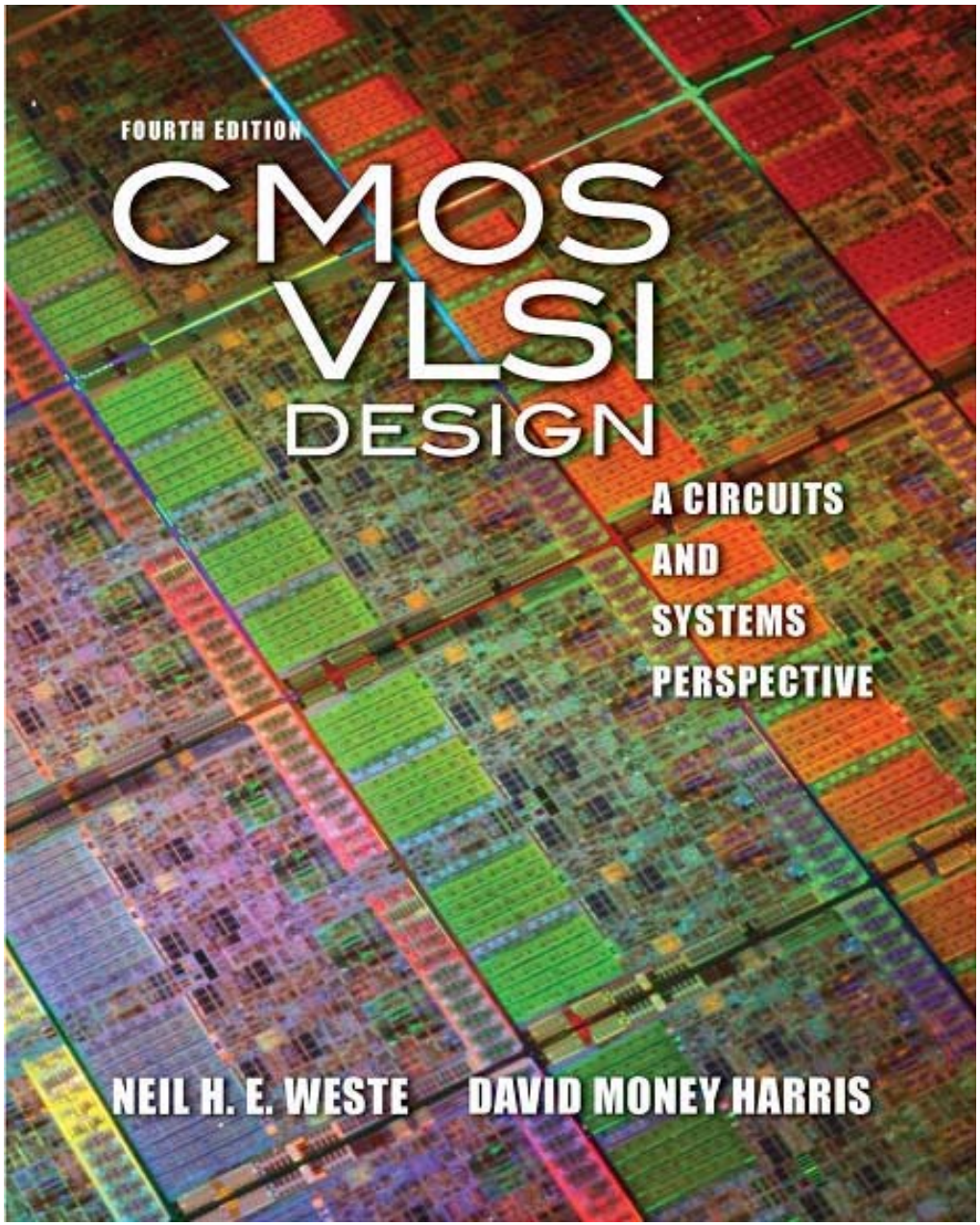
NOR2



g_u =
g_d =
g_{avg} =

Observations

- For speed:
  - NAND vs. NOR
  - Many simple stages vs. fewer high fan-in stages
  - Latest-arriving input
- For area and power:
  - Many simple stages vs. fewer high fan-in stages



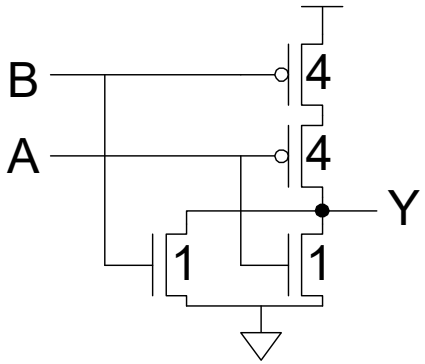
Lecture_10: Circuit Families

Outline

- Pseudo-nMOS Logic
- Dynamic Logic
- Pass Transistor Logic

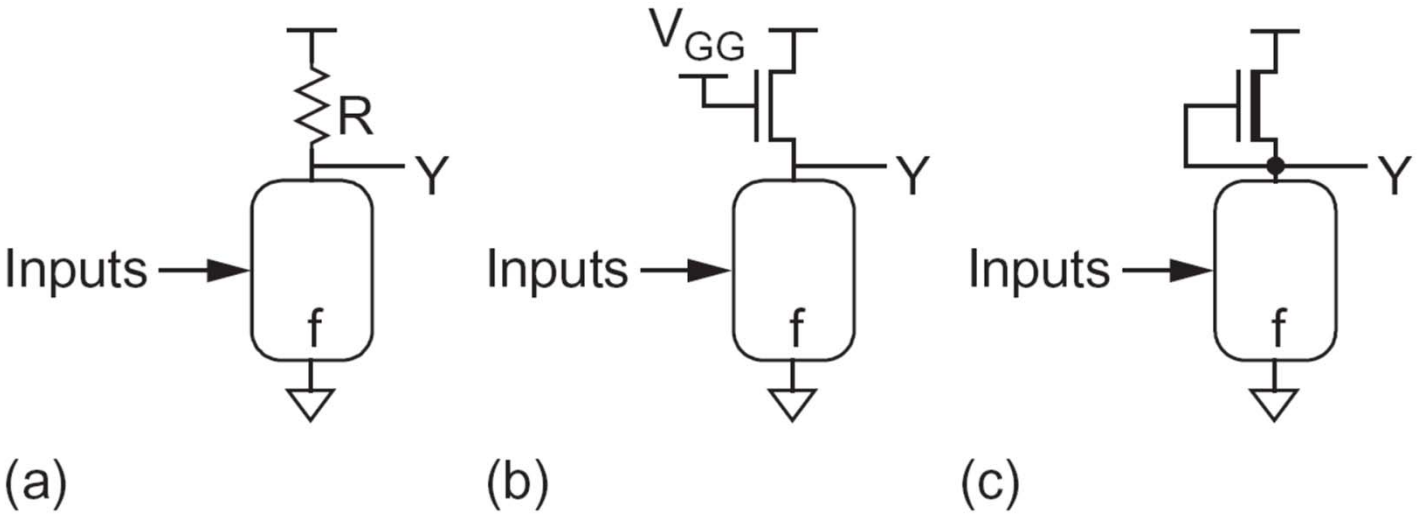
Introduction

- What makes a circuit fast?
  - $I = C\, dV/dt \rightarrow t_{pd} \propto (C/I)\, \Delta V$
  - low capacitance
  - high current
  - small swing
- Logical effort is proportional to C/I
- pMOS are the enemy!
  - High capacitance for a given current
- Can we take the pMOS capacitance off the input?
- Various circuit families try to do this…



Ratioed circuits: nMOS Technology

- nMOS only Technology.
- Popular 1970 – to -1980 before CMOS.
- Pulldown network off, static load (R or T) pulls output high.
- Pulldown network on, PDN fights the always on static load.
- Enhancement nMOS requires additional Supply V_{GG} for strong V_{OH} , use instead depletion mode MOS

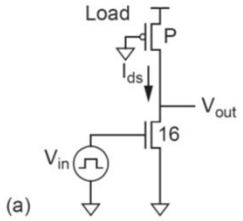


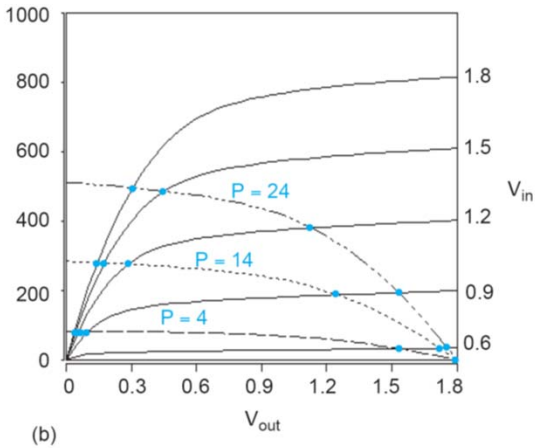
Pseudo-nMOS

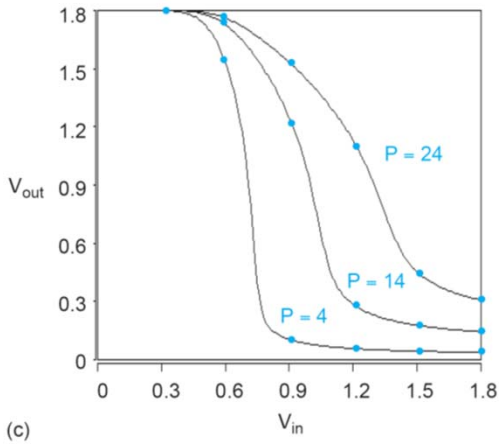
- In CMOS, use a pMOS that is always ON
- ***Ratio* issue**

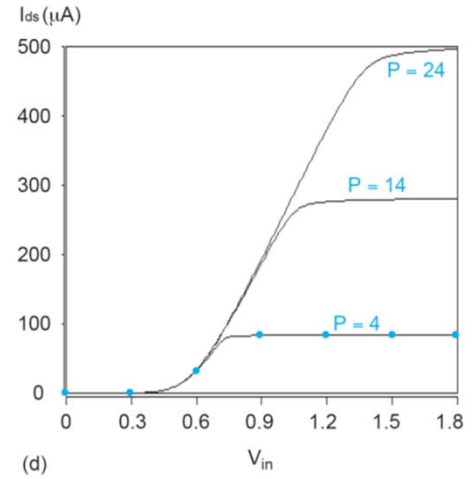
  Make pMOS about ¼ effective strength of pulldown network.

P = (2x16)/4 = 8

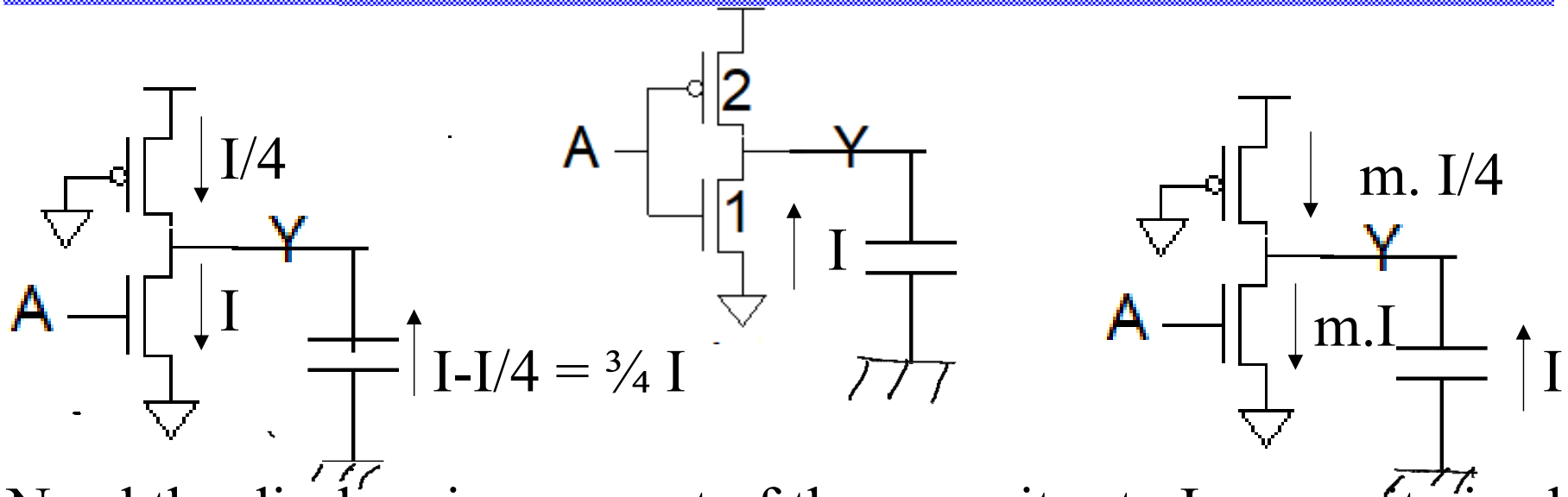








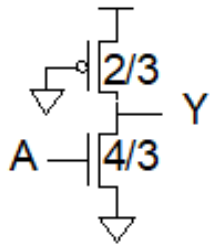
Pseudo-nMOS



Need the discharging current of the capacitor to I as a unit-sized inverter I. Required transistor size m to do so, keeping the pMOS transistor of ¼ the stregnth of the nMOS.

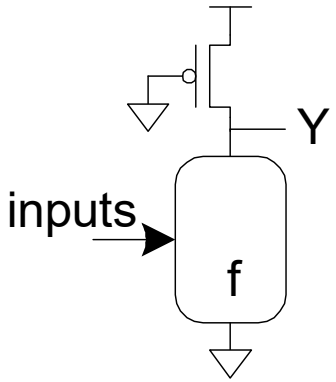
m . I – m . I/4 = I which gives m = 4/3

Which gives μ(4/3)∗ $\frac{1}{4} = \frac{2}{3}$

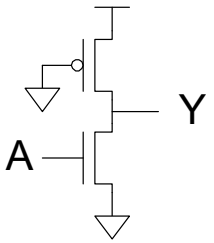


Pseudo-nMOS Gates

- Design for unit current on output to compare with unit inverter.
- pMOS fights nMOS

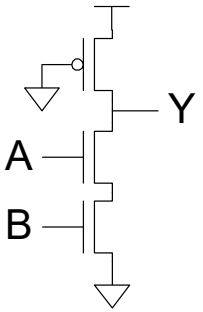


Inverter



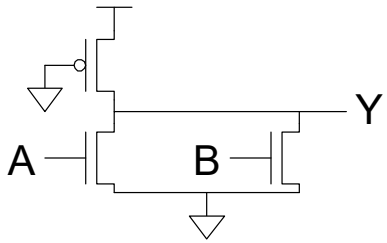
g_u =
g_d =
g_{avg} =
p_u =
p_d =
p_{avg} =

NAND2



g_u =
g_d =
g_{avg} =
p_u =
p_d =
p_{avg} =

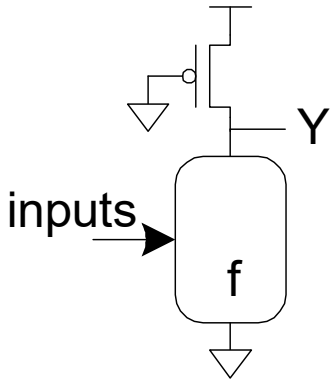
NOR2



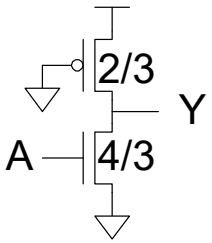
g_u =
g_d =
g_{avg} =
p_u =
p_d =
p_{avg} =

Pseudo-nMOS Gates

- Design for unit current on output to compare with unit inverter.
- pMOS fights nMOS

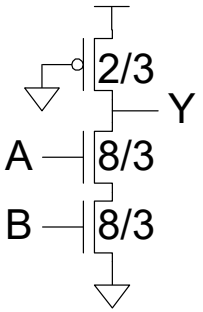


Inverter



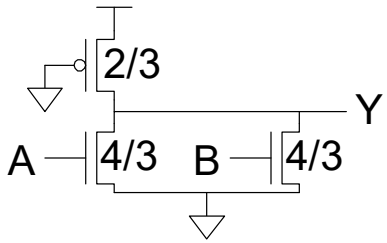
$g_u = 4/3$
$g_d = 4/9$
$g_{avg} = 8/9$
$p_u = 6/3$
$p_d = 6/9$
$p_{avg} = 12/9$

NAND2



$g_u = 8/3$
$g_d = 8/9$
$g_{avg} = 16/9$
$p_u = 10/3$
$p_d = 10/9$
$p_{avg} = 20/9$

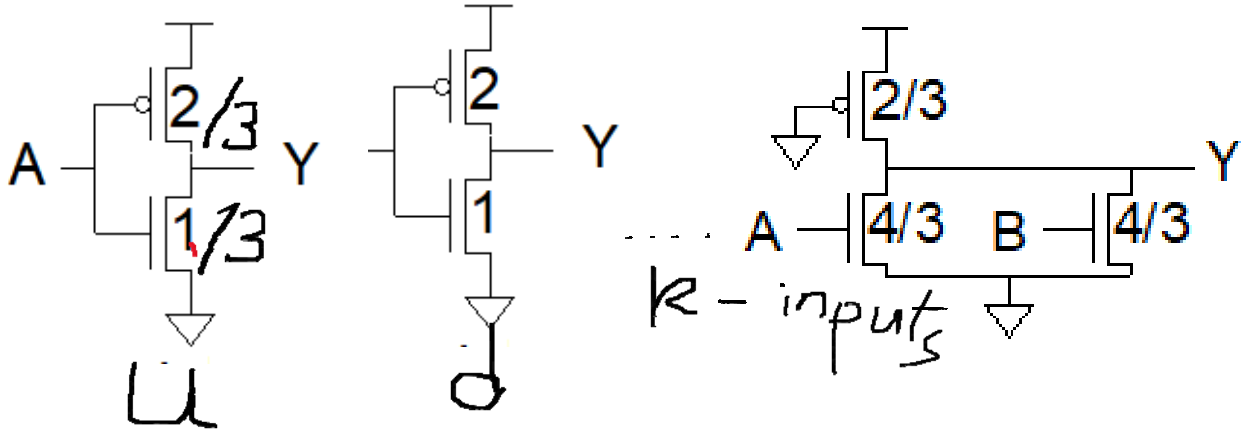
NOR2



$g_u = 4/3$
$g_d = 4/9$
$g_{avg} = 8/9$
$p_u = 10/3$
$p_d = 10/9$
$p_{avg} = 20/9$

