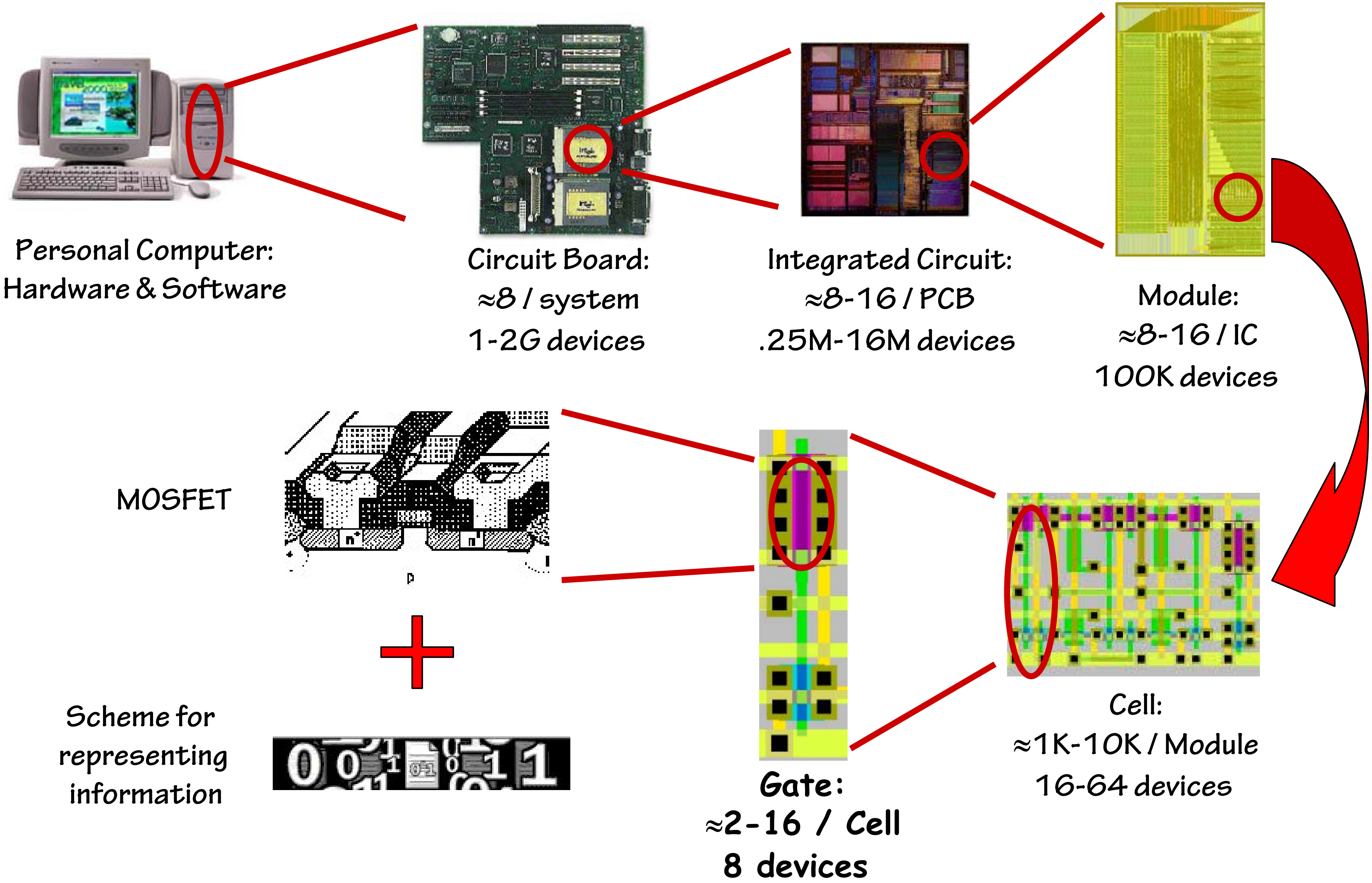
Digital Design & FPGA Workshop

- Week 2 – Combinatorial Logic
  - Representation of Information
  - The MOSFET switch
  - Boolean Logic
  - Timing and Hazards
  - Make sure VMs are setup for next week

How do you build systems with >1G components?



Personal Computer:
Hardware & Software

Circuit Board:
≈8 / system
1-2G devices

Integrated Circuit:
≈8-16 / PCB
.25M-16M devices

Module:
≈8-16 / IC
100K devices

Cell:
≈1K-10K / Module
16-64 devices

Gate:
≈2-16 / Cell
8 devices

MOSFET

+

Scheme for
representing
information

Concrete encoding of information

To this point we've discussed encoding information using bits. But where do bits come from?

If we're going to design a machine that manipulates information, how should that information be physically encoded?

He said to his friend, "If the British march
By land or sea from the town to-night,
Hang a lantern aloft in the belfry arch
Of the North Church tower as a signal light,--
One if by land, and two if by sea;
And I on the opposite shore will be,
Ready to ride and spread the alarm
Through every Middlesex village and farm,
For the country folk to be up and to arm."

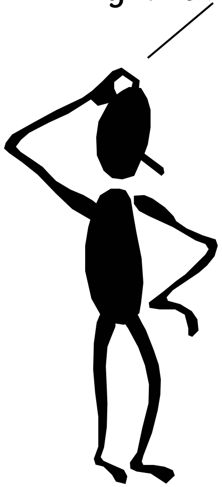
What makes a good bit?

- cheap (we want a lot of them)
- stable (reliable, repeatable)
- ease of manipulation
  (access, transform, combine, transmit, store)

A substrate for computation

We can build upon almost any physical phenomenon

Wait!
Those last ones
might have potential...



~~lanterns~~

~~elephants~~

~~engraved stone tablets~~

~~Billiard balls~~

~~sequences of amino acids~~

~~polarization of a photon~~

But, since we're EE's…

Stick with things we know about:

voltagesphase

currents frequency

This semester we'll use voltages to encode information. But the best choice depends on the intended application…

Voltage pros:

- easy generation, detection
- lots of engineering knowledge
- potentially ~~low~~ power in steady state

zero

Voltage cons:

- easily affected by environment
- DC connectivity required?
- R & C effects slow things down

Representing information with voltage

Representation of each point (x, y) on a B&W Picture:

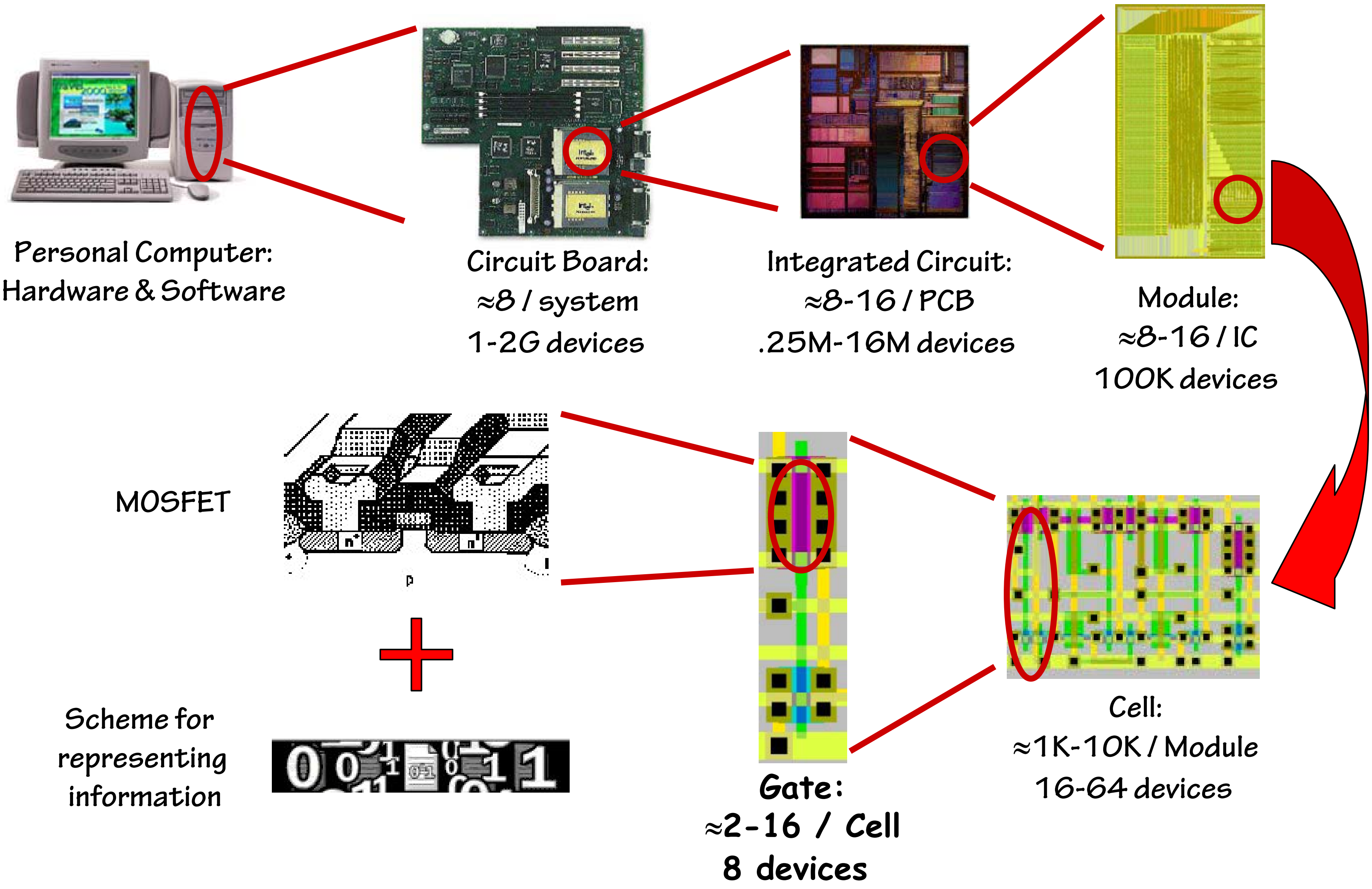
0 volts: BLACK
1 volt: WHITE
0.37 volts: 37% Gray
etc.

Representation of a picture:

Scan points in some prescribed raster order… generate voltage waveform

How much information at each point?

How do you build systems with >1G components?



Personal Computer:
Hardware & Software

Circuit Board:
≈8 / system
1-2G devices

Integrated Circuit:
≈8-16 / PCB
.25M-16M devices

Module:
≈8-16 / IC
100K devices

Cell:
≈1K-10K / Module
16-64 devices

Gate:
≈2-16 / Cell
8 devices

MOSFET

+

Scheme for
representing
information

The Key to System Design

A system is a structure that is guaranteed to exhibit a specified behavior, assuming all of its components obey their specified behaviors.

How is this achieved?

Contracts!

Every system component will have clear obligations and responsibilities. If these are maintained we have every right to expect the system to behave as planned. If contracts are violated all bets are off.

The Digital Panacea ...

Why digital?

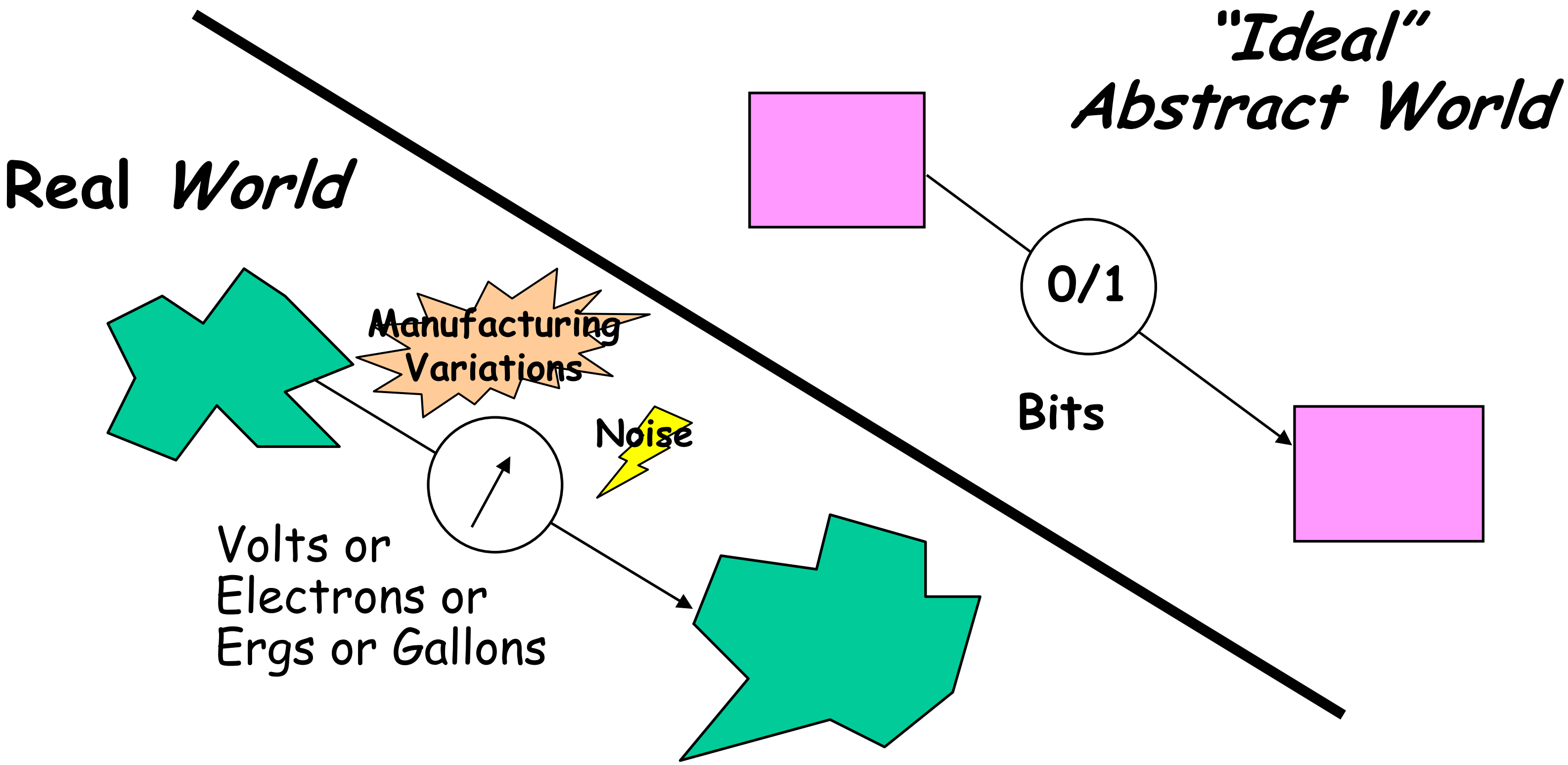
… because it keeps the contracts simple!

The price we pay for this robustness…

All the information that we transfer between modules is only 1 crummy bit!

But, we get a guarantee of reliable processing.

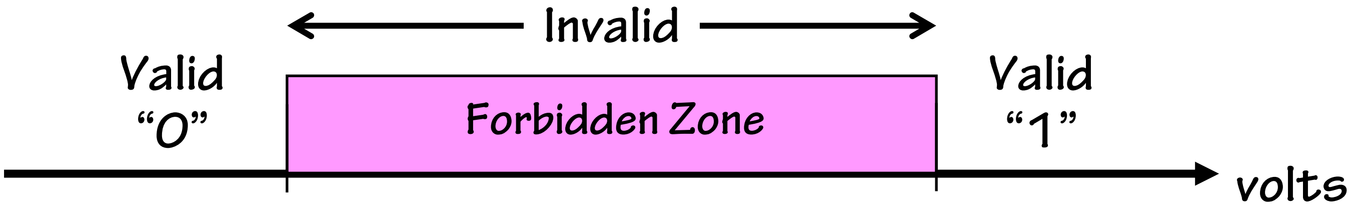
The Digital Abstraction



Keep in mind that the world is not digital, we would simply like to engineer it to behave that way. Furthermore, we must use real physical phenomena to implement digital designs!

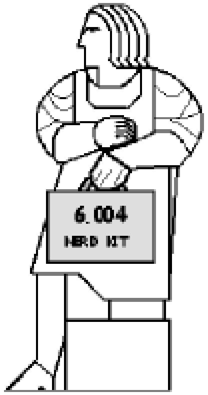
Using Voltages "Digitally"

- Key idea: don't allow "0" to be mistaken for a "1" or vice versa
- Use the same "uniform representation convention", for *every* component and wire in our digital system
- To implement devices with high reliability, we outlaw "close calls" via a representation convention which forbids a range of voltages between "0" and "1".