Pseudo-nMOS Gates

Calculate g_{ave} and P_{ave} for k-input pseudo-nMOS NOR gate



$g_u = (4/3)/1 = 4/3$

$g_d = (4/3)/3 = 4/9$

$g_{ave} = ½(4/3 + 4/9) = 8/9$ independent of k

$P_u = (2/3 + kx4/3)/1$

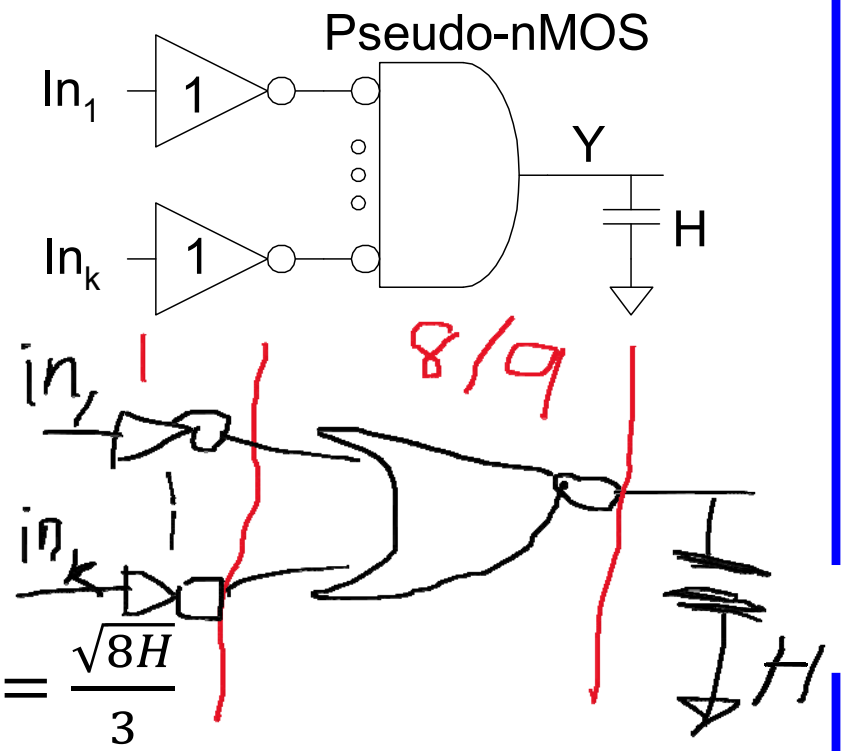
$P_d = (2/3 + kx4/3)/3$

$P_{ave} = ½ [2/3 + 4/3xk + 2/9 + 4/9xk) = 4/9 + 8k/9$

Pseudo-nMOS Design

- Ex: Design a k-input AND gate using pseudo-nMOS. Estimate the delay driving a fanout of H

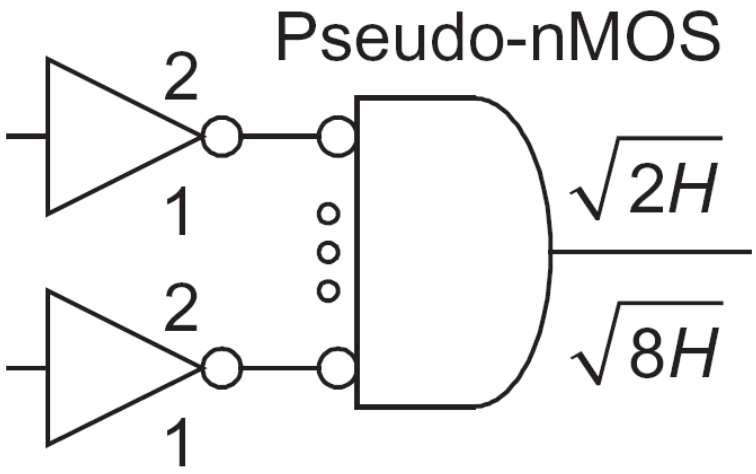
- G =
- F =
- P = 1 + (4+8k)/9 = (8k+13)/9
- N =
- D =



Which gives : $C_{in} = \frac{g\,C_{out}}{\hat{f}} = \frac{\frac{8}{9}H}{\frac{2\sqrt{2H}}{3}} = \frac{\sqrt{8H}}{3}$

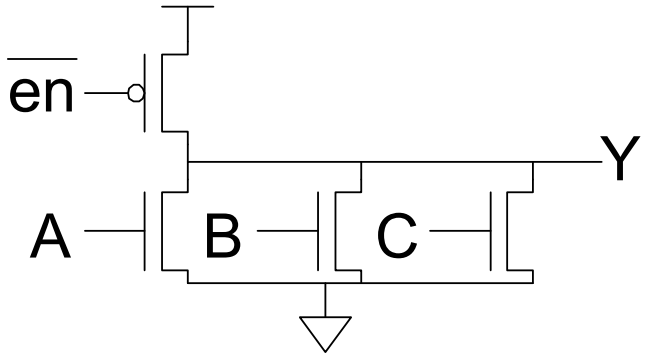
Pseudo-nMOS Design

Since the unit-sized inverter has an input capacitance of 3 units, the sizing of the nMOS NOR gate transistors should be $\sqrt{8H}$ and the size of the pMOS NOR gate would be $2.(\sqrt{8H})/4$ which makes it one fourth the nMOS strength.

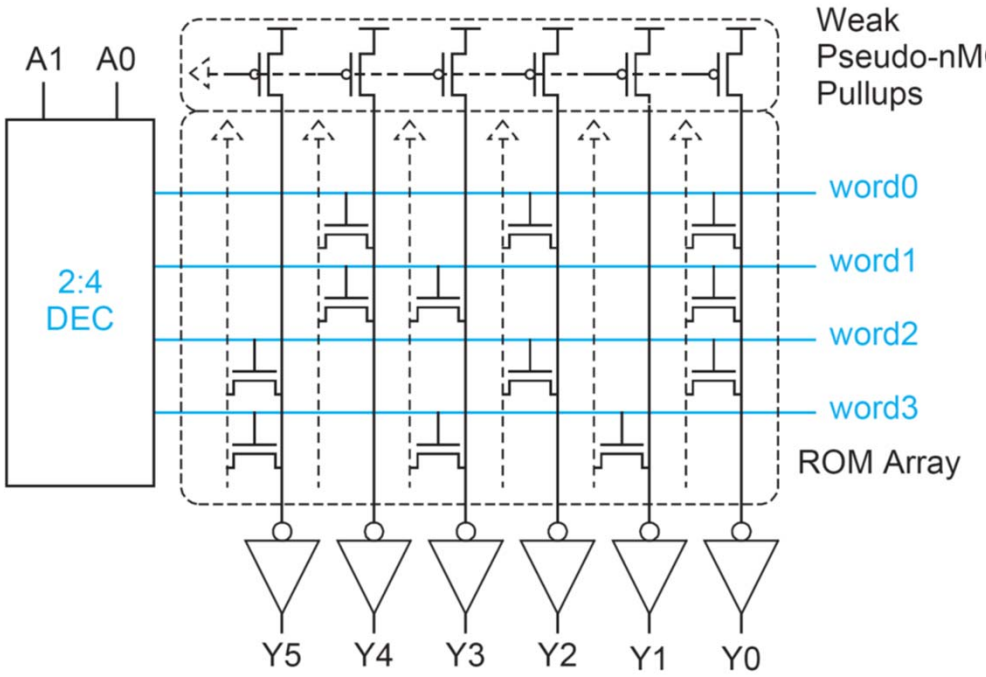


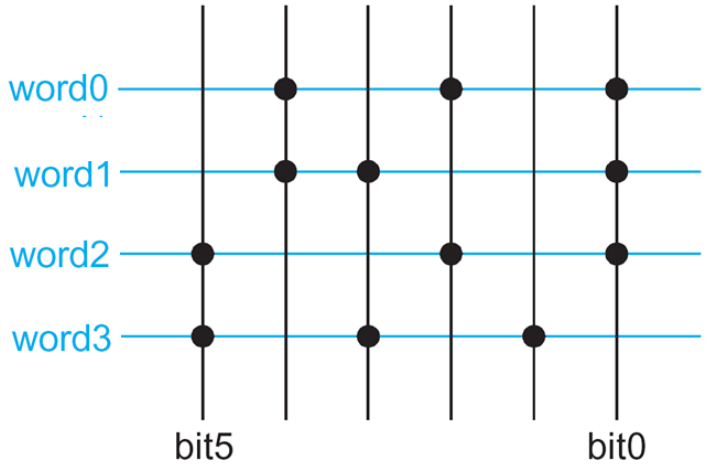
Pseudo-nMOS Power

- Pseudo-nMOS draws power whenever Y = 0
  - Called static power $P = I_{DD}V_{DD}$
  - A few mA / gate * 1M gates would be a problem
  - Explains why nMOS went extinct
- Use pseudo-nMOS sparingly for wide NORs
- Turn off pMOS when not in use



Pseudo nMOS ROM



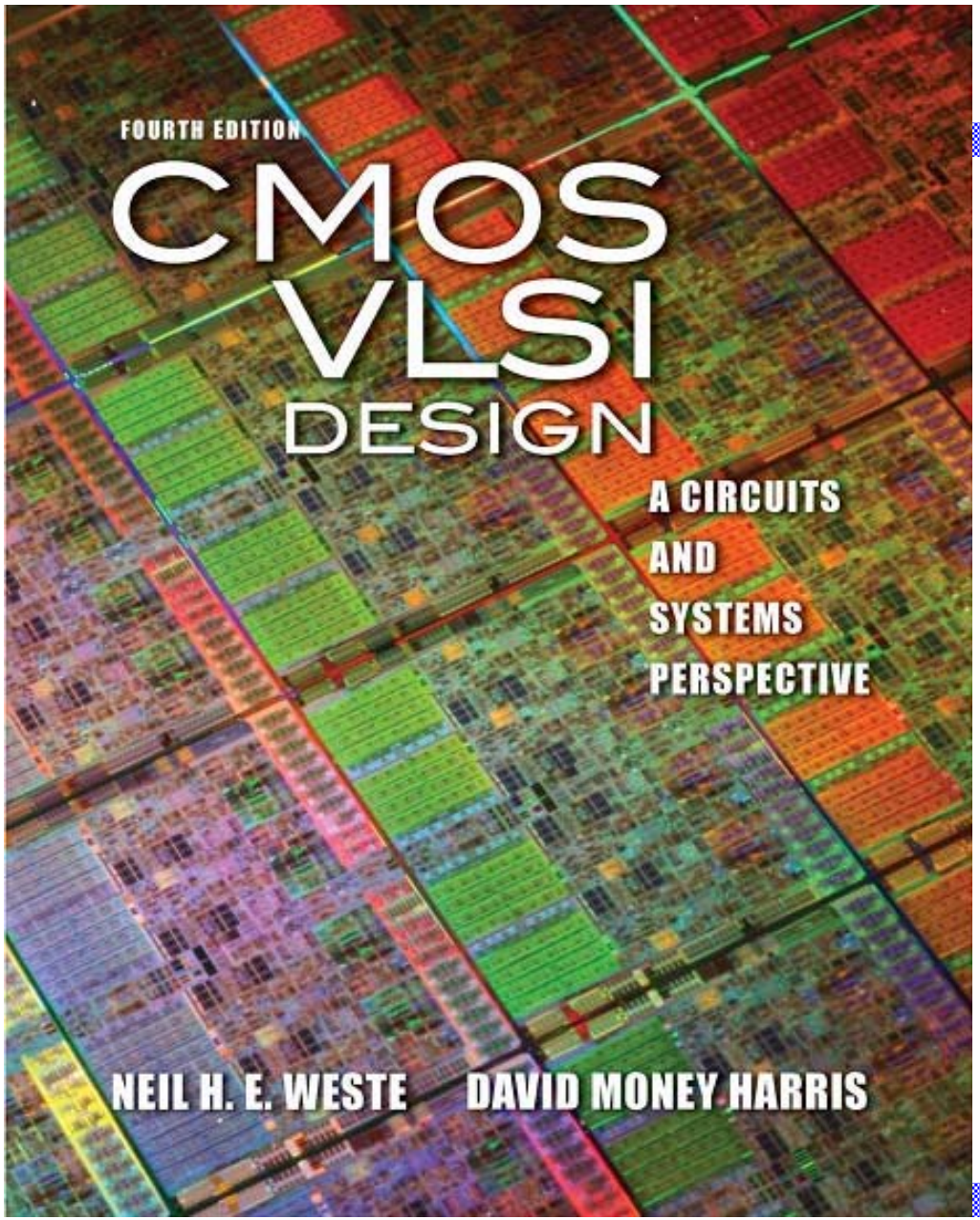


Ratio Example

- The chip contains a 32 word x 48 bit ROM
  - Uses pseudo-nMOS decoder and bitline pullups
  - On average, one wordline and 24 bitlines are high
- Find static power drawn by the ROM
  - $I_{on\text{-}p}$ = 36 μA, V_{DD} = 1.0 V
- Solution:

$$P_{\text{pull-up}} =$$

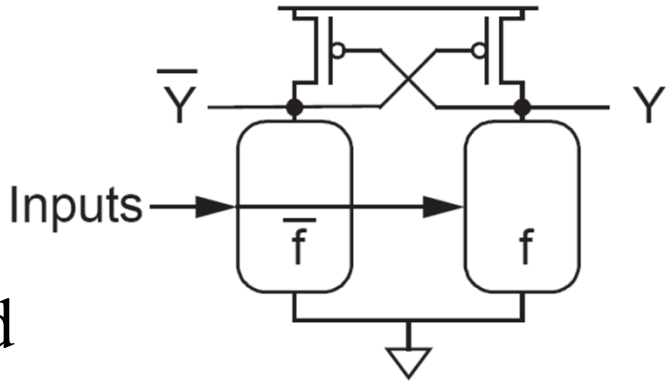
$$P_{\text{static}} =$$



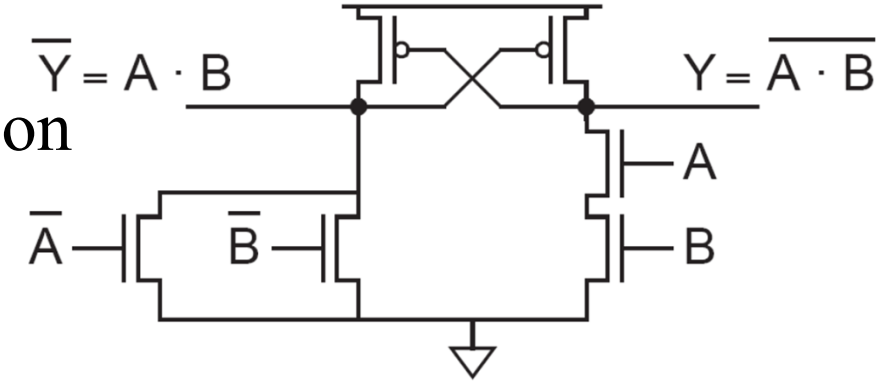
Lecture_11: Circuits Families

Differential Cascode Voltage Switch Logic (DCVSL)

- Benefit from ratioed logic without static power.
- True and complementary inputs and outputs.
- Use a pair of complementary PD network.
- PMOS size is important, contention during transition.



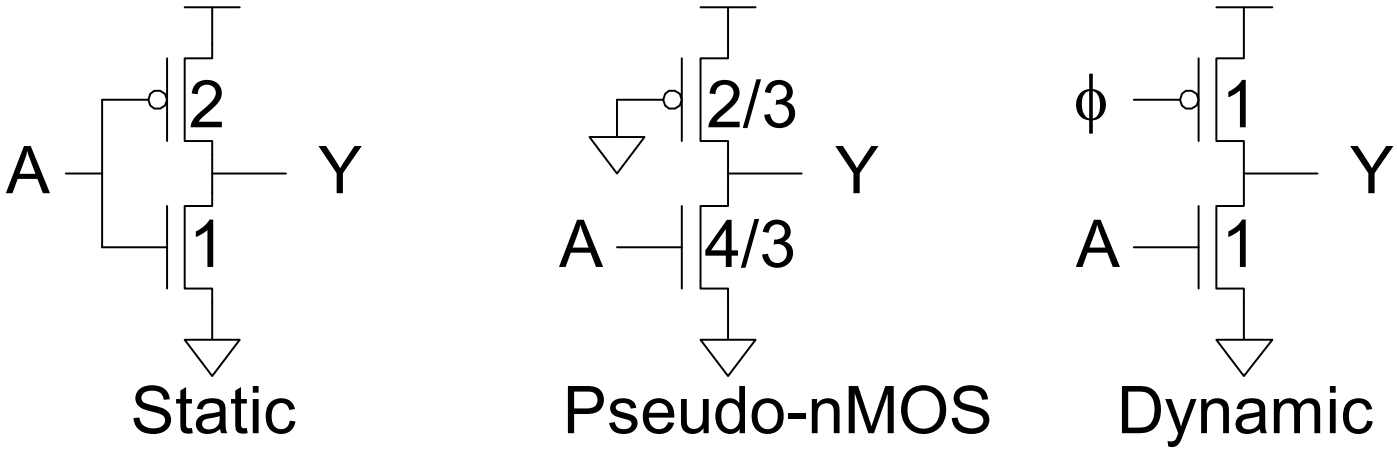
(a)

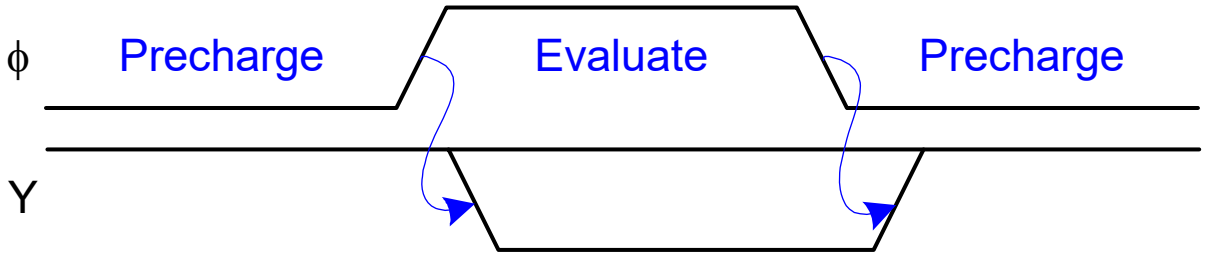


(b)

Dynamic Logic

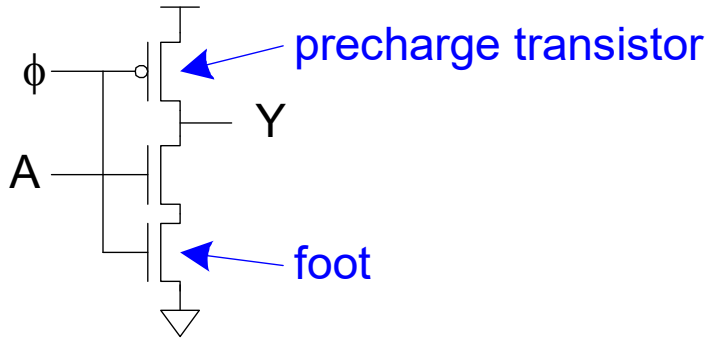
- *Dynamic* gates uses a clocked pMOS pullup
- Two modes: *precharge* and *evaluate*

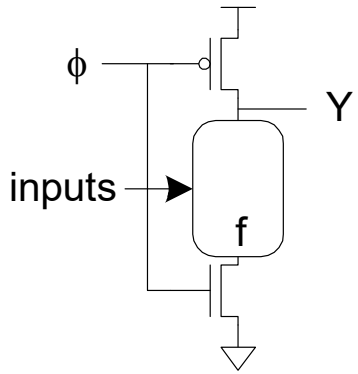




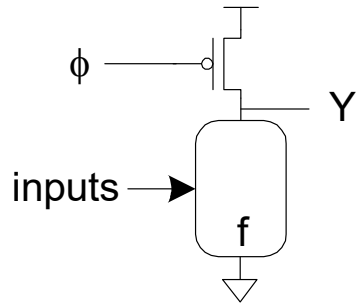
The Foot

- What if pulldown network is ON during precharge?
- Use series evaluation transistor to prevent fight.