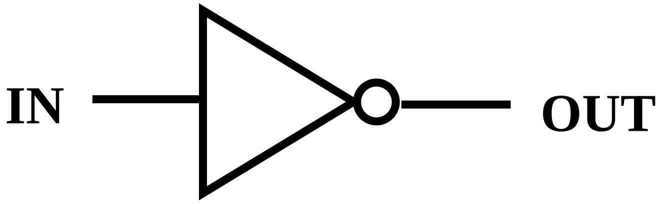


CONSEQUENCE:

Notion of "VALID" and "INVALID" logic levels

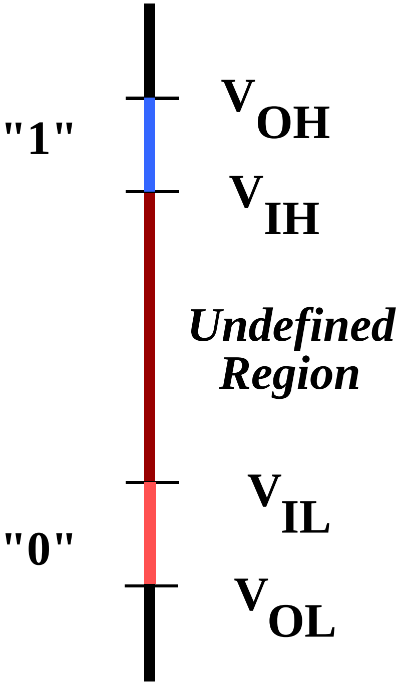


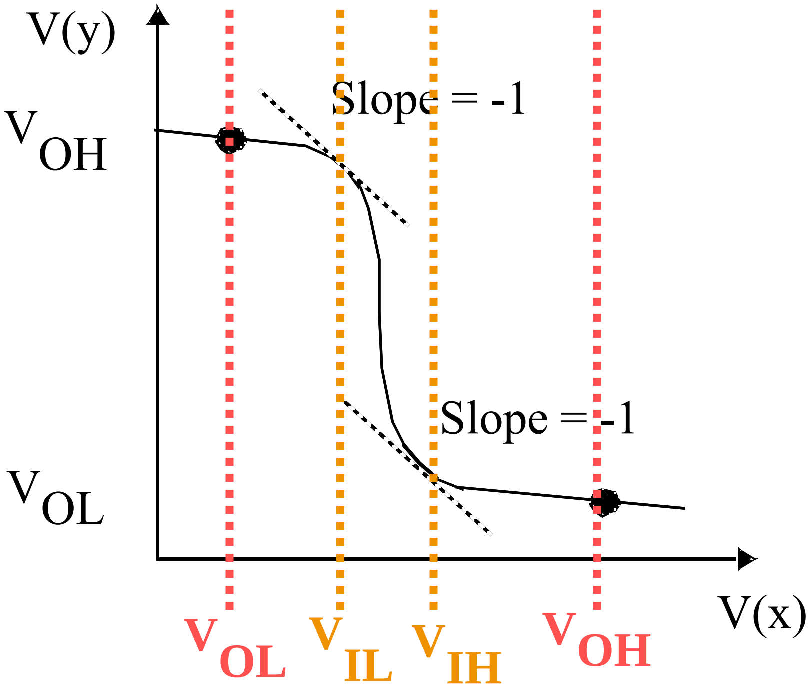
Review: Noise Margin



Truth Table

IN	OUT
0	1
1	0

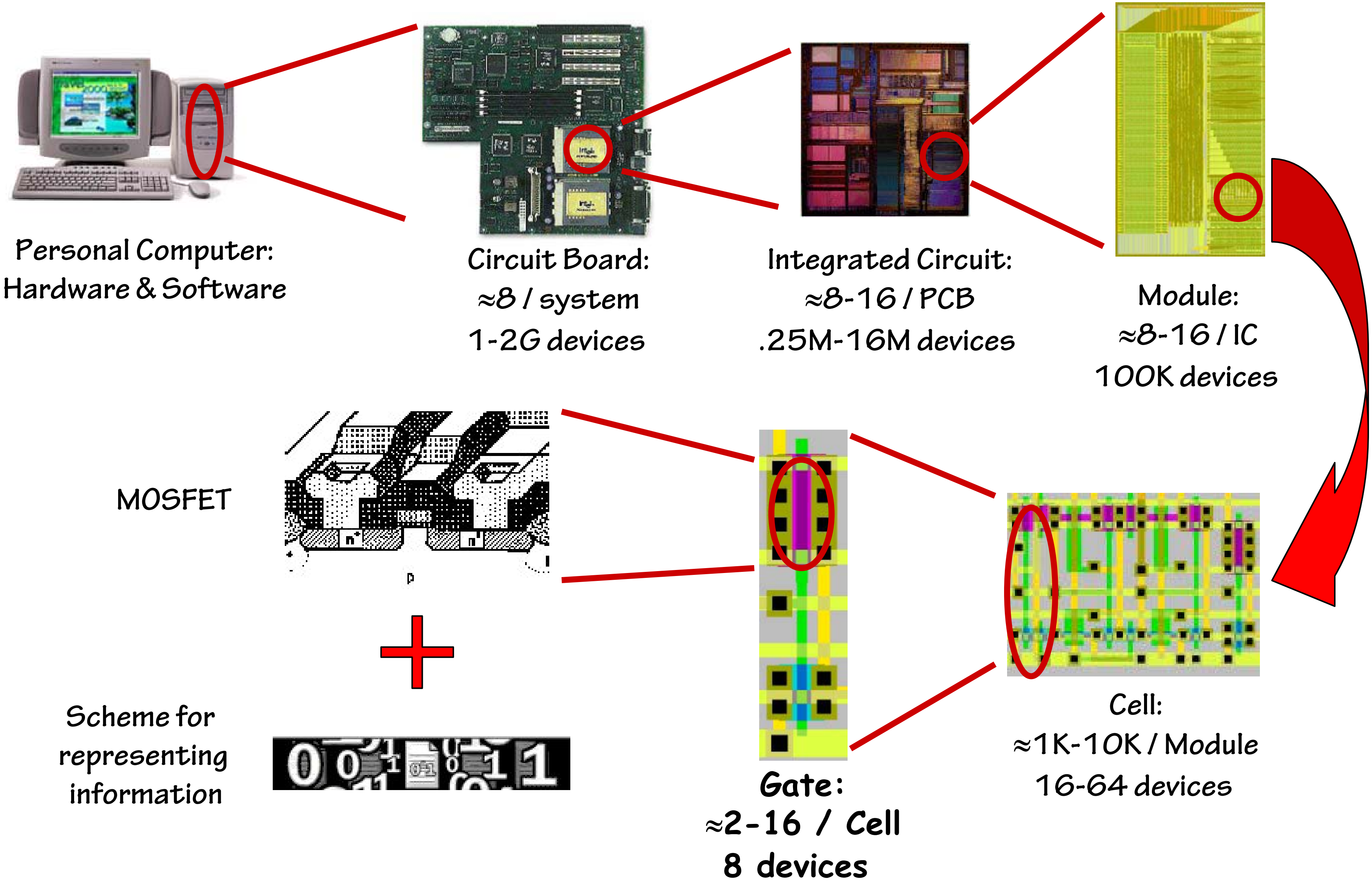




$$NM_L = V_{IL} - V_{OL}$$
$$NM_H = V_{OH} - V_{IH}$$

- **Large noise margins protect against various noise sources**

How do you build systems with >1G components?



Personal Computer:
Hardware & Software

Circuit Board:
≈8 / system
1-2G devices

Integrated Circuit:
≈8-16 / PCB
.25M-16M devices

Module:
≈8-16 / IC
100K devices

Cell:
≈1K-10K / Module
16-64 devices

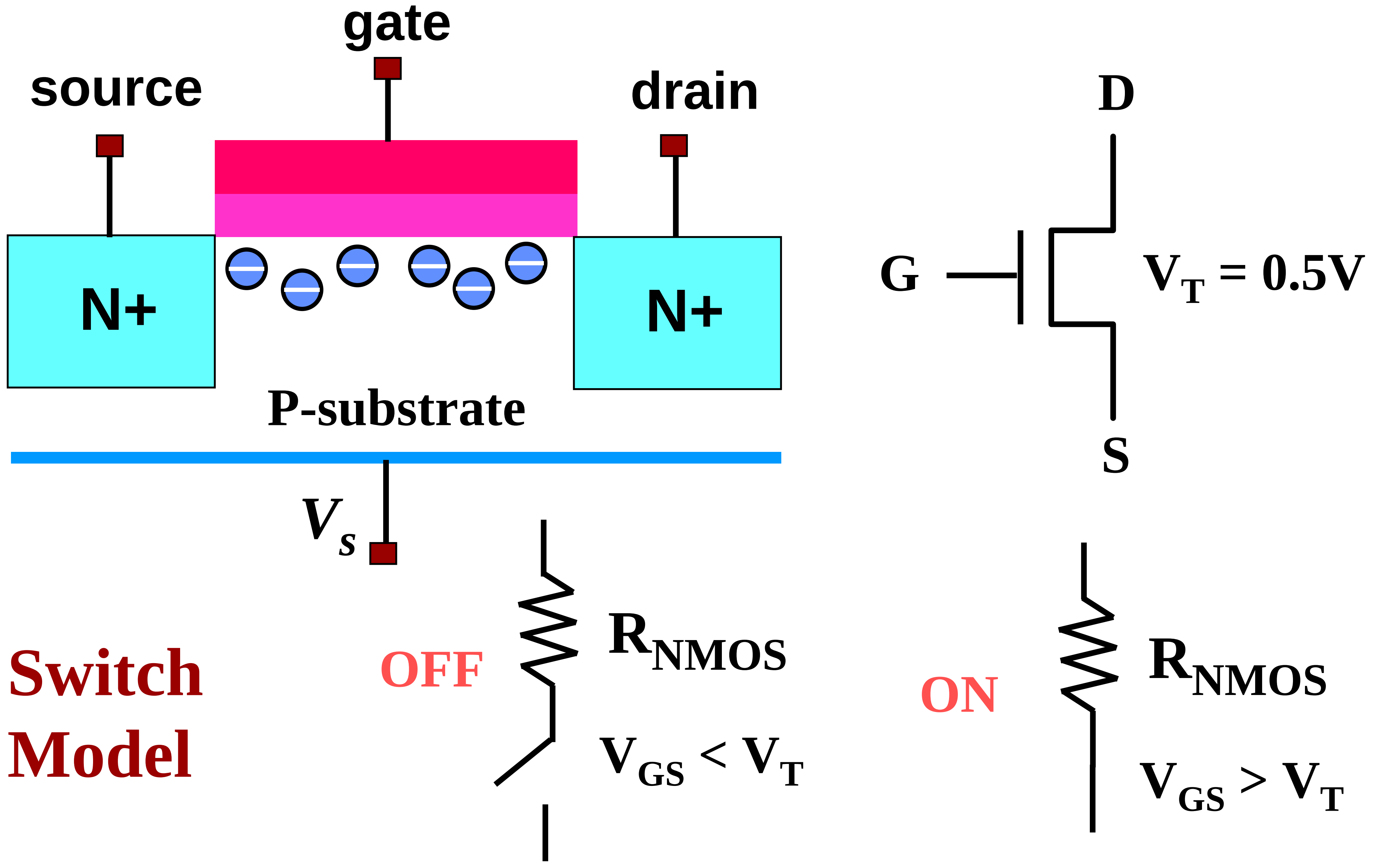
Gate:
≈2-16 / Cell
8 devices

MOSFET

+

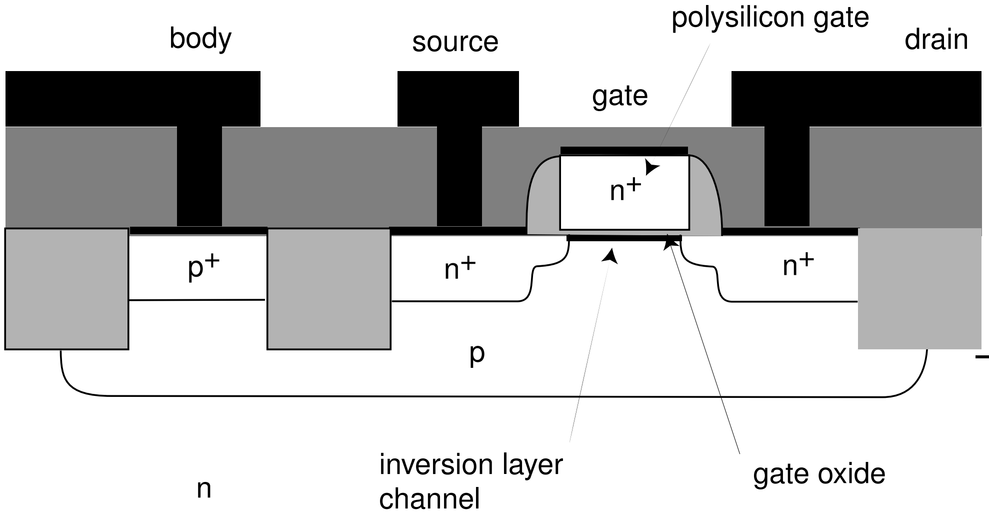
Scheme for
representing
information