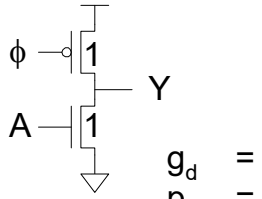
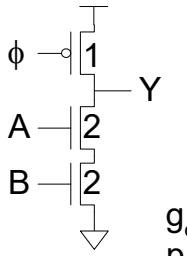
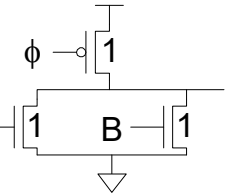
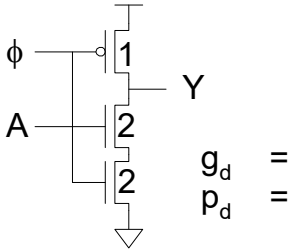
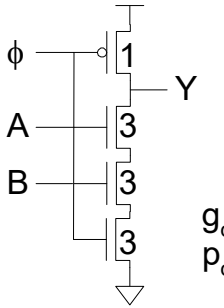
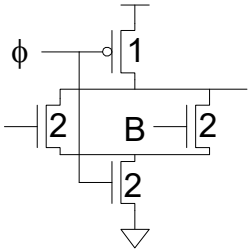


footed



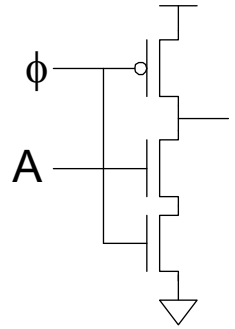
unfooted

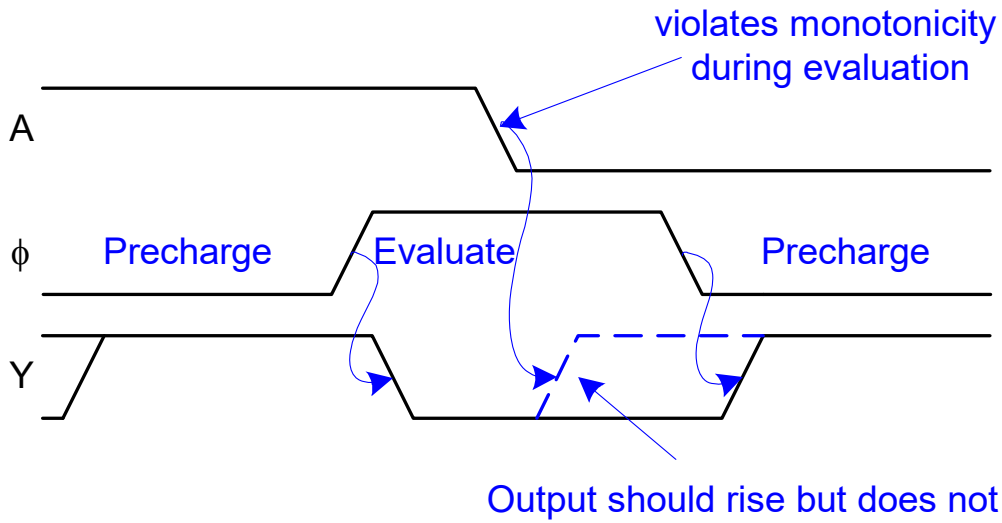
Logical Effort

	Inverter	NAND2	NOR2
unfooted	 g_d = p_d =	 g_d = p_d =	 g_d = p_d =
footed	 g_d = p_d =	 g_d = p_d =	 g_d = p_d =

Monotonicity

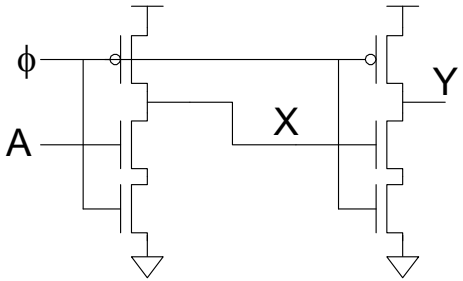
- Dynamic gates require *monotonically rising* inputs during evaluation
  - 0 -> 0
  - 0 -> 1
  - 1 -> 1
  - But not 1 -> 0

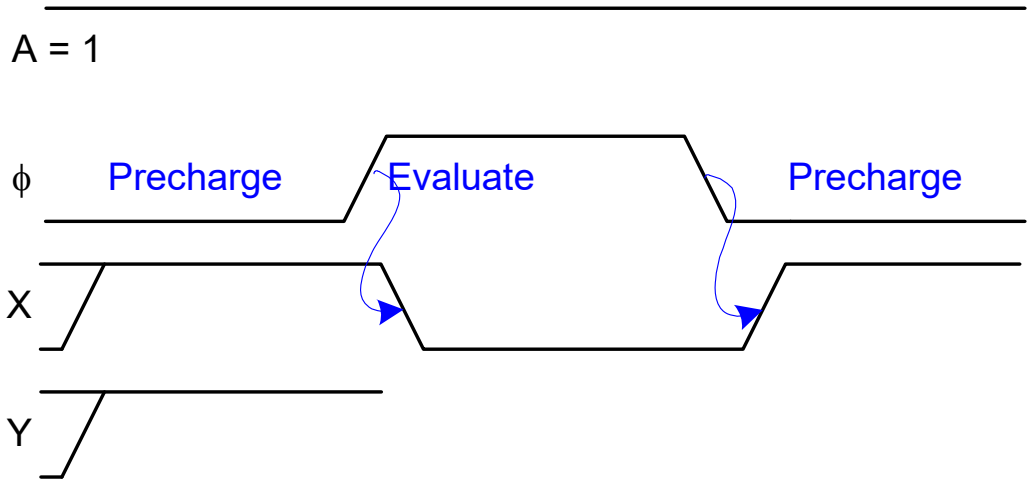




Monotonicity Woes

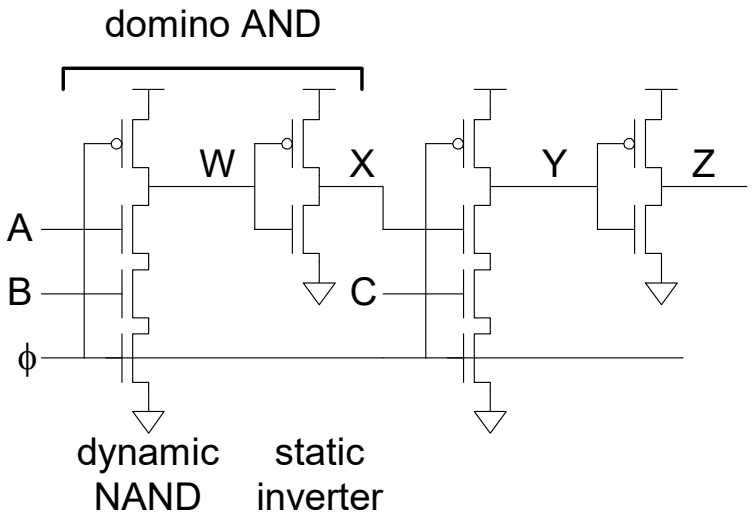
- But dynamic gates produce monotonically falling outputs during evaluation
- Illegal for one dynamic gate to drive another!

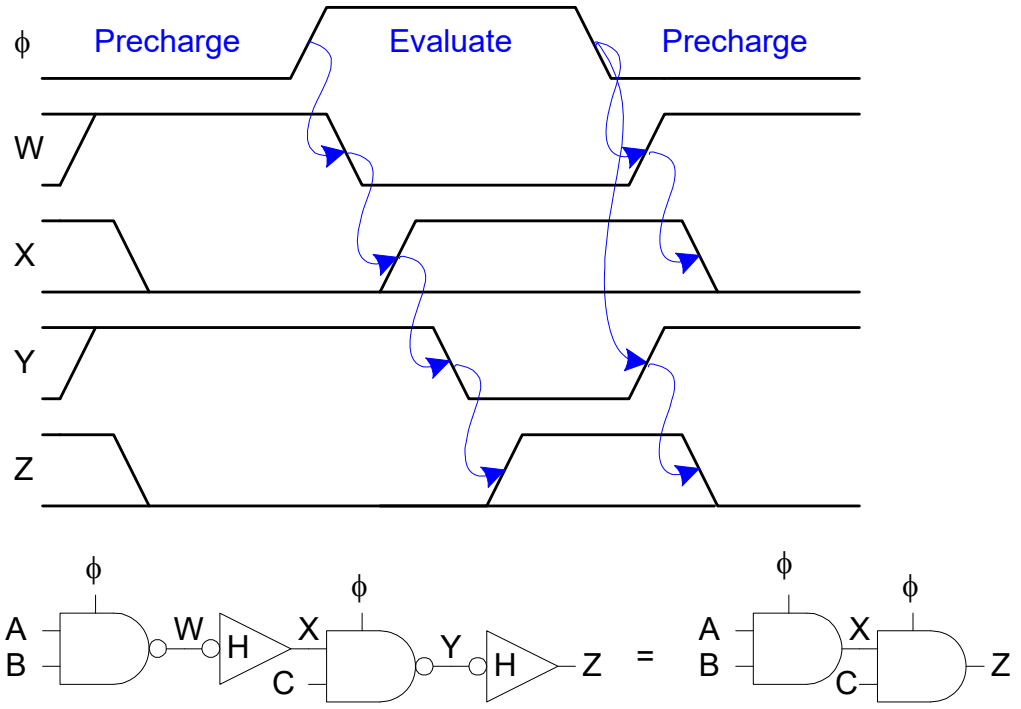




Domino Gates

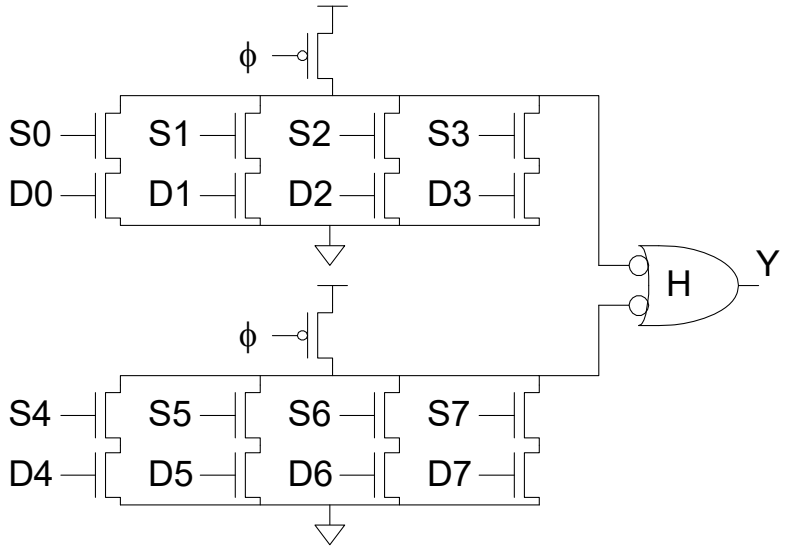
- Follow dynamic stage with inverting static gate
  - Dynamic / static pair is called domino gate
  - Produces monotonic outputs





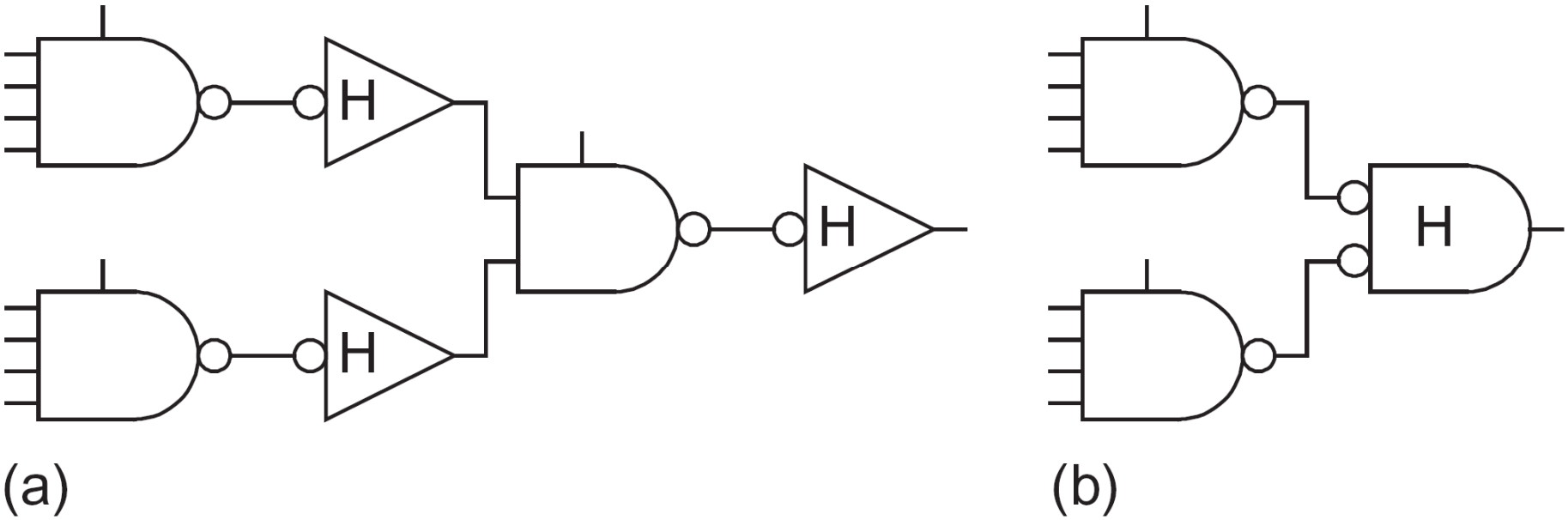
Domino Optimizations

- Each domino gate triggers next one, like a string of dominos toppling over
- Gates evaluate sequentially but precharge in parallel
- Thus evaluation is more critical than precharge
- HI-skewed static stages can perform logic



Domino and Compound Domino

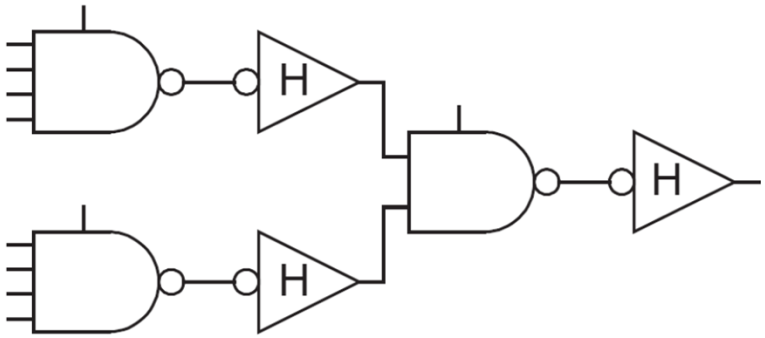
8-input NAND gate



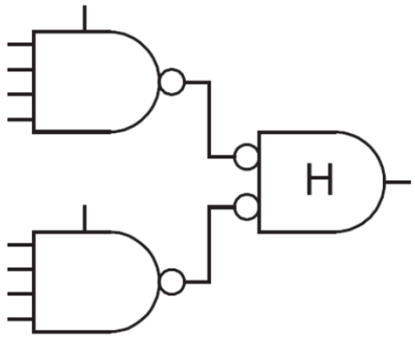
(a) (b)

Domino Compound Domino

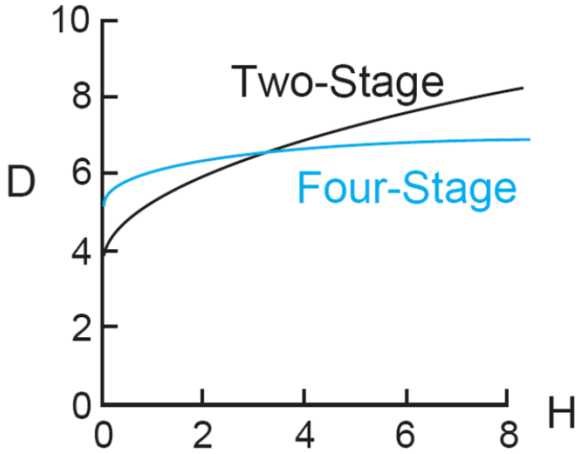
Domino and Compound Domino



(a)



(b)