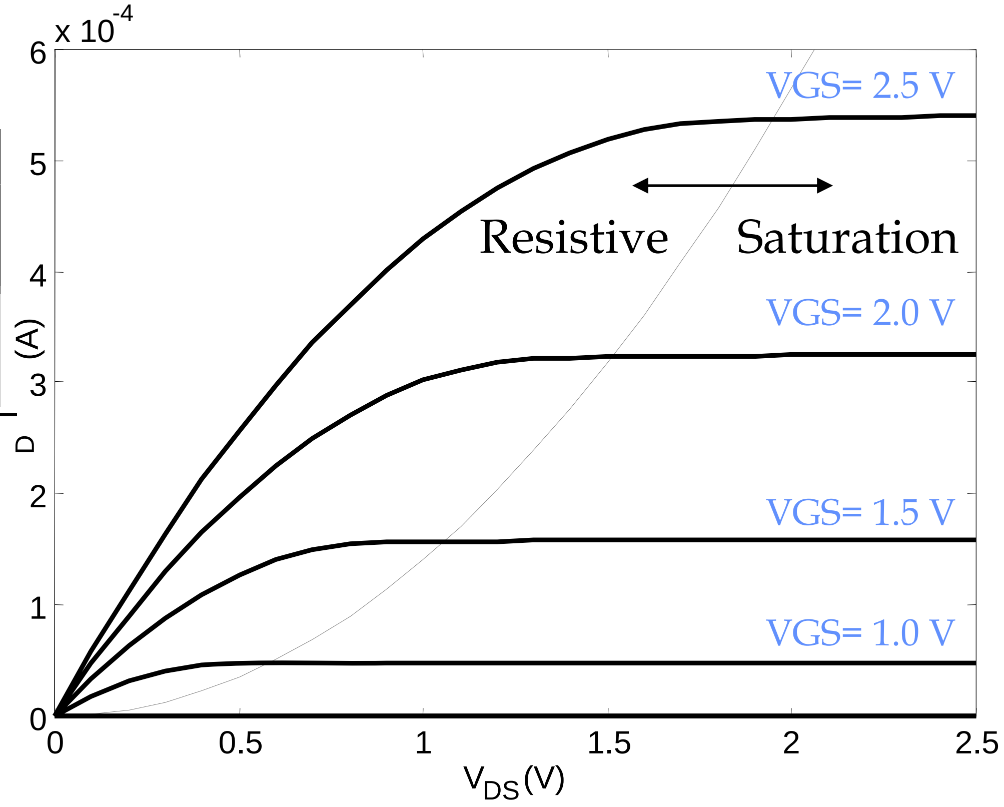
MOS Technology: The NMOS Switch

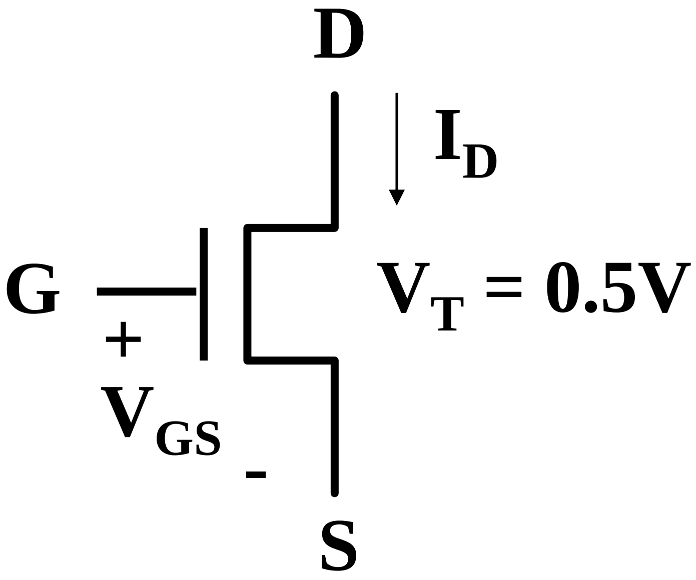


NMOS ON when Switch Input is High

NMOS Device Characteristics



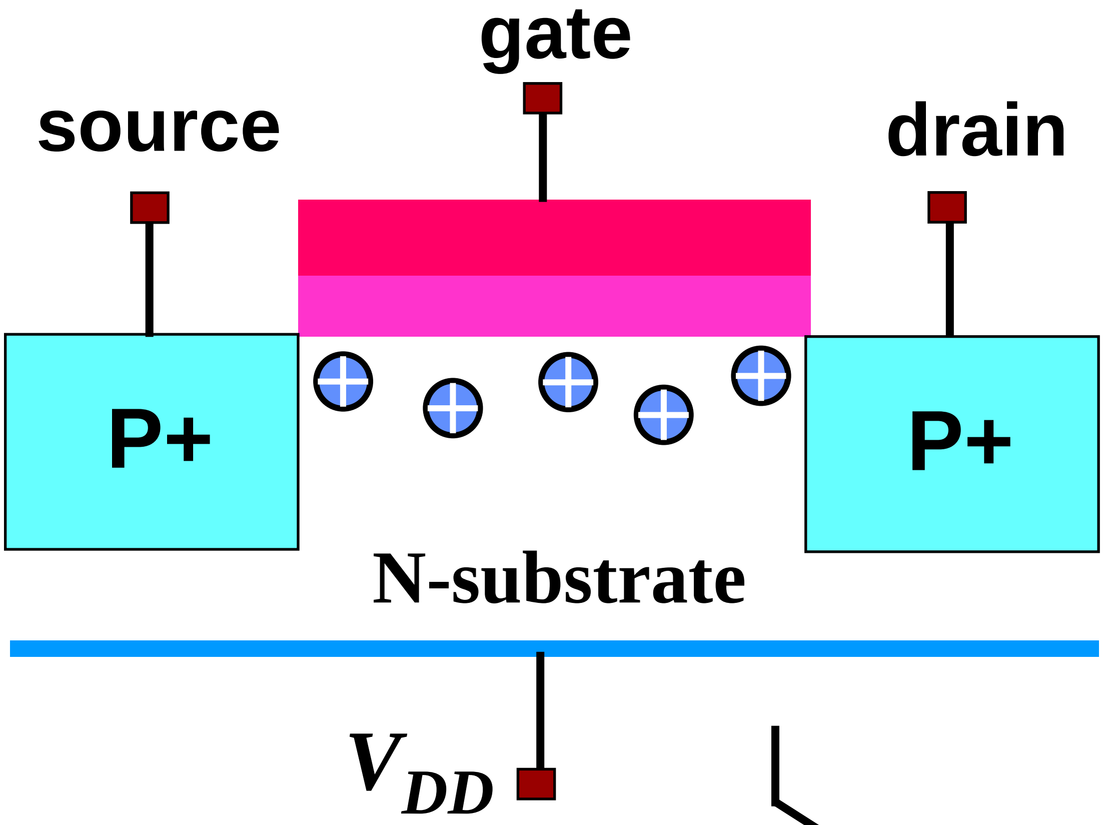


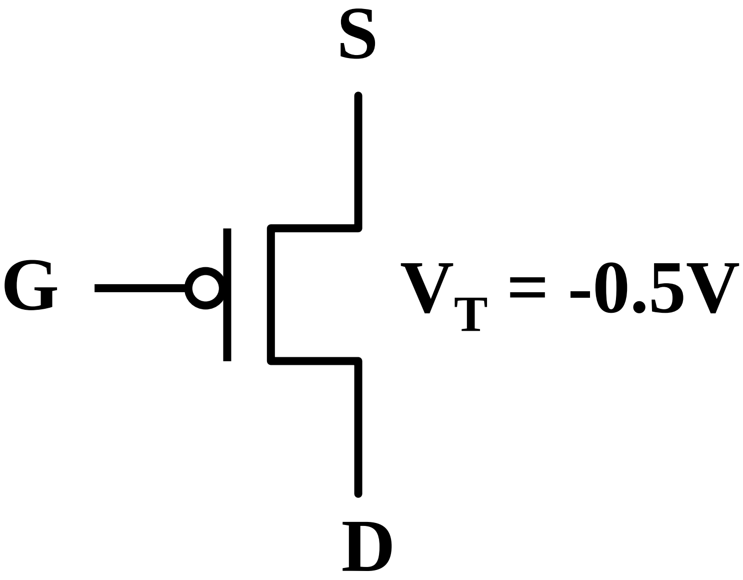


$V_T = 0.5V$

- ➢ **MOS is a very non-linear.**
- ➢ **Switch-resistor model sufficient for first order analysis.**