(c)

g = 5/3 g = 5/6 g = 3/3 g = 5/6
p = 6/3 p = 5/6 p = 4/3 p = 5/6

G = (5/3)(5/6)(3/3)(5/6) = 125/108
P = 6/3 + 5/6 + 4/3 + 5/6 = 5

$$D = 4\left(\frac{125}{108}H\right)^{1/4} + 5$$

g = 5/3 g = 3/2
p = 6/3 p = 5/3

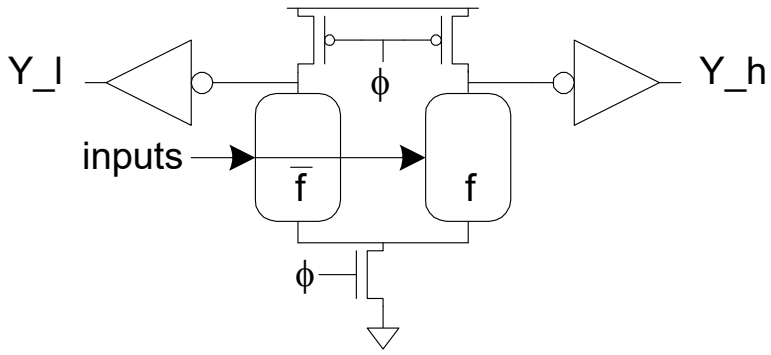
G = (5/3)(3/2) = 5/2
P = 6/3 + 5/3 = 11/3

$$D = 2\left(\frac{5}{2}H\right)^{1/2} + \frac{11}{3}$$

Dual-Rail Domino

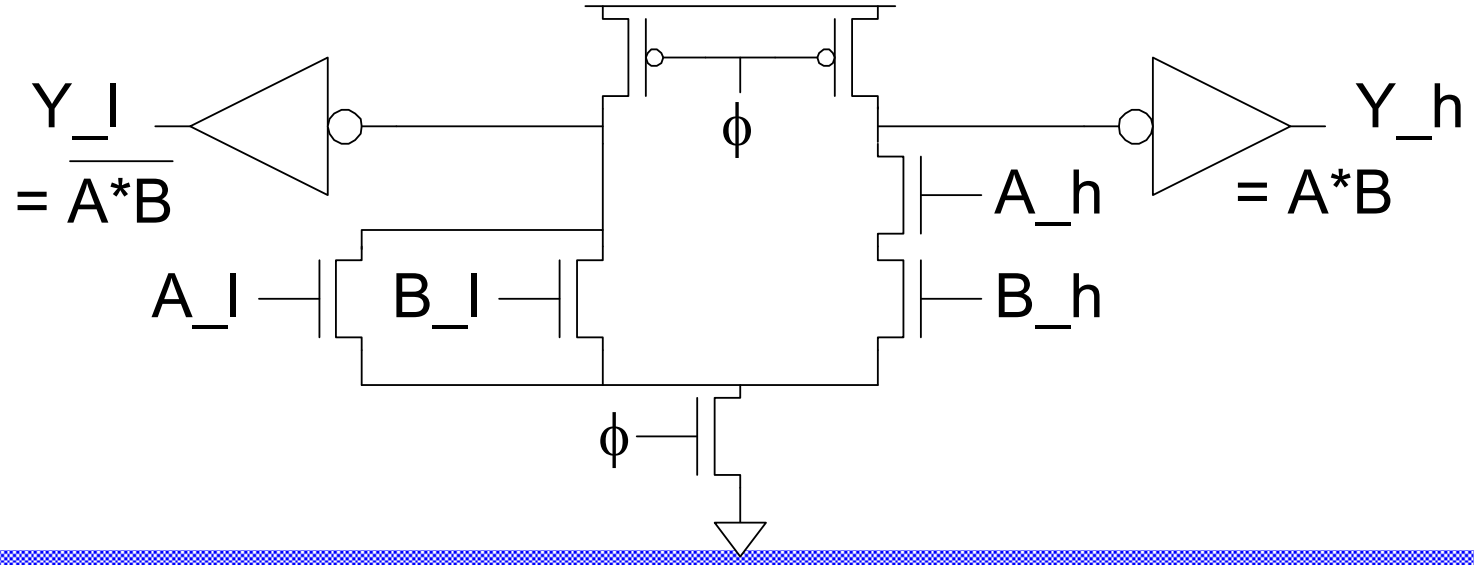
- Domino only performs noninverting functions:
  - AND, OR but not NAND, NOR, or XOR
- Dual-rail domino solves this problem
  - Takes true and complementary inputs
  - Produces true and complementary outputs

sig_h	sig_l	Meaning
0	0	Precharged
0	1	'0'
1	0	'1'
1	1	invalid

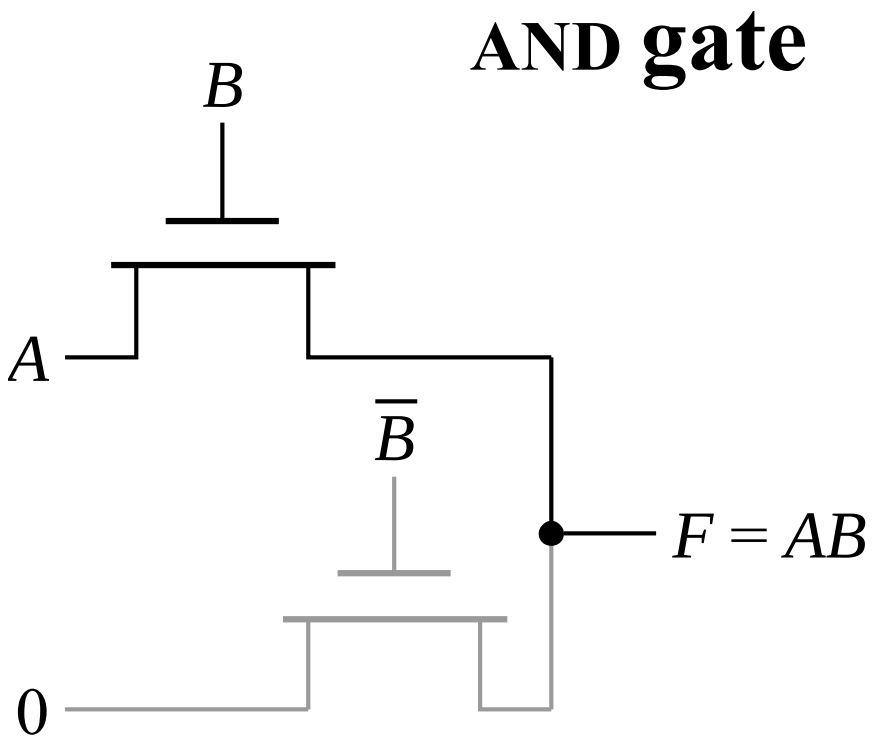


Example: AND/NAND

- Given A_h, A_l, B_h, B_l
- Compute Y_h = AB, Y_l = $\overline{AB}$
- Pulldown networks are conduction complements
- More area, wiring and power
- Perform inverting and noninverting logic



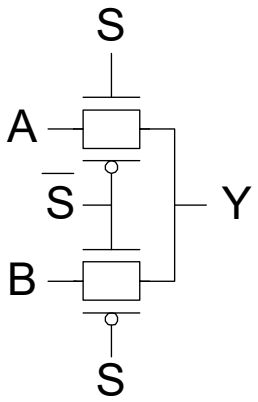
Pass Transistor Circuits

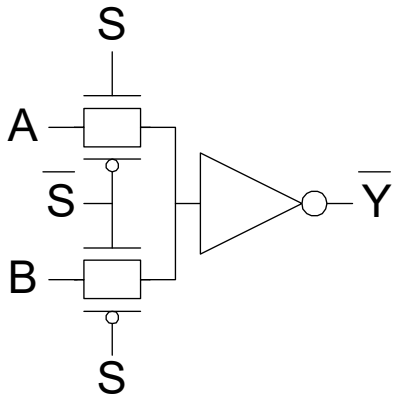


Pass Transistor Circuits

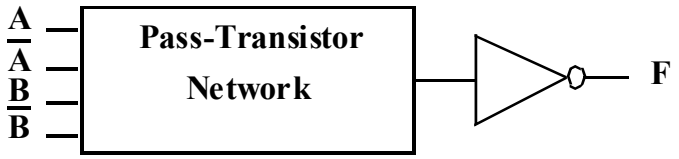
- Use pass transistors like switches to do logic
- Inputs drive diffusion terminals as well as gates

- CMOS + Transmission Gates:
  - 2-input multiplexer
  - Gates should be restoring

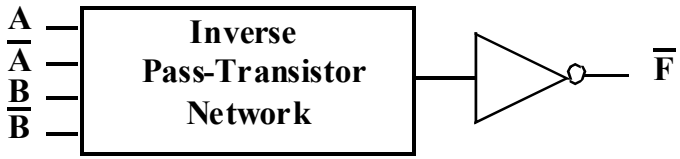


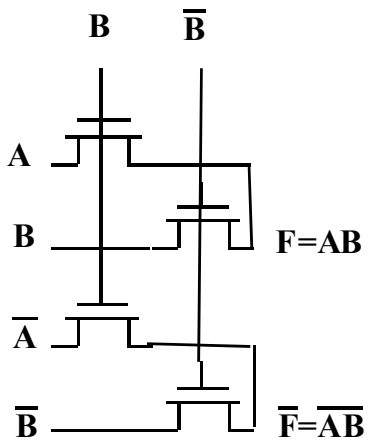


Complementary Pass-transistor Logic CPL

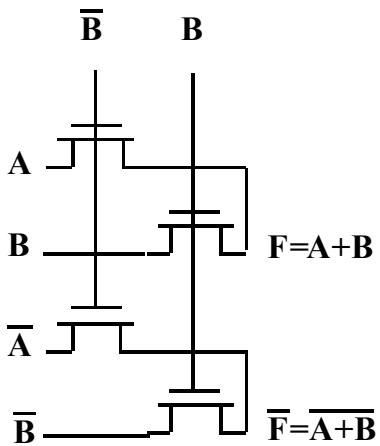


(a)

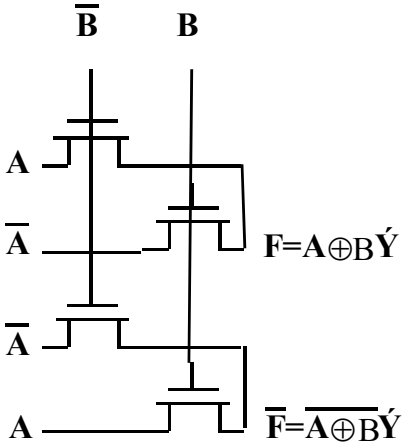




AND/NAND



OR/NOR

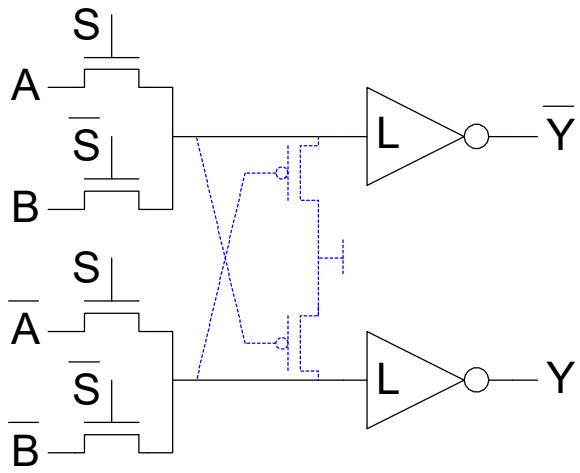


EXOR/NEXOR

(b)

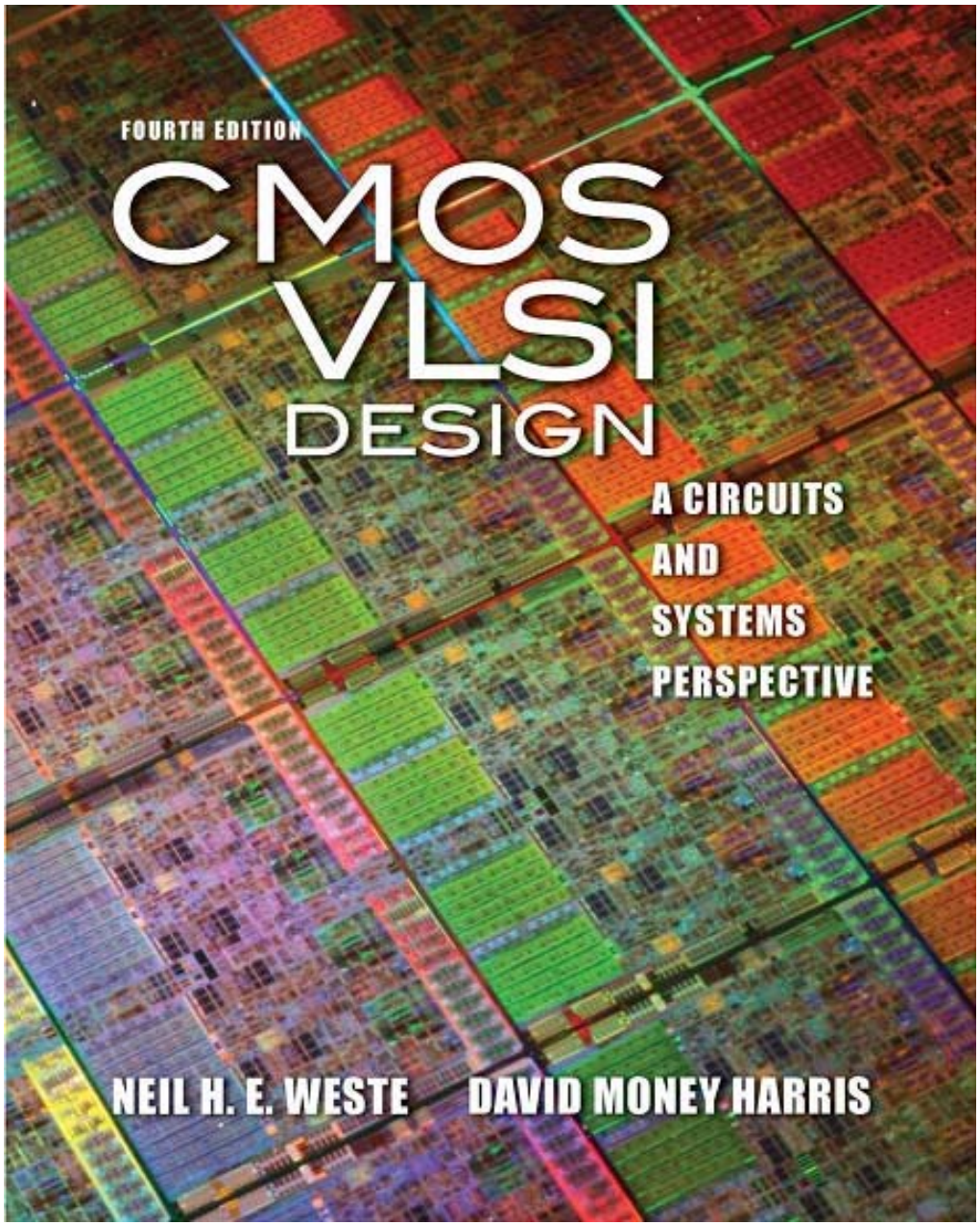
Complementary Pass-transistor Logic CPL

- Dual-rail form of pass transistor logic
- Avoids need for ratioed feedback
- Optional cross-coupling for rail-to-rail swing



Pass Transistor Summary

- Researchers investigated pass transistor logic for general purpose applications in the 1990's
  - Benefits over static CMOS were small or negative
  - No longer generally used
- However, pass transistors still have a niche in special circuits such as memories where they offer small size and the threshold drops can be managed



Lecture 12: Adders

Outline

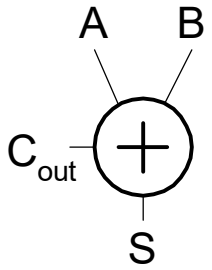
- Single-bit Addition
- Carry-Ripple Adder
- Carry-Skip Adder
- Carry-Select Adder
- Carry-Lookahead Adder
- Carry-Increment Adder
- Tree Adder

Single-Bit Addition

Half Adder

$$S = A \oplus B$$

$$C_{out} = A \cdot B$$

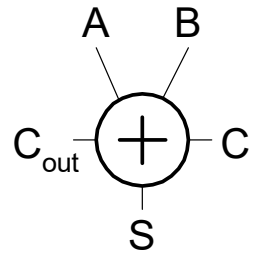


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

Full Adder

$$S = A \oplus B \oplus C$$

$$C_{out} = MAJ(A, B, C)$$



A	B	C	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

PGK

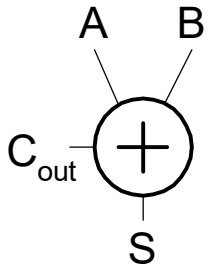
- For a full adder, define what happens to carries (in terms of A and B)
  - Generate: C_{out} = 1 independent of C
    - G =
  - Propagate: C_{out} = C
    - P =
  - Kill: C_{out} = 0 independent of C
    - K =

Single-Bit Addition

Half Adder

$$S = A \oplus B$$

$$C_{out} = A \cdot B$$

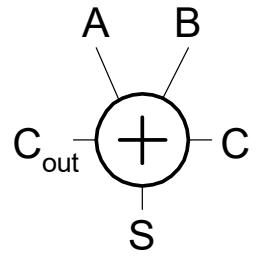


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

Full Adder

$$S = A \oplus B \oplus C$$

$$C_{out} = MAJ(A, B, C)$$



A	B	C	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

For the Full Adder
Note the symmetry of S and C_{out}
Inverting the inputs inverts the outputs

Full Adder Design I

- Brute force implementation from eqns

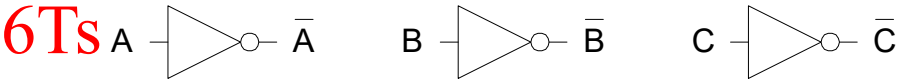
$$S = A \oplus B \oplus C$$

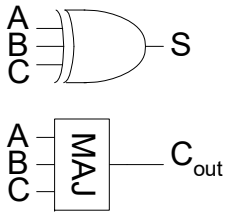
$$C_{out} = MAJ(A, B, C)$$

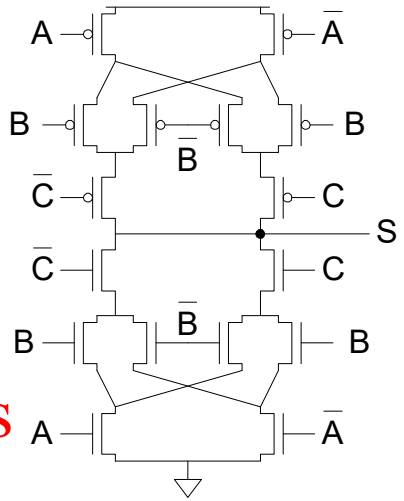
$$C_{out} = \overline{\bar{A}\bar{B} + \bar{C}(\bar{A} + \bar{B})}$$