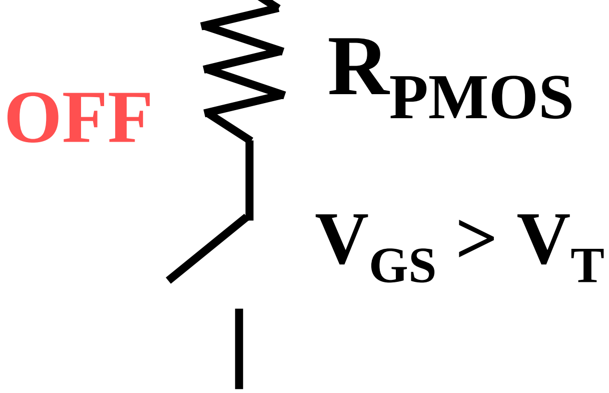
PMOS: The Complementary Switch

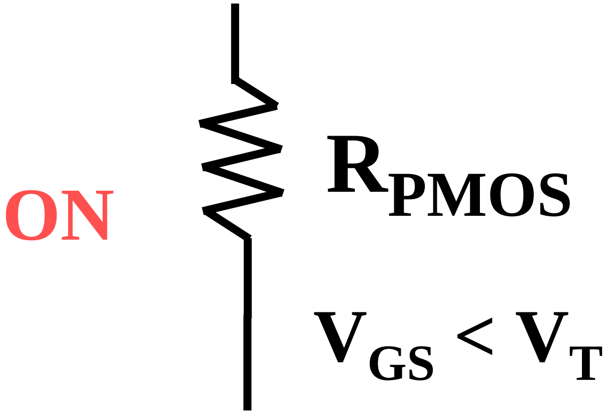




Switch

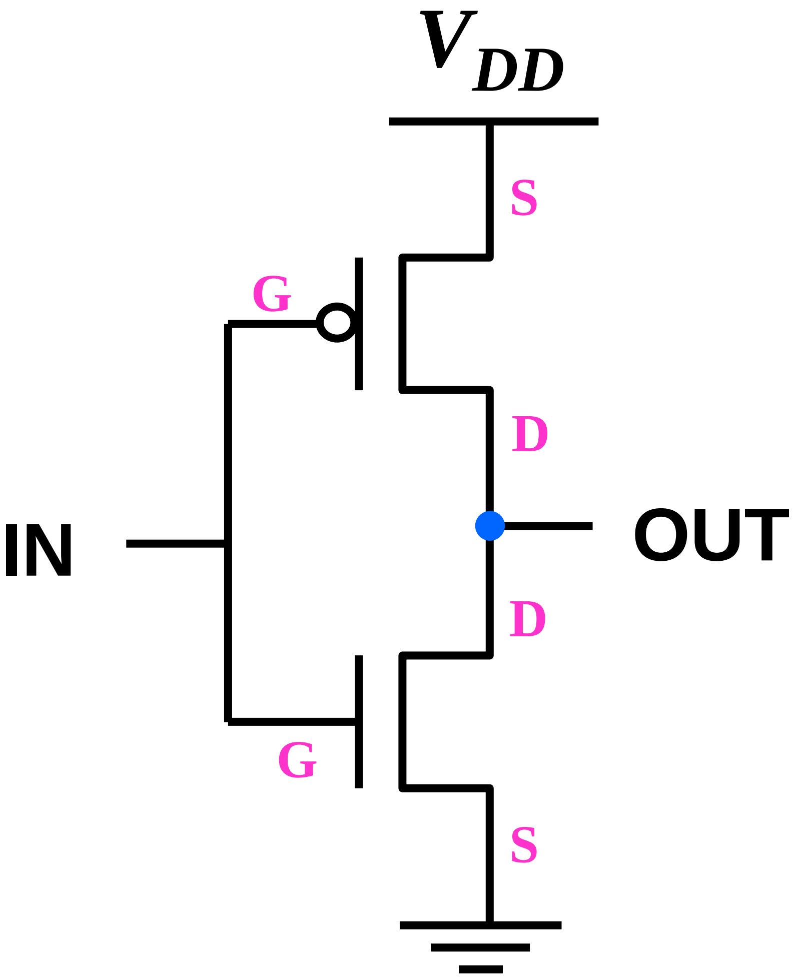
Model



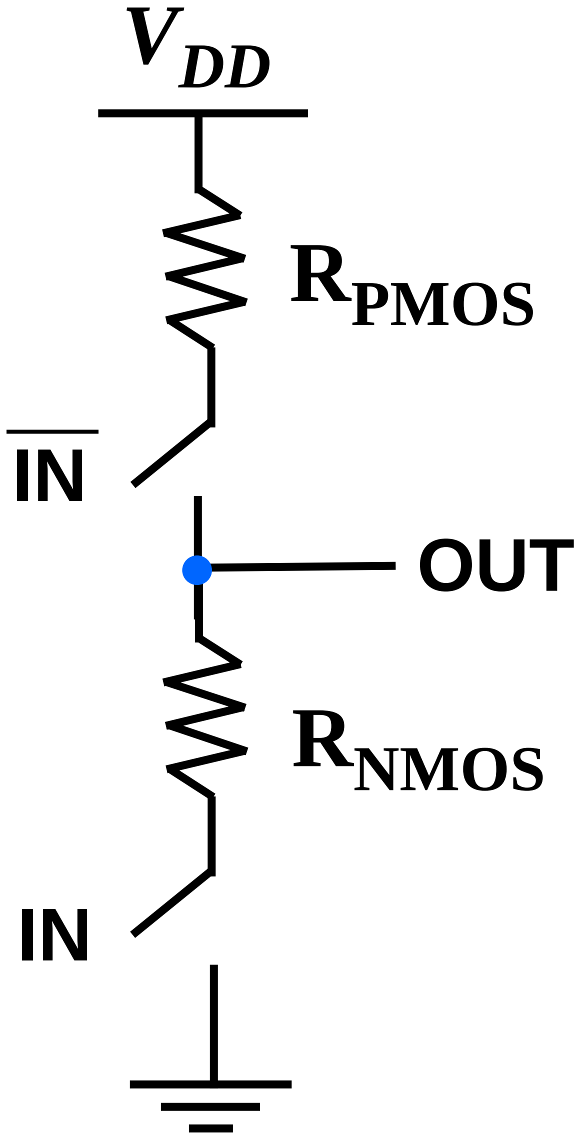


PMOS ON when Switch Input is Low

The CMOS Inverter

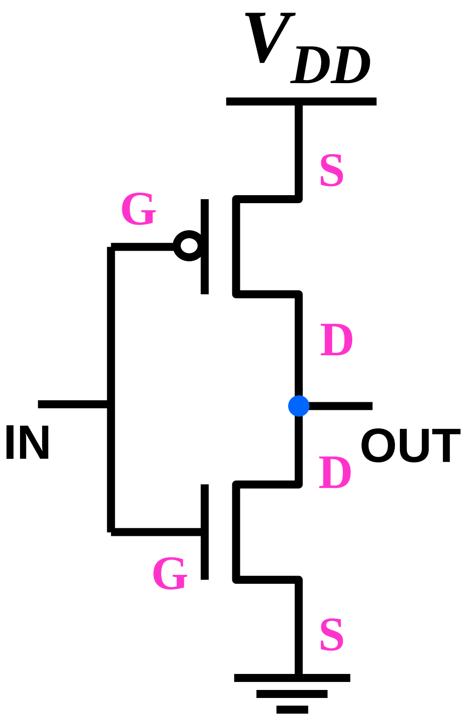


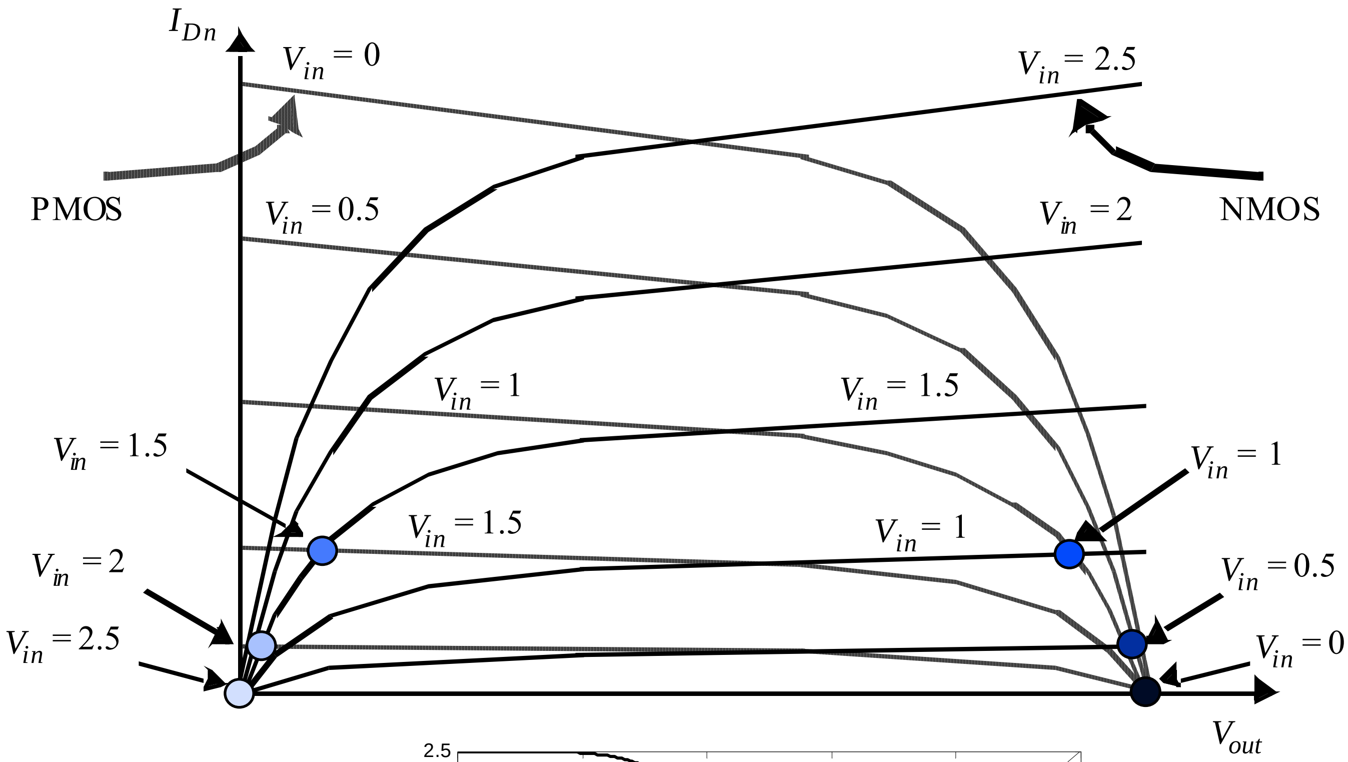
Switch Model

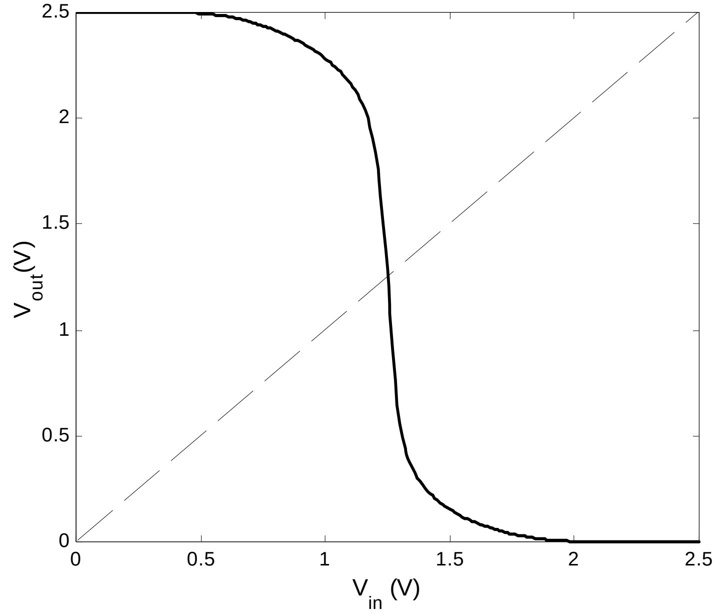


Rail-to-rail Swing in CMOS

Inverter VTC: Load Line Analysis



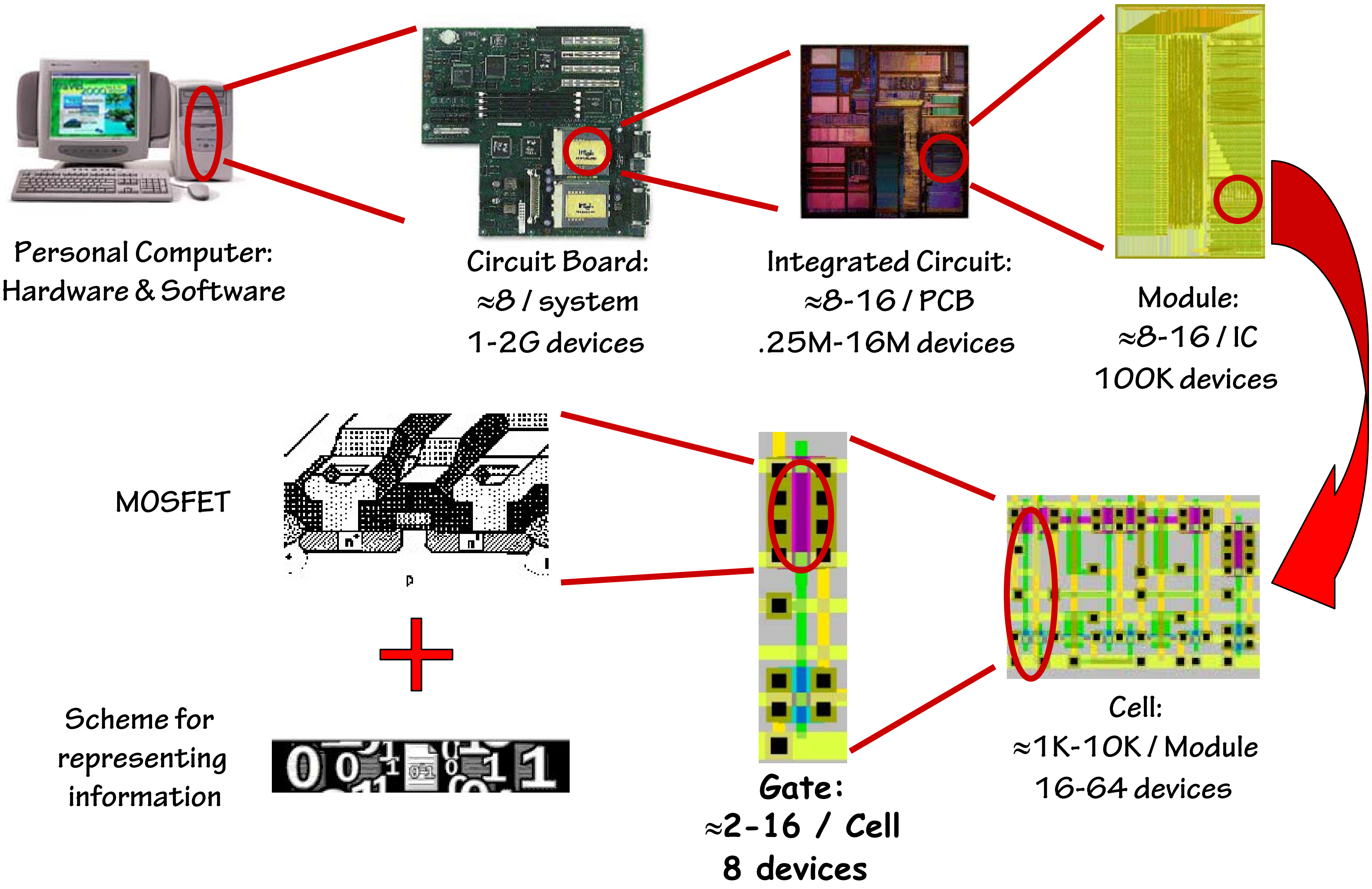




CMOS gates have:

- Rail-to-rail swing (0V to V_{DD})
- Large noise margins
- “zero” static power dissipation

How do you build systems with >1G components?



Personal Computer:
Hardware & Software

Circuit Board:
≈8 / system
1-2G devices

Integrated Circuit:
≈8-16 / PCB
.25M-16M devices

Module:
≈8-16 / IC
100K devices

Cell:
≈1K-10K / Module
16-64 devices

Gate:
≈2-16 / Cell
8 devices

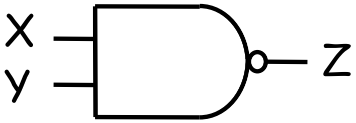
MOSFET

Scheme for
representing
information

Common Logic Gates

Gate | Symbol | Truth-Table | Expression

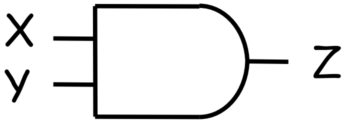
NAND



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

$Z = \overline{X \bullet Y}$

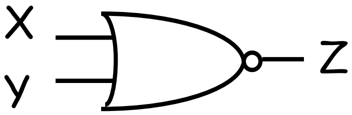
AND



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

$Z = X \bullet Y$

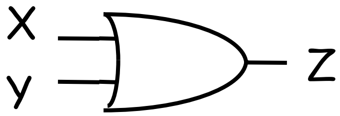
NOR



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

$Z = \overline{X + Y}$

OR



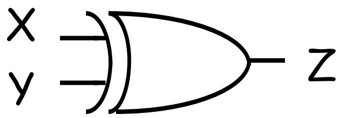
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

$Z = X + Y$

Exclusive (N)OR Gate

XOR

$(X \oplus Y)$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

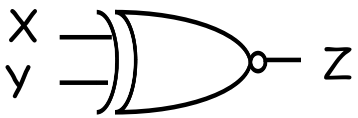
$Z = X\overline{Y} + \overline{X}Y$

X or Y but not both

("inequality", "difference")

XNOR

$\overline{(X \oplus Y)}$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

$Z = \overline{X}\,\overline{Y} + XY$

X and Y the same

("equality")

Widely used in arithmetic structures such as adders and multipliers

Theorems of Boolean Algebra (I)

- **Elementary**

1. $X + 0 = X$ 1D. $X \bullet 1 = X$

2. $X + 1 = 1$ 2D. $X \bullet 0 = 0$

3. $X + X = X$ 3D. $X \bullet X = X$

4. $(\overline{\overline{X}}) = X$

5. $X + \overline{X} = 1$ 5D. $X \bullet \overline{X} = 0$

- **Commutativity:**

6. $X + Y = Y + X$ 6D. $X \bullet Y = Y \bullet X$

- **Associativity:**

7. $(X + Y) + Z = X + (Y + Z)$ 7D. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$