32Ts

6Ts

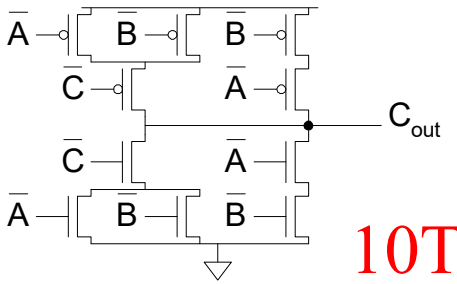






16Ts

Self-Duality



10Ts

Full Adder Design II

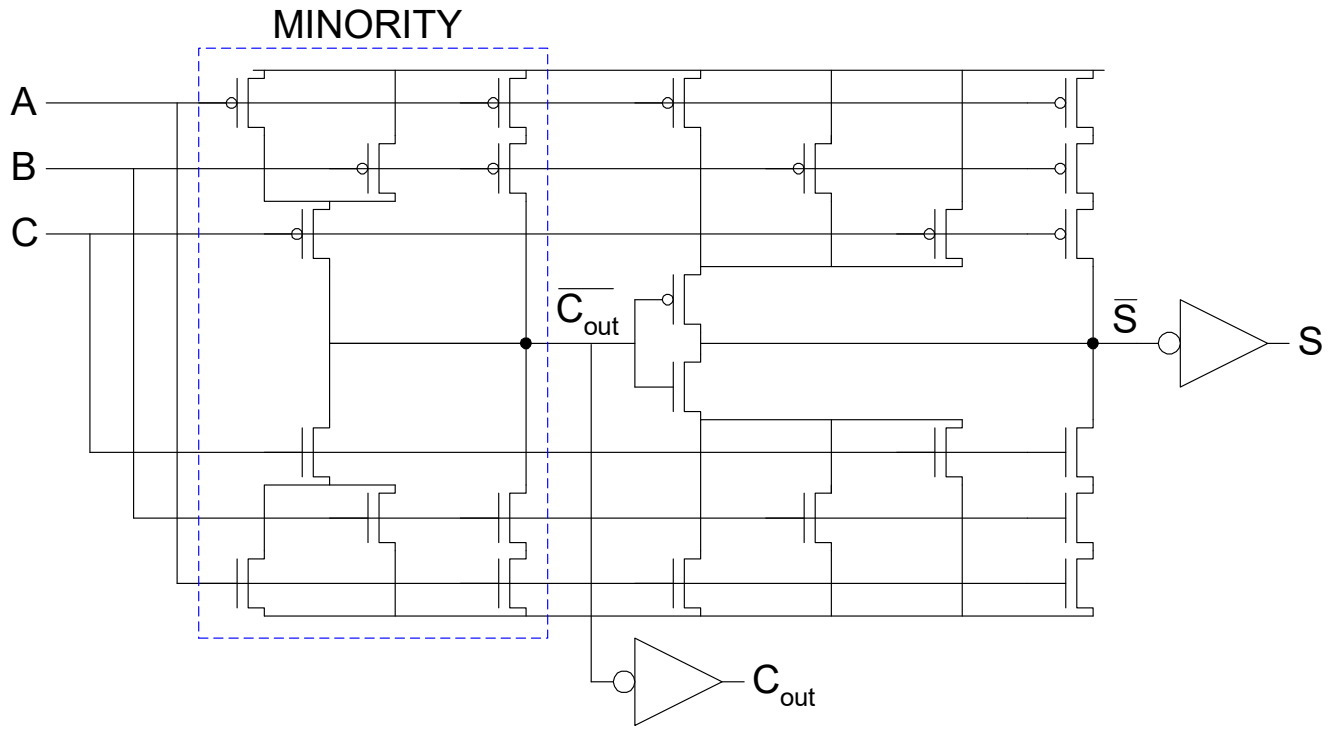
Mirror Adder

- Factor S in terms of C_{out}

  $S = ABC + (A + B + C)(\sim C_{out})$

  Saves 4Ts

- Critical path is usually C to C_{out} in ripple adder

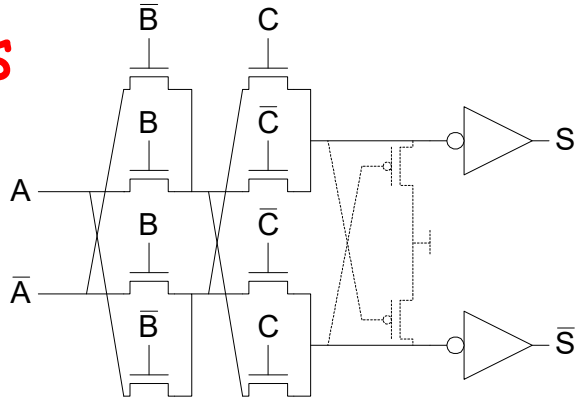


Full Adder Design III

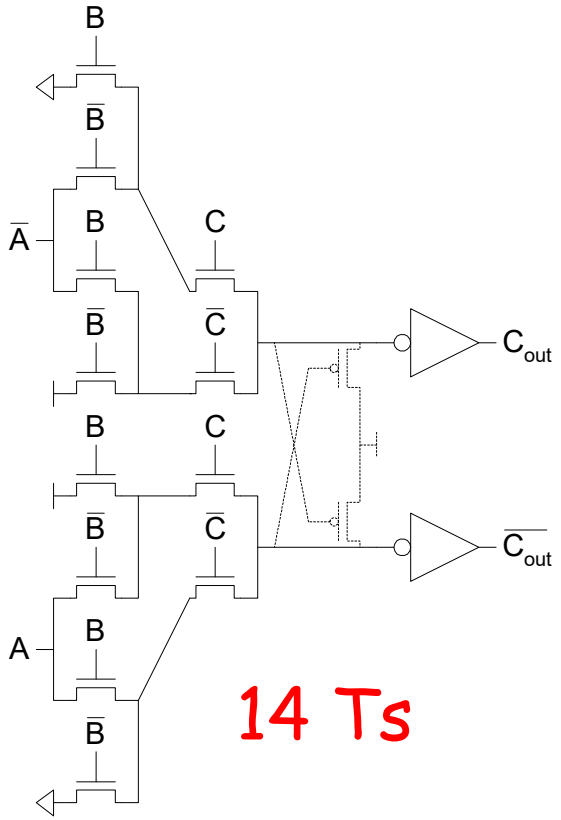
- Complementary Pass Transistor Logic (CPL)
  - Slightly faster, but more area

38 Ts

10 Ts



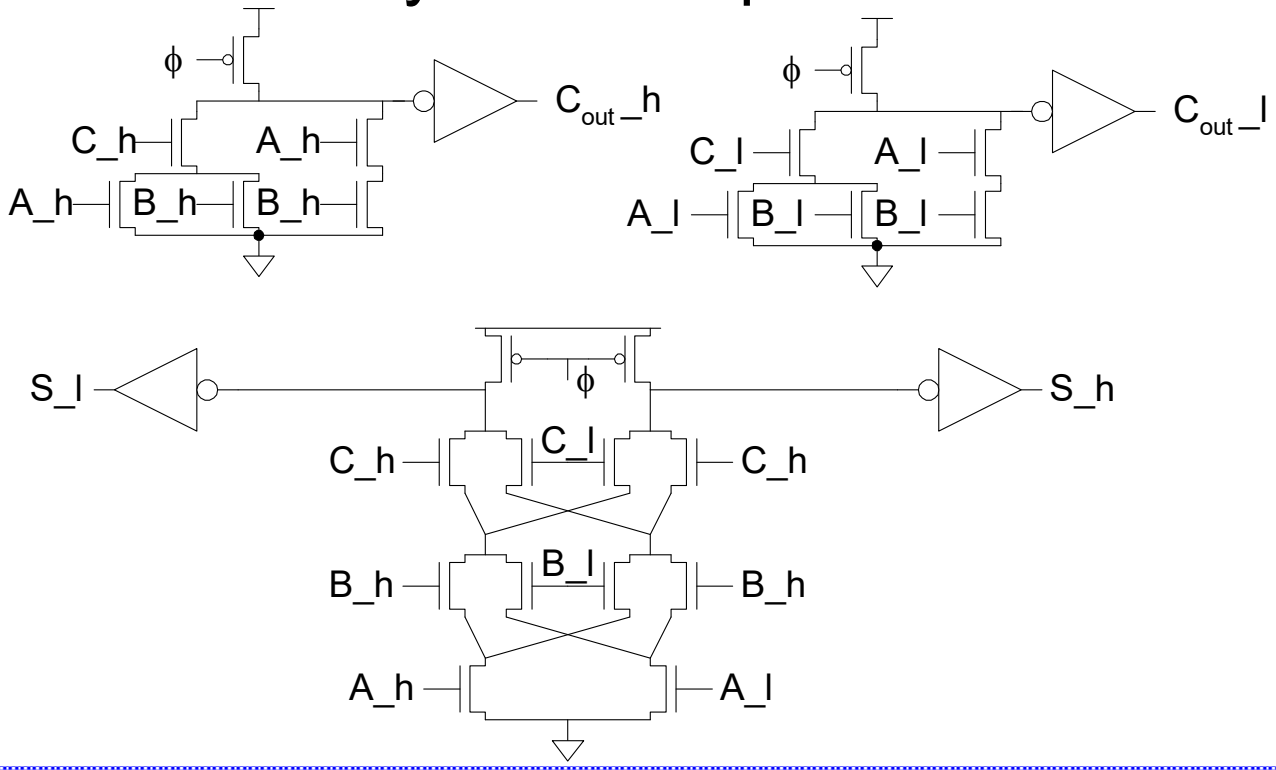
14 Ts for invertrs



14 Ts

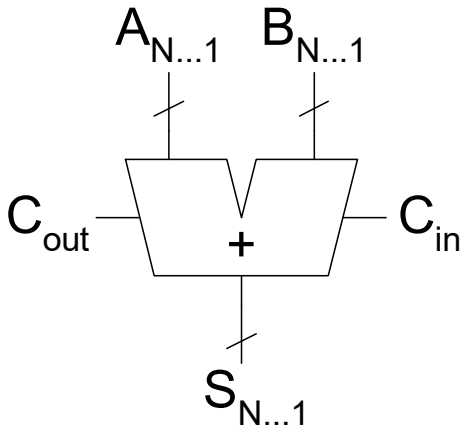
Full Adder Design IV :Dynamic Logic

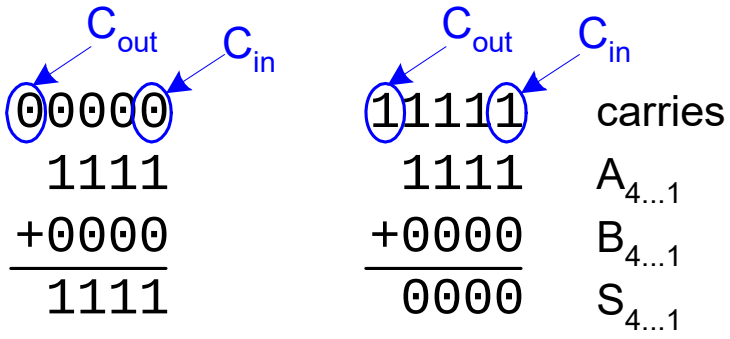
- Dual-rail domino
  - Very fast, but large and power hungry
  - Used in very fast multipliers



Carry Propagate Adders

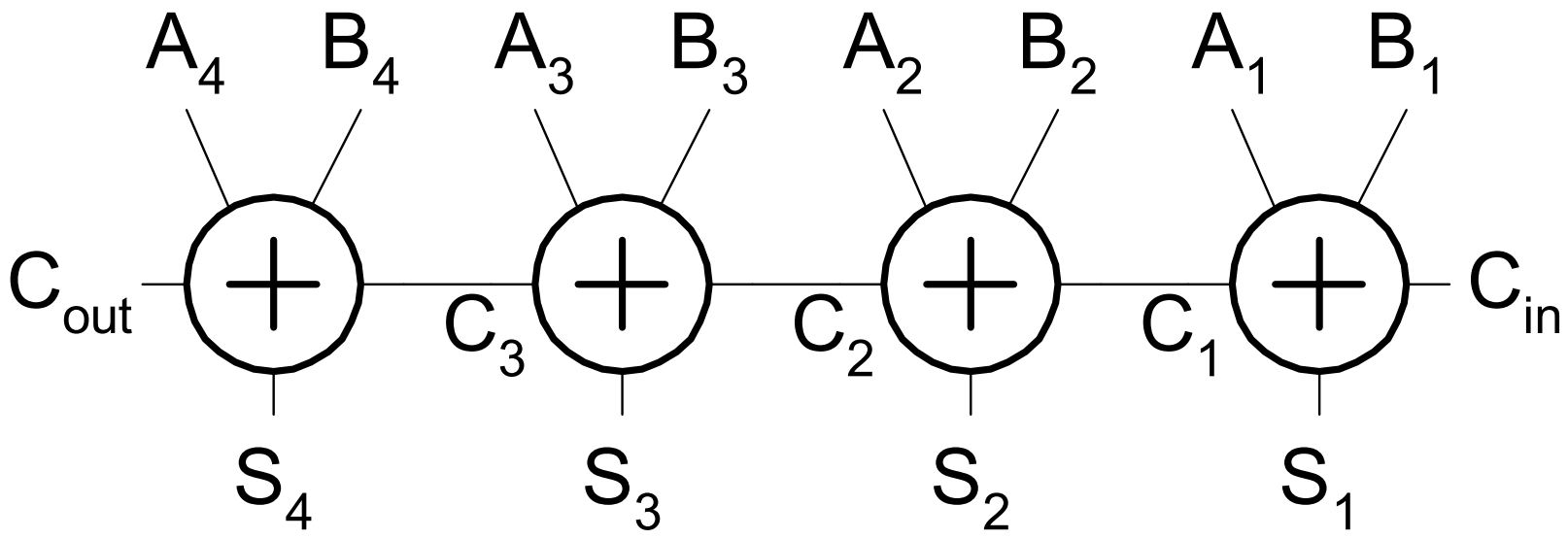
- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?





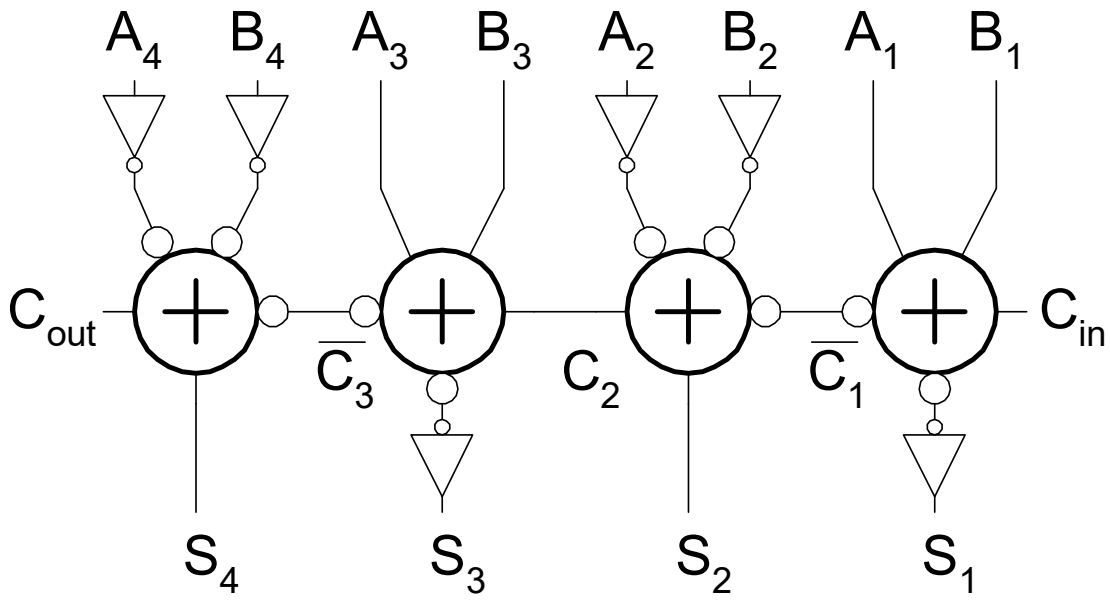
Carry-Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from C_{in} to C_{out}
  - Design full adder to have fast carry delay



Inversions

- Critical path passes through majority gate
  - Built from minority + inverter
  - Eliminate inverter and use inverting full adder