- **Distributivity:**

8. $X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z)$ 8D. $X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$

- **Uniting:**

9. $X \bullet Y + X \bullet \overline{Y} = X$ 9D. $(X + Y) \bullet (X + \overline{Y}) = X$

- **Absorption:**

10. $X + X \bullet Y = X$ 10D. $X \bullet (X + Y) = X$

11. $(X + \overline{Y}) \bullet Y = X \bullet Y$ 11D. $(X \bullet \overline{Y}) + Y = X + Y$

Theorems of Boolean Algebra (II)

- **Factoring:**

  12. $(X \bullet Y) + (X \bullet Z) =$
  $X \bullet (Y + Z)$

  12D. $(X + Y) \bullet (X + Z) =$
  $X + (Y \bullet Z)$

- **Consensus:**

  13. $(X \bullet Y) + (Y \bullet Z) + (\overline{X} \bullet Z) =$
  $X \bullet Y + \overline{X} \bullet Z$

  13D. $(X + Y) \bullet (Y + Z) \bullet (\overline{X} + Z) =$
  $(X + Y) \bullet (\overline{X} + Z)$

- **De Morgan's:**

  14. $\overline{(X + Y + \ldots)} = \overline{X} \bullet \overline{Y} \bullet \ldots$

  14D. $\overline{(X \bullet Y \bullet \ldots)} = \overline{X} + \overline{Y} + \ldots$

- **Generalized De Morgan's:**

  15. $\overline{f}(X1,X2,\ldots,Xn,0,1,+,\bullet) = f(\overline{X1},\overline{X2},\ldots,\overline{Xn},1,0,\bullet,+)$

- **Duality**
  - □ Dual of a Boolean expression is derived by replacing • by +, + by •, 0 by 1, and 1 by 0, and leaving variables unchanged
  - □ $f\,(X1,X2,\ldots,Xn,0,1,+,\bullet) \Leftrightarrow f(X1,X2,\ldots,Xn,1,0,\bullet,+)$

There are only so many gates

There are only 16 possible 2-input gates

… some we know already, others are just silly

INPUT AB	ZERO	AND	A > B	A	B > A	B	XOR	OR	NOR	XNOR	NOT ‘B’	A <= B	NOT ‘A’	B <= A	NAND	ONE
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

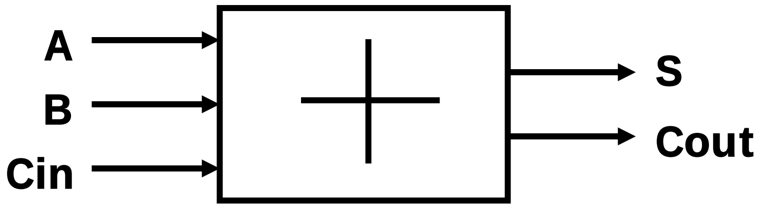
How many of these gates can be implemented using a single CMOS gate?

Do we need all of these gates?

Nope. After all, we describe them all using AND, OR, and NOT.

Simple Example: One Bit Adder

- 1-bit binary adder
  - □ inputs: A, B, Carry-in
  - □ outputs: Sum, Carry-out



A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum-of-Products Canonical Form

$$S = \overline{A}\ \overline{B}\ Cin + \overline{A}\ B\ \overline{Cin} + A\ \overline{B}\ \overline{Cin} + A\ B\ Cin$$

$$Cout = \overline{A}\ B\ Cin + A\ \overline{B}\ Cin + A\ B\ \overline{Cin} + A\ B\ Cin$$

- Product term (or minterm)
  - □ ANDed product of literals – input combination for which output is true
  - □ Each variable appears exactly once, in true or inverted form (but not both)