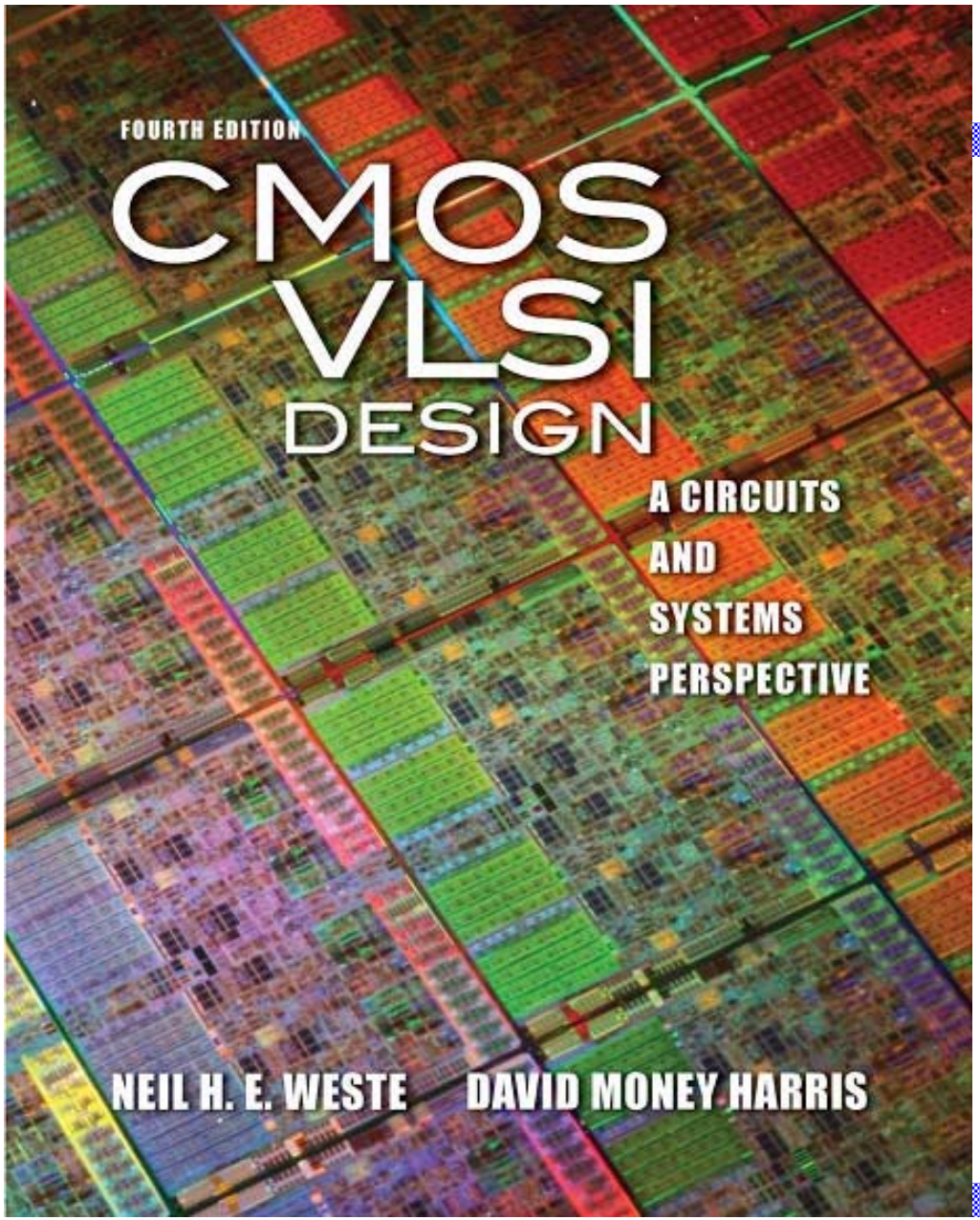
Generate, Propagate , and Kill

Truth table for full adder

A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

$$G = A.B, C_{out} = 1$$
$$P = A \oplus B, C_{out} = C_{in}$$
$$K = \bar{A}.\bar{B}, C_{out}=0$$

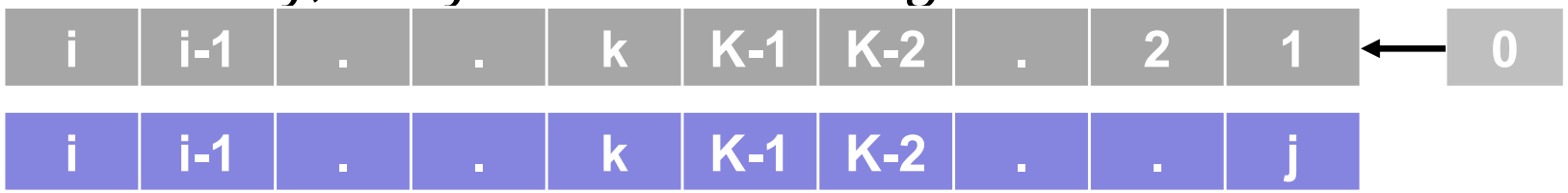
$$S = A \oplus B \oplus C$$
$$= P \oplus C$$
$$C_{out} = G + P.C$$
$$= G + \bar{K}.C$$



Lecture 13: Adders

Group PG Signals

- Generalize the G and P Signals to describe a group of bits from $i:j, i > j$ *with a running index* k



  - A group of bits generates a carry if its carry-out is true independent of the carry-in.
  - A group of bits propagates a carry if its carry-out is true and there exists a carry-in.
- The carry-in is defined as coming from bit 0.

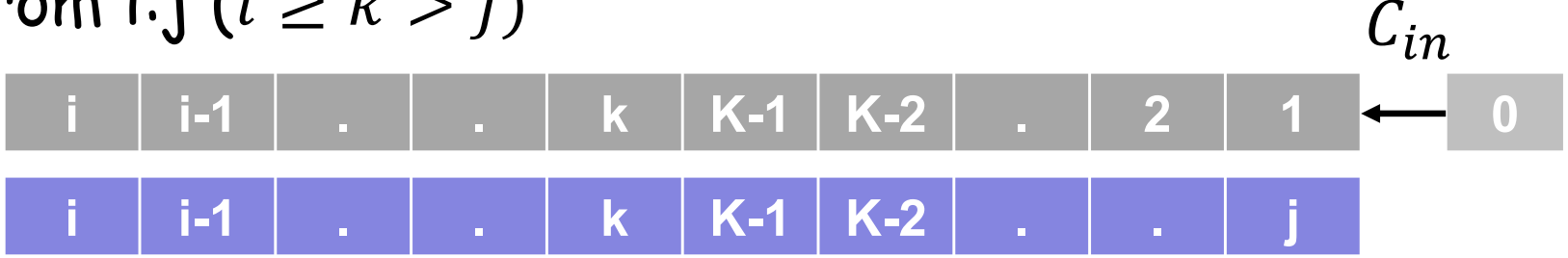
$$G_0 = C_{in} \qquad and \qquad P_0 = 0$$

Group PG Signals

- Bitwise generate and propagate signals:

$$G_{i:i} = G_i = A_i.B_i \qquad and \qquad P_{i:i} = P_i = A_i \oplus B_i$$

- Generate and propagate for groups of bits spanning from i:j ($i \geq k > j$)



$$G_{i:j} = G_{i:k} + P_{i:k}.G_{k-1:j}$$
$$P_{i:j} = P_{i:k}.P_{k-1:j}$$

$$C_i = G_i + P_i C_{i-1}$$
$$C_i = G_i + P_i G_{i-1:0}$$
$$C_i = G_{i:0}$$

Generate / Propagate

- Equations often factored into G and P
- Generate and propagate for groups spanning i:j

$G_{i:j} =$

$P_{i:j} =$

- Base case

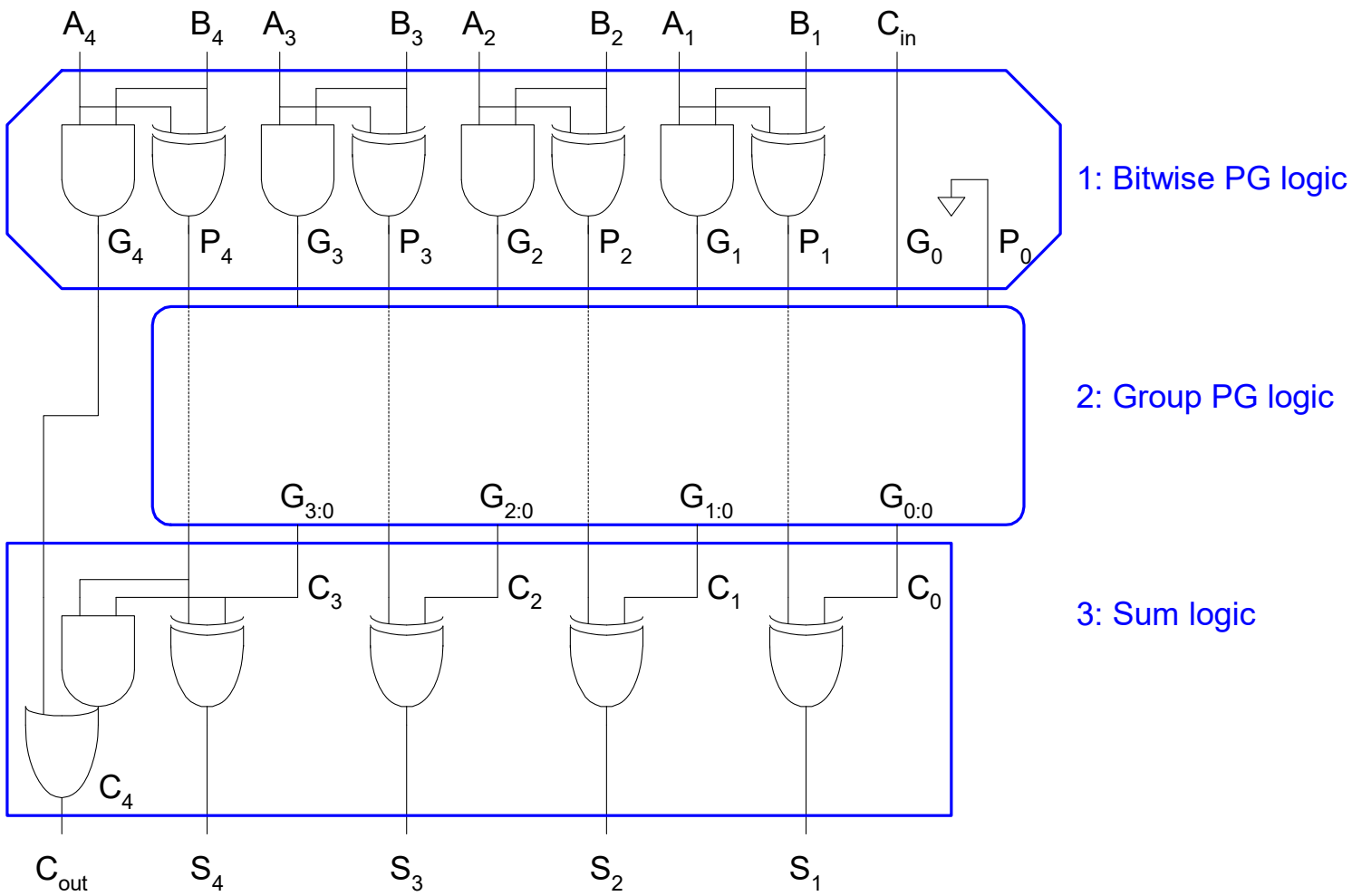
$G_{i:i} \equiv$ $\qquad$ $G_{0:0} \equiv$

$P_{i:i} \equiv$ $\qquad$ $P_{0:0} \equiv$

- Sum: $S_i = P_i \oplus C_i$

$S_i =$

PG Logic


A4
B4
A3
B3
A2
B2
A1
B1
Cin
G4
P4
G3
P3
G2
P2
G1
P1
G0
P0
G3:0
G2:0
G1:0
G0:0
C3
C2
C1
C0
C4
Cout
S4
S3
S2
S1
1: Bitwise PG logic
2: Group PG logic
3: Sum logic

Carry-Ripple Revisited

- P and G signals simplify the majority function into AND-OR

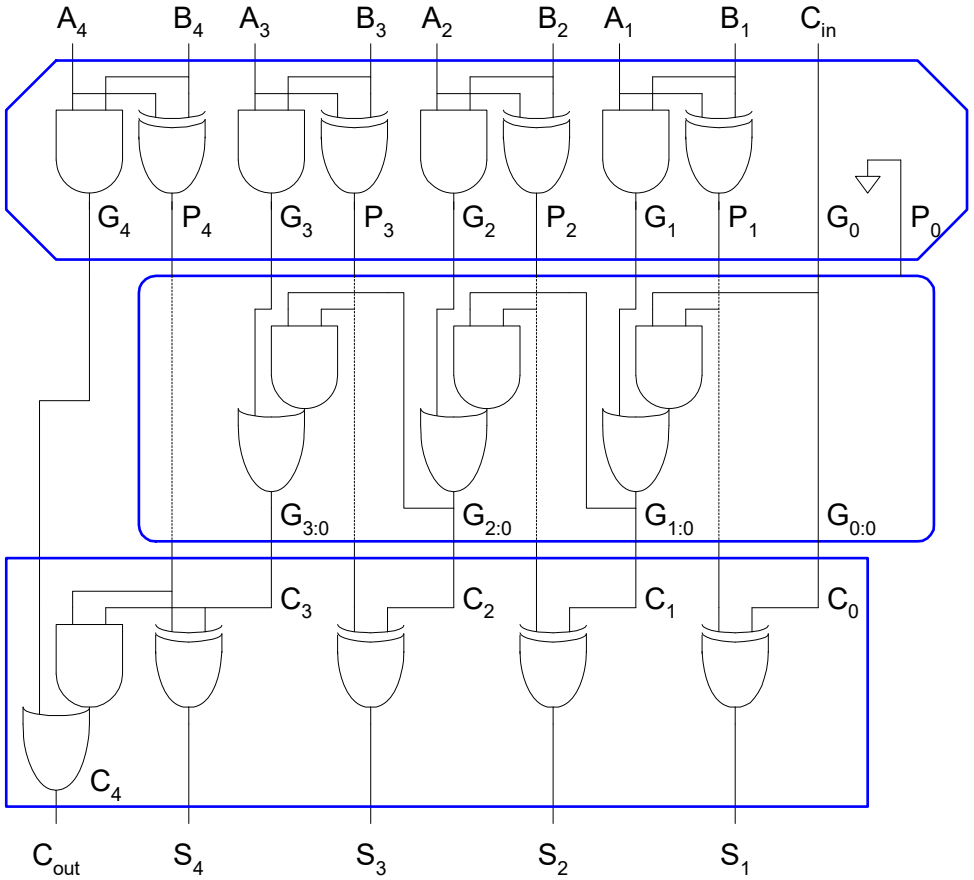
$$S_i = P_i \oplus C_{i-1}$$

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

$$C_i = G_i + P_i . G_{i-1:0}$$

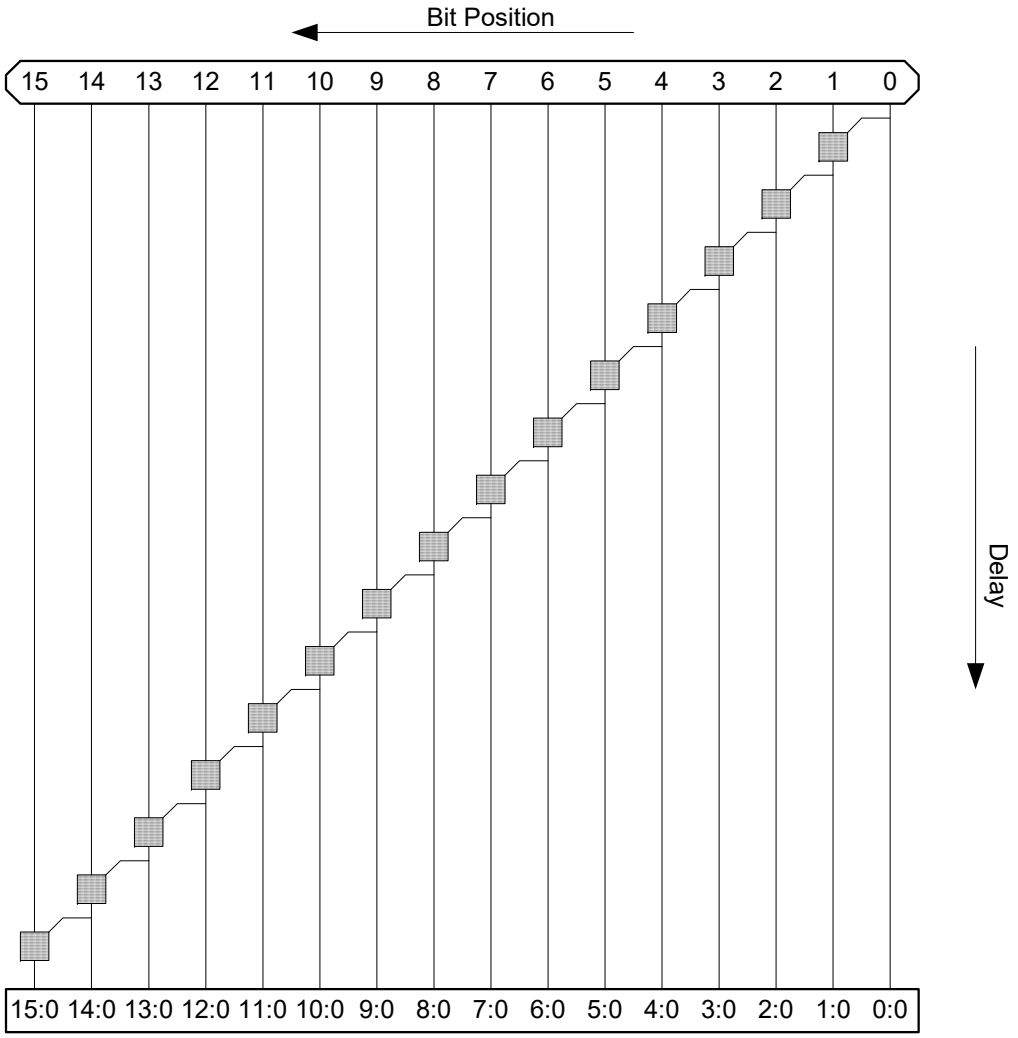
$$C_i = G_{i:0}$$

- AND-OR current bit G with previous group G (carry-in)
- Group P not required



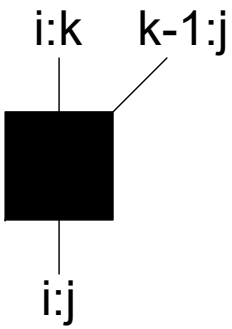
Carry-Ripple PG Diagram

$t_{\text{ripple}} =$

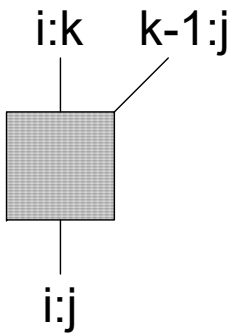


PG Diagram Notation

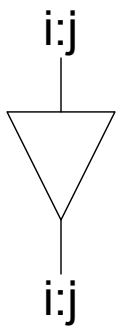
Black cell

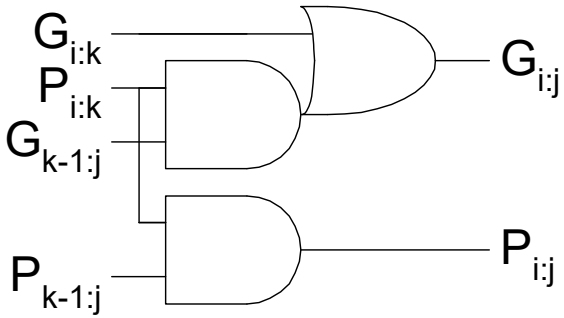


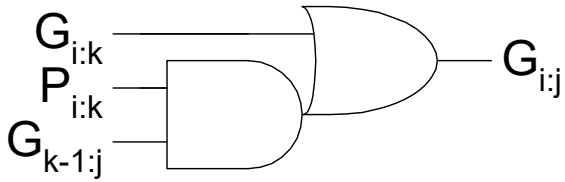
Gray cell

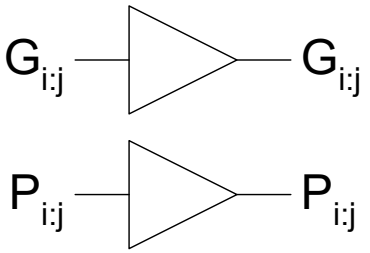


Buffer



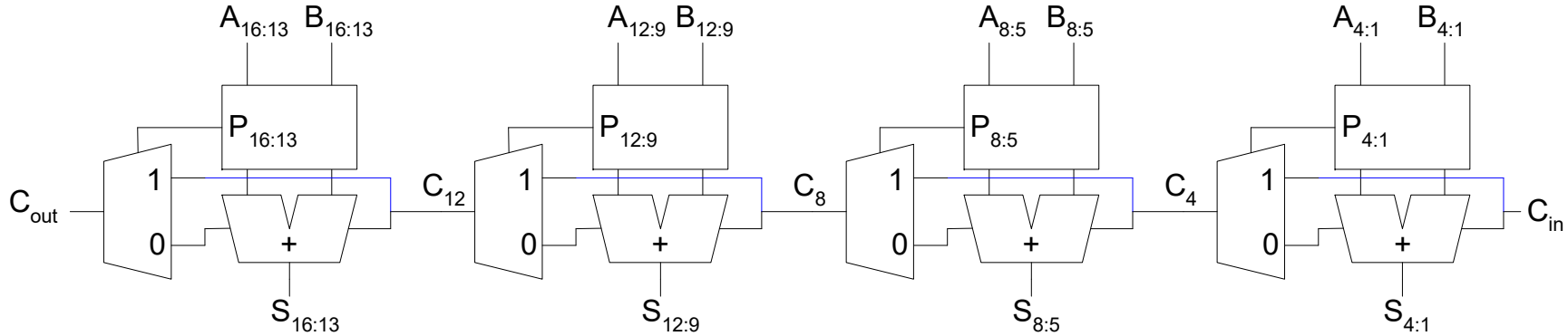






Carry-Skip Adder

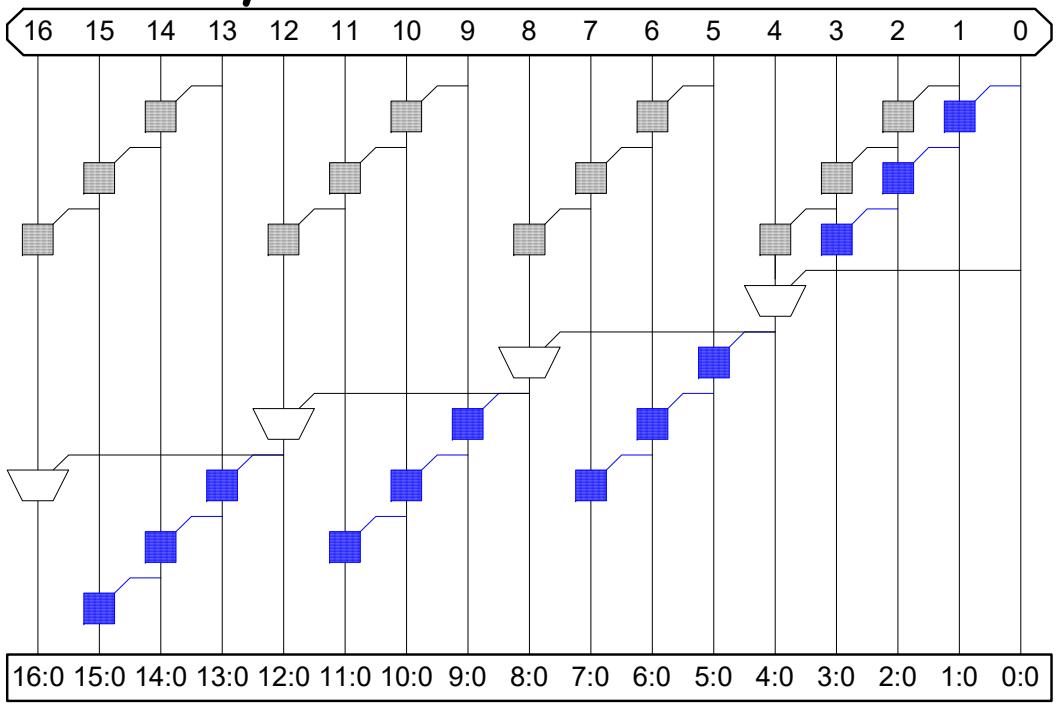
- Carry-ripple is slow through all N stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal



- Critical Path: 1. Bit 1 generates a carry.
  2. Carry ripples through next 3 bits
  3. Skip through the next two 4-bit blocks
  4. Ripple through the last 4-bit block to produce its sums.

Carry-Skip PG Diagram

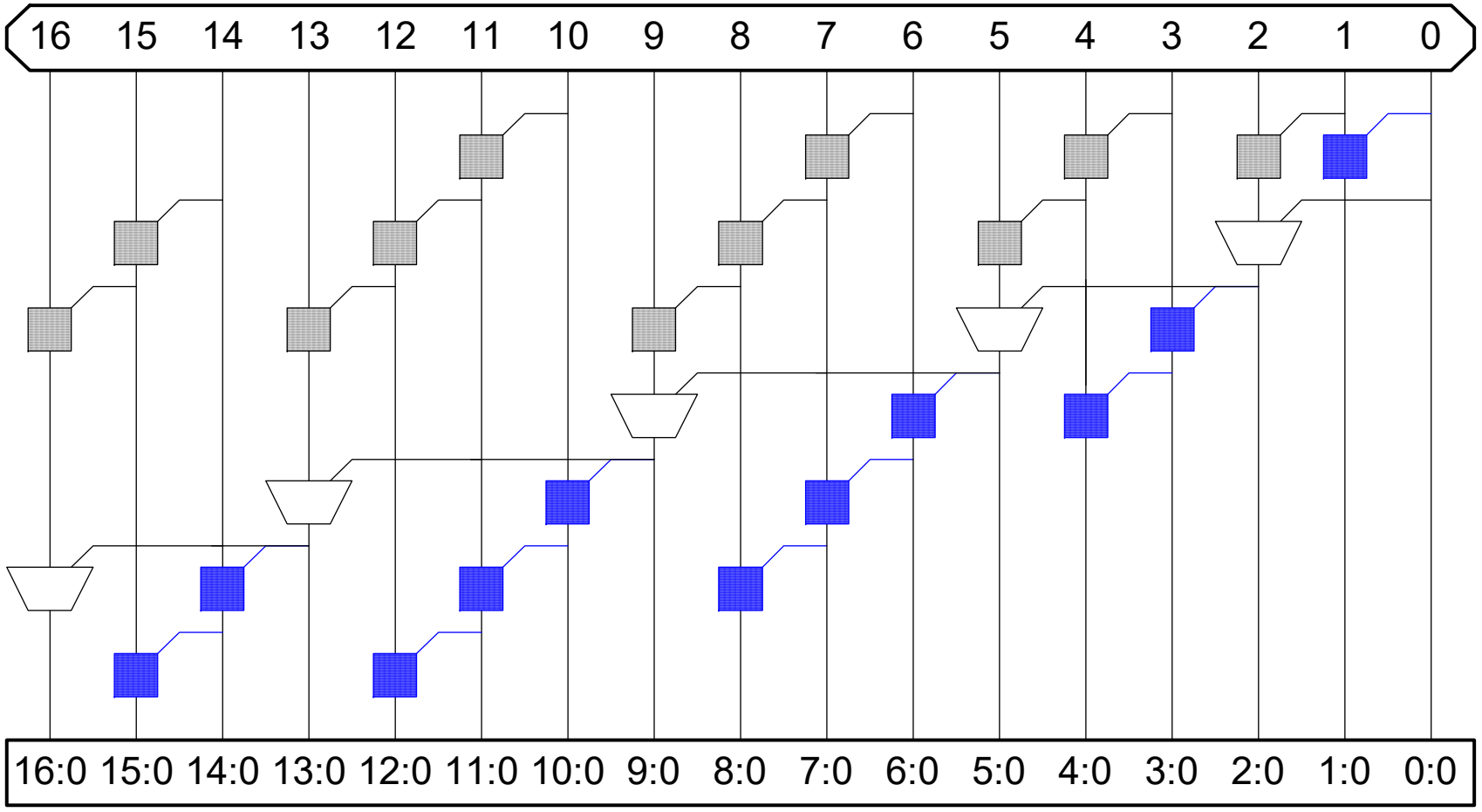
- Gray: when a group generates a carry out.
- Blue: $G_{i:0}$ updated when a carry-in arrives.



- For k groups of and n bits per group (N=nk)

$$t_{skip} = t_{pg} + 2(n-1)t_{AO} + (k-1)t_{mux} + t_{xor}$$

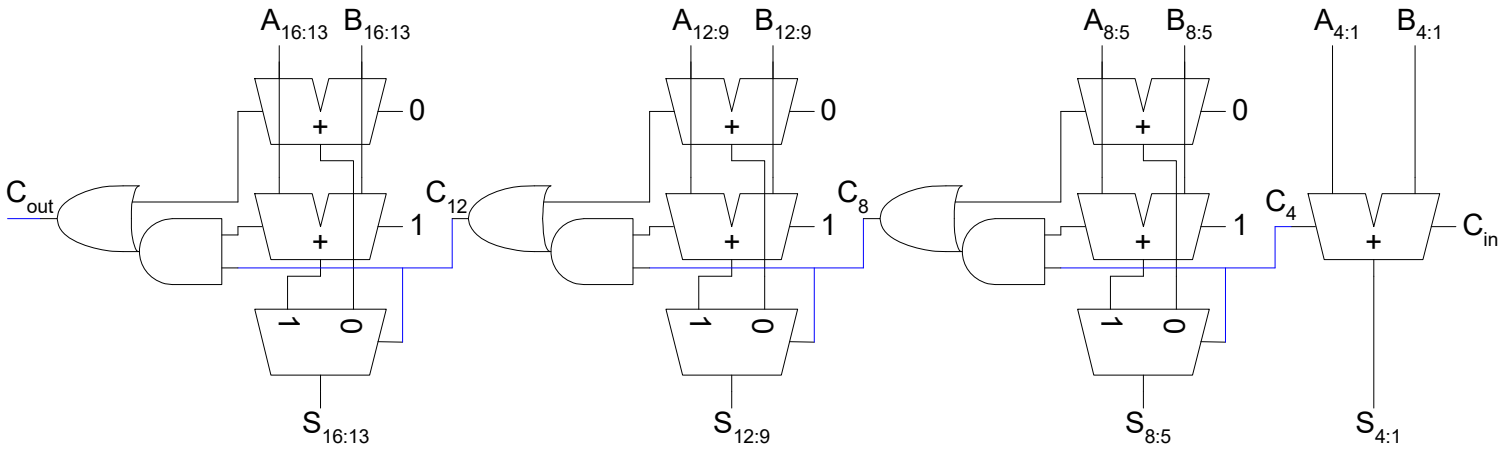
Variable Group Size



Delay grows as O(sqrt(N))

Carry-Select Adder

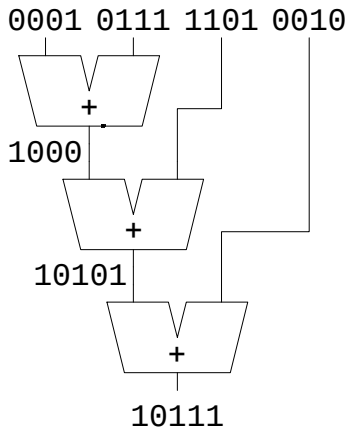
- Trick for critical paths dependent on late input X
  - Precompute two possible outputs for X = 0, 1
  - Select proper output when X arrives
- Carry-select adder precomputes n-bit sums
  - For both possible carries into n-bit group



$$t_{select} = t_{pg} + [n + (k-2)]t_{AO} + t_{mux}$$

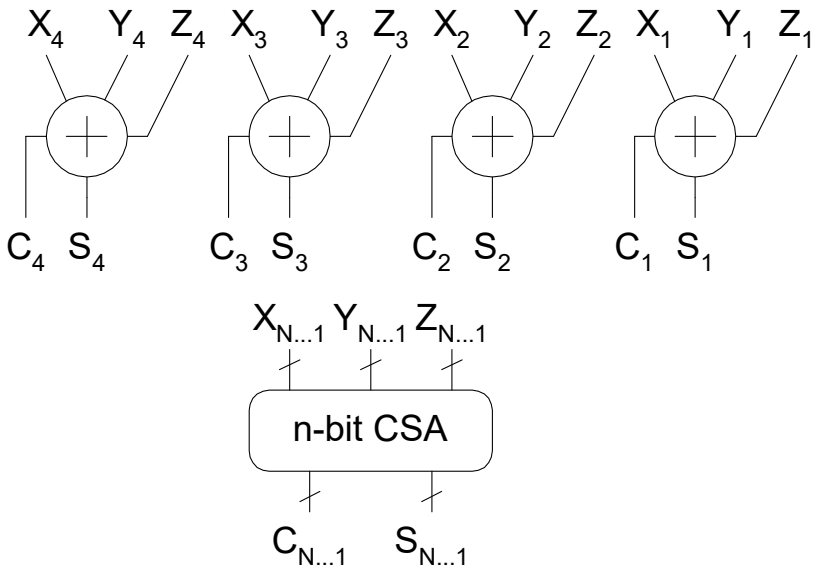
Multi-input Adders

- Suppose we want to add k N-bit words
  - Ex: 0001 + 0111 + 1101 + 0010 = 10111
- Straightforward solution: k-1 N-input CPAs
  - Large and slow



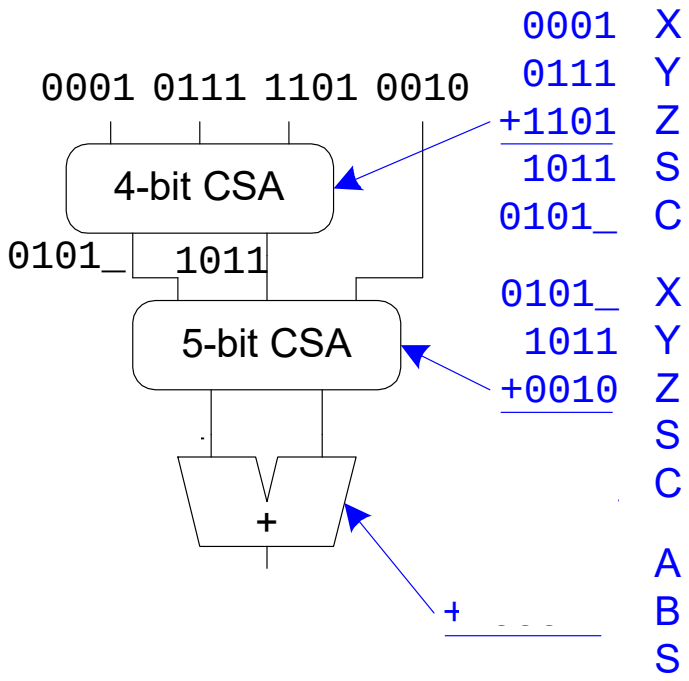
Carry Save Addition

- A full adder sums 3 inputs and produces 2 outputs
  - Carry output has twice *weight* of sum output
- N full adders in parallel are called *carry save adder*
  - Produce N sums and N carry outs



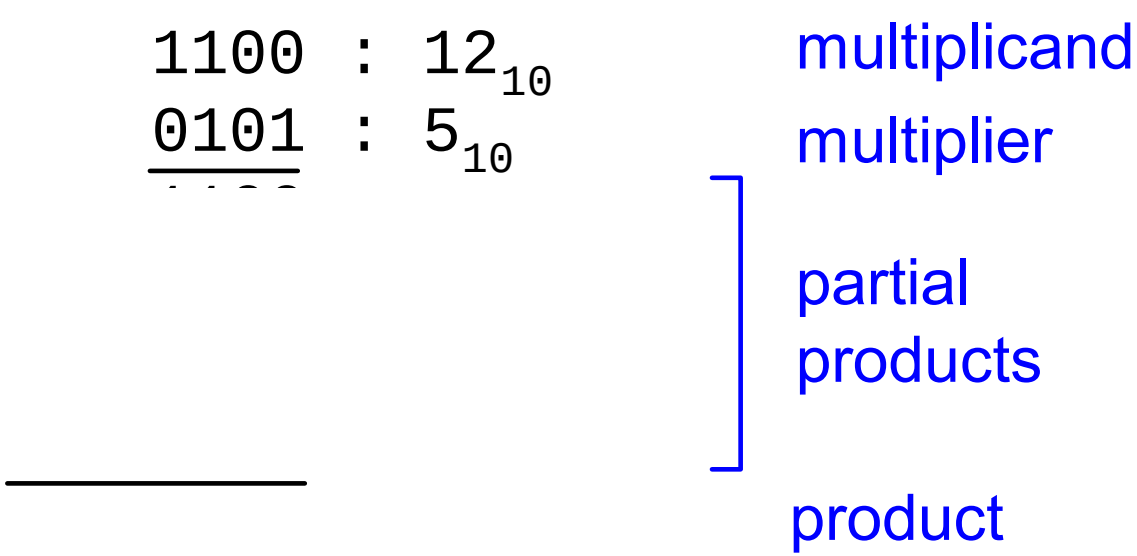
CSA Application

- Use k-2 stages of CSAs
  - Keep result in carry-save redundant form
- Final CPA computes actual result



Multiplication

- Example:

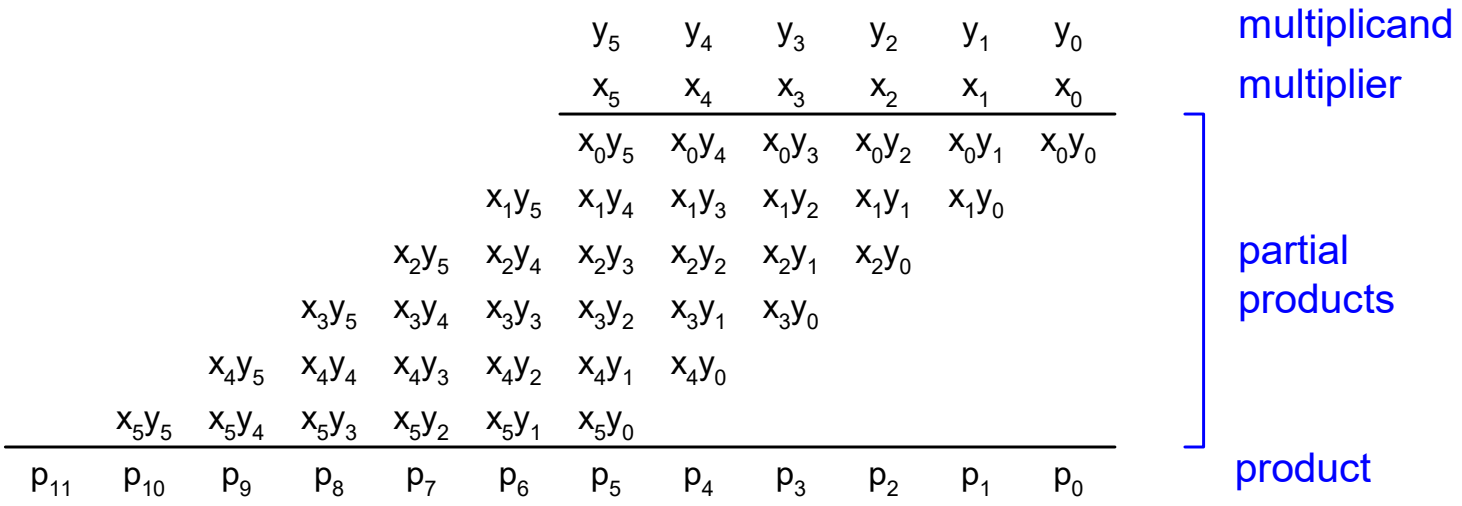


- M x N-bit multiplication
  - Produce N M-bit partial products
  - Sum these to produce M+N-bit product

General Form

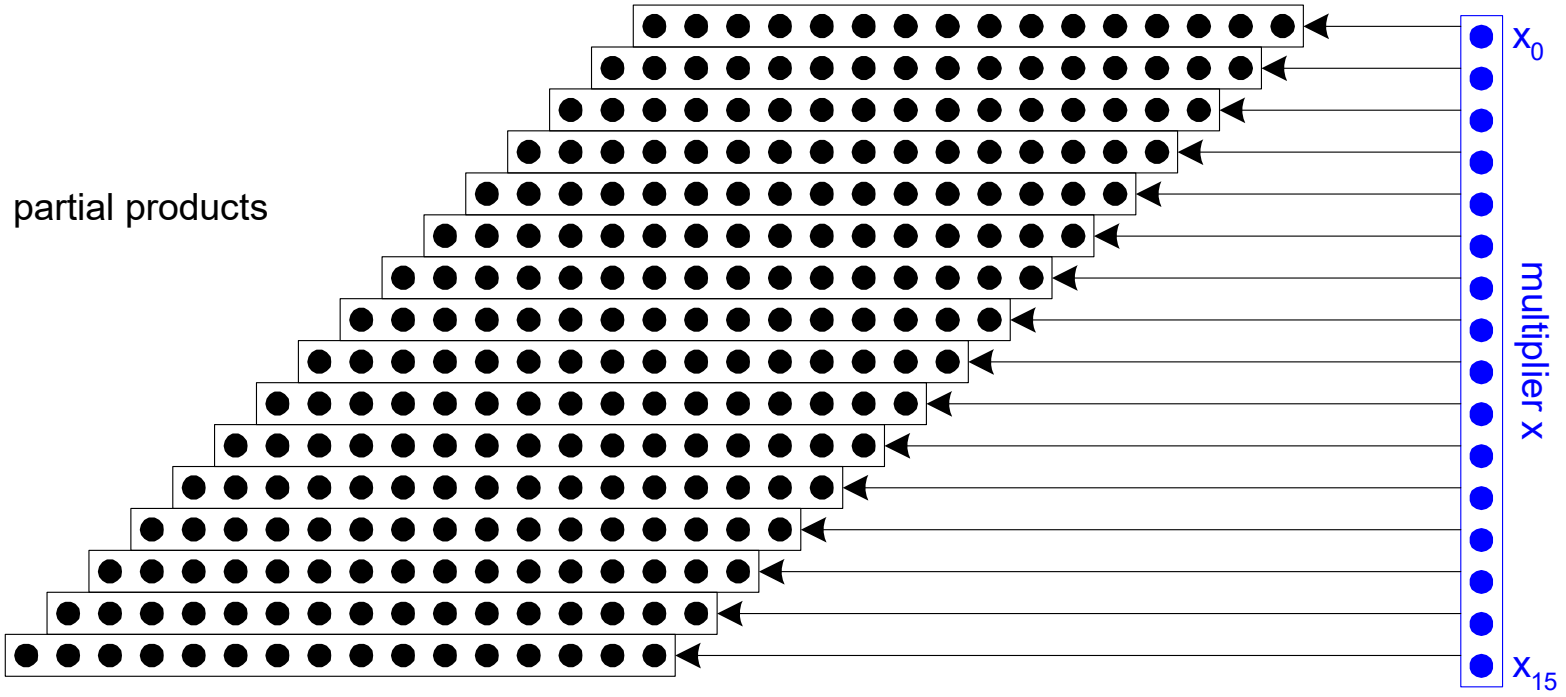
- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product:

$$P = \left(\sum_{j=0}^{M-1} y_j 2^j \right) \left(\sum_{i=0}^{N-1} x_i 2^i \right) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$

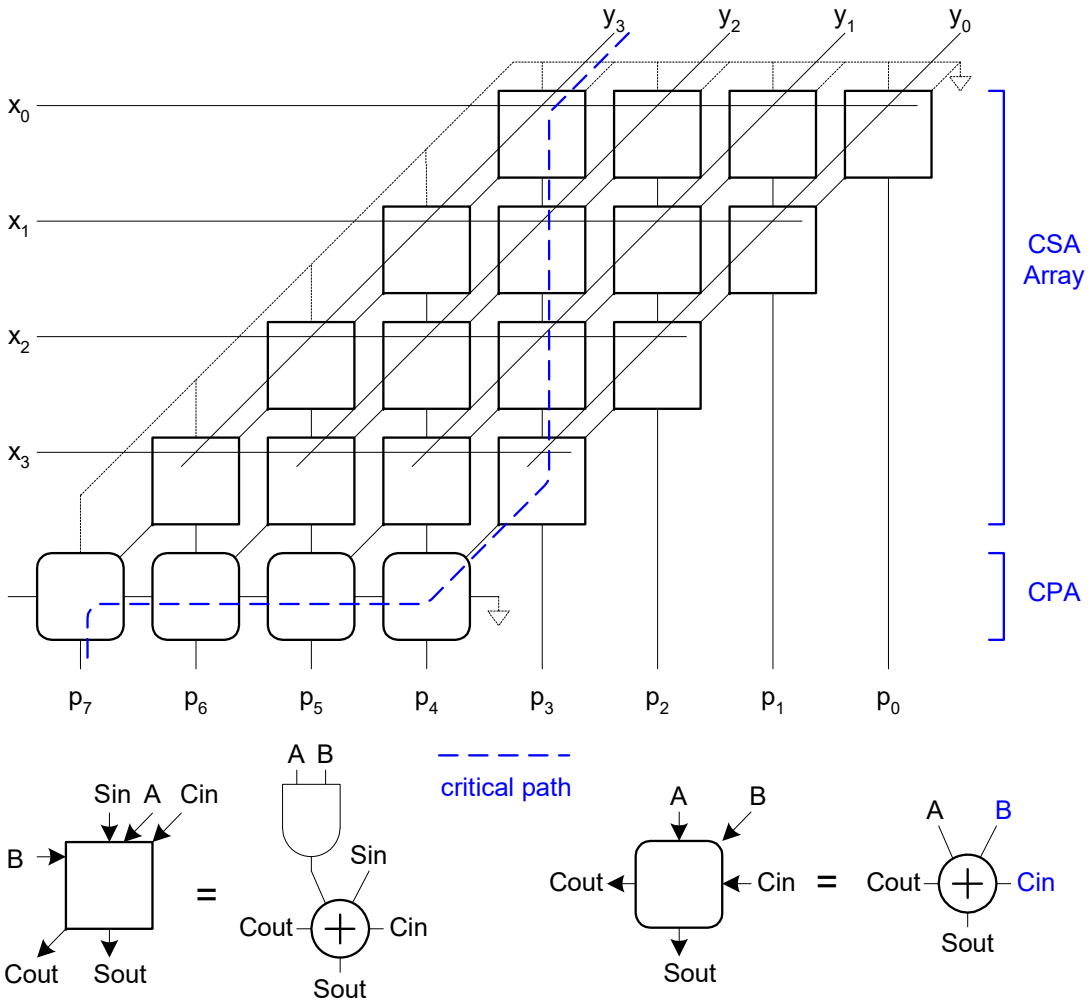


Dot Diagram

- Each dot represents a bit

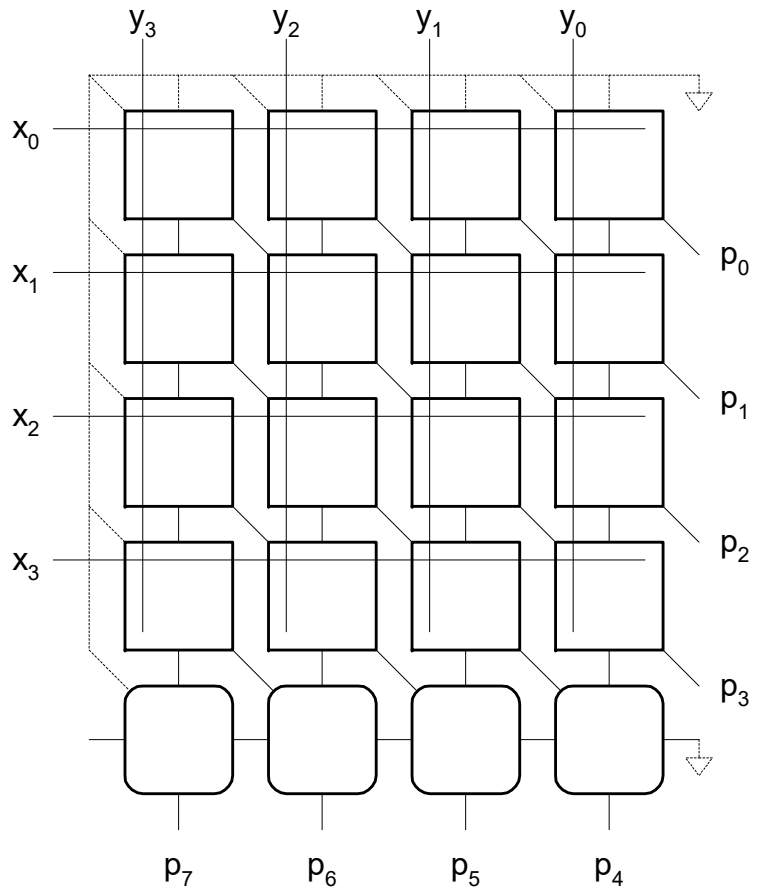


Array Multiplier



Rectangular Array

- Squash array to fit rectangular floorplan



Fewer Partial Products

- Array multiplier requires N partial products
- If we looked at groups of r bits, we could form N/r partial products.
  - Faster and smaller?
  - Called radix-2^r encoding
- Ex: r = 2: look at pairs of bits
  - Form partial products of 0, Y, 2Y, 3Y
  - First three are easy, but 3Y requires adder ☹

Booth Encoding

- Replace 3y with –y ⟹ add y to the next partial product (y<< 2 = 4y)
  - The next PP has 4-times the weight
  - Adding y to the next PP is actually adding 4y to the current PP
- But what about the next PP?
  - 0 + y = y
  - y + y = 2y
  - 2y + y = 3y ⟹ requires an adder.
- Replace 2y with –2y ⟹ add y to the next partial product (y<< 2 = 4y)
  - -2y + y = y ⟹ no adder is required
- Need to check the MSB from the previous pair of bits
  - If it is 1 (case of 2y or 3y in previous PP) the add y.

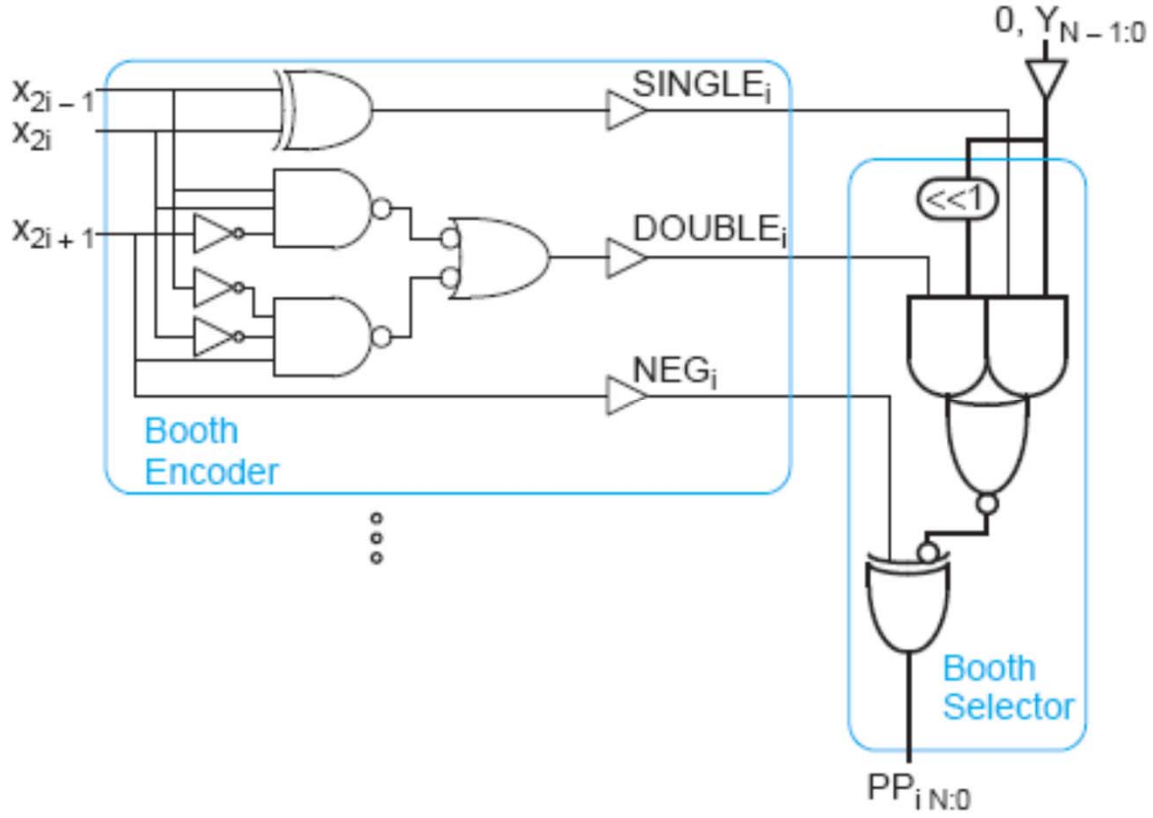
Booth Encoding

- Instead of 3Y, try –Y, then increment next partial product to add 4Y
- Similarly, for 2Y, try –2Y + 4Y in next partial product

Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	SINGLE_i	DOUBLE_i	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

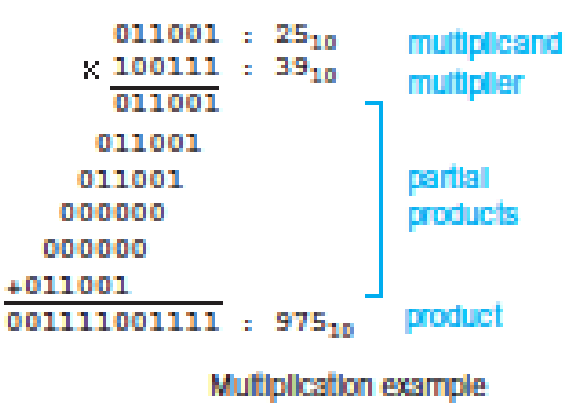
Booth Hardware

- Booth encoder generates control lines for each PP
  - Booth selectors choose PP bits

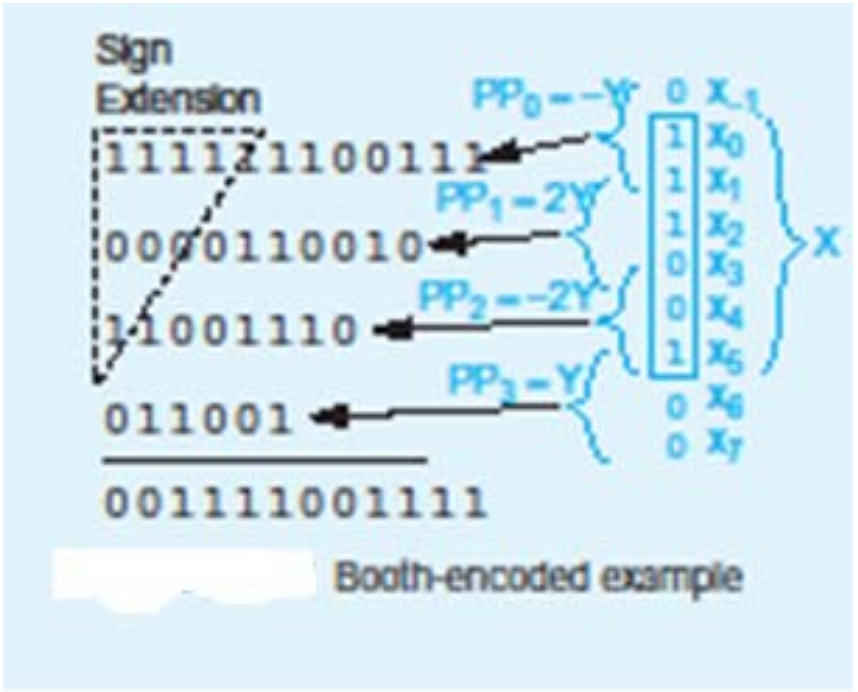


Multiplication Example

$$P = Y \times X = 011001_2 \times 100111_2$$



Multiplication example



Booth-encoded example