Simplify Boolean Expressions

$$\begin{aligned}
\text{Cout} &= \overline{A}\,B\,\text{Cin} + A\,\overline{B}\,\text{Cin} + A\,B\,\overline{\text{Cin}} + A\,B\,\text{Cin} \\
&= \overline{A}\,B\,\text{Cin} + A\,B\,\text{Cin} + A\,\overline{B}\,\text{Cin} + A\,B\,\text{Cin} + A\,B\,\overline{\text{Cin}} + A\,B\,\text{Cin} \\
&= (\overline{A} + A)\,B\,\text{Cin} + A\,(\overline{B} + B)\,\text{Cin} + A\,B\,(\overline{\text{Cin}} + \text{Cin}) \\
&= B\,\text{Cin} + A\,\text{Cin} + A\,B \\
&= (B + A)\,\text{Cin} + A\,B
\end{aligned}$$

$$\begin{aligned}
S &= \overline{A}\,\overline{B}\,\text{Cin} + \overline{A}\,B\,\overline{\text{Cin}} + A\,\overline{B}\,\overline{\text{Cin}} + A\,B\,\text{Cin} \\
&= (\overline{A}\,\overline{B} + A\,B\,)\text{Cin} + (A\,\overline{B} + \overline{A}\,B)\,\overline{\text{Cin}} \\
&= \overline{(A \oplus B)}\,\text{Cin} + (A \oplus B)\,\overline{\text{Cin}} \\
&= A \oplus B \oplus \text{Cin}
\end{aligned}$$

Sum-of-Products & Product-of-Sum

- **Product term** (or minterm): ANDed product of literals – input combination for which output is true

A	B	C	minterms	
0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	m0
0	0	1	$\overline{A}\,\overline{B}\,C$	m1
0	1	0	$\overline{A}\,B\,\overline{C}$	m2
0	1	1	$\overline{A}\,B\,C$	m3
1	0	0	$A\,\overline{B}\,\overline{C}$	m4
1	0	1	$A\,\overline{B}\,C$	m5
1	1	0	$A\,B\,\overline{C}$	m6
1	1	1	$A\,B\,C$	m7

short-hand notation form in terms of 3 variables

F in canonical form:

$$F(A, B, C) = \Sigma m(1,3,5,6,7)$$
$$= m1 + m3 + m5 + m6 + m7$$
$$F = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C} + ABC$$

canonical form ≠ minimal form

$$F(A, B, C) = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\overline{B}\,C + ABC + AB\overline{C}$$
$$= (\overline{A}\,\overline{B} + \overline{A}\,B + A\overline{B} + AB)C + AB\overline{C}$$
$$= ((\overline{A} + A)(\overline{B} + B))C + AB\overline{C}$$
$$= C + AB\overline{C} = AB\overline{C} + C = AB + C$$

- **Sum term** (or maxterm) - ORed sum of literals – input combination for which output is false

A	B	C	maxterms	
0	0	0	$A + B + C$	M0
0	0	1	$A + B + \overline{C}$	M1
0	1	0	$A + \overline{B} + C$	M2
0	1	1	$A + \overline{B} + \overline{C}$	M3
1	0	0	$\overline{A} + B + C$	M4
1	0	1	$\overline{A} + B + \overline{C}$	M5
1	1	0	$\overline{A} + \overline{B} + C$	M6
1	1	1	$\overline{A} + \overline{B} + \overline{C}$	M7

short-hand notation for maxterms of 3 variables

F in canonical form:

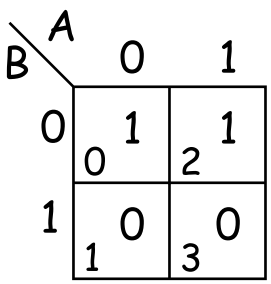
$$F(A, B, C) = \Pi M(0,2,4)$$
$$= M0 \cdot M2 \cdot M4$$
$$= (A + B + C)\,(A + \overline{B} + C)\,(\overline{A} + B + C)$$

canonical form ≠ minimal form

$$F(A, B, C) = (A + B + C)\,(A + \overline{B} + C)\,(\overline{A} + B + C)$$
$$= (A + B + C)\,(A + \overline{B} + C)$$
$$(A + B + C)\,(\overline{A} + B + C)$$
$$= (A + C)\,(B + C)$$

Karnaugh Maps

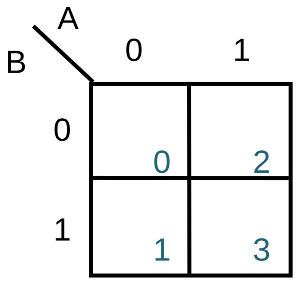
- Alternative to truth-tables to help visualize adjacencies
  - Guide to applying the uniting theorem - On-set elements with only one variable changing value are adjacent unlike in a linear truth-table



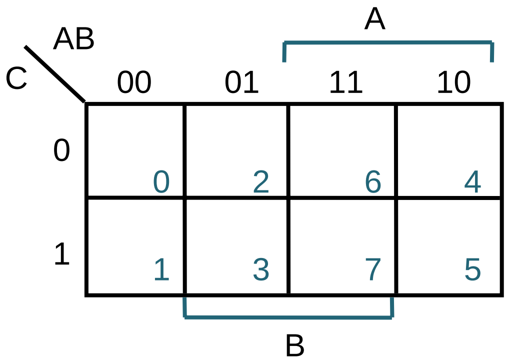
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

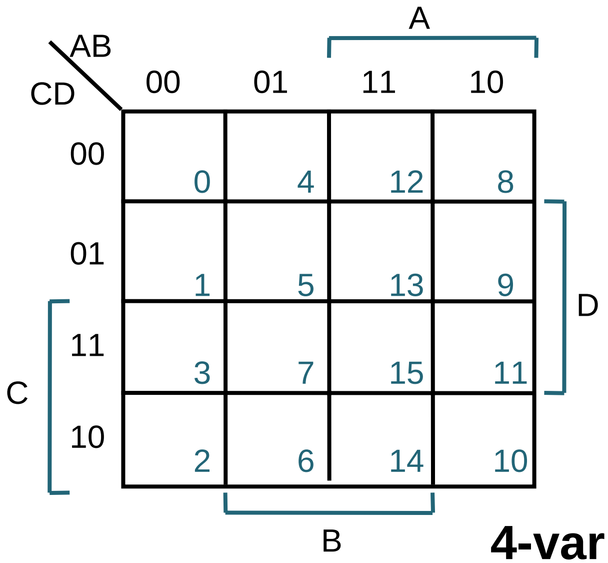
- Numbering scheme based on Gray–code
  - e.g., 00, 01, 11, 10 (only a single bit changes in code for adjacent map cells)

2-variable K-map



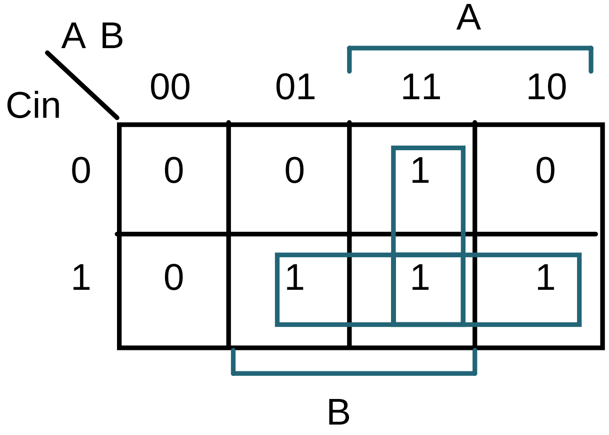
3-variable K-map





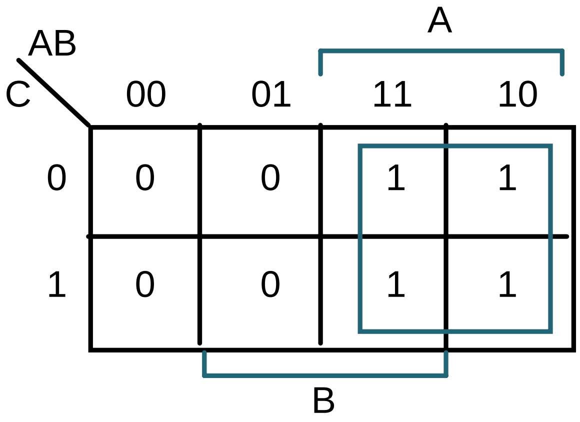
4-variable K-map

K-Map Examples



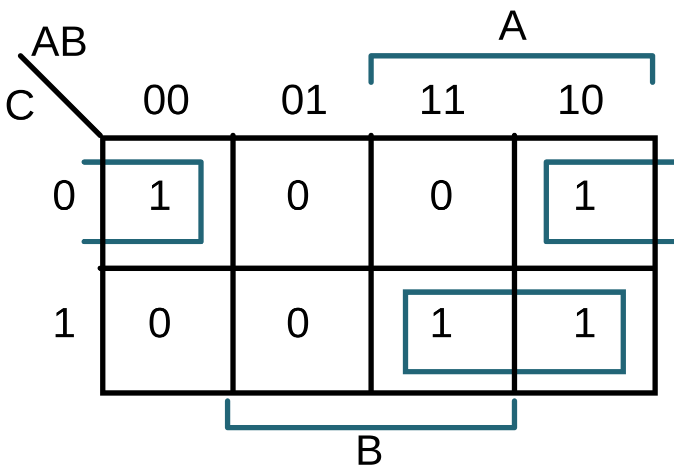
Cin \ A B	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Cout =



C \ AB	00	01	11	10
0	0	0	1	1
1	0	0	1	1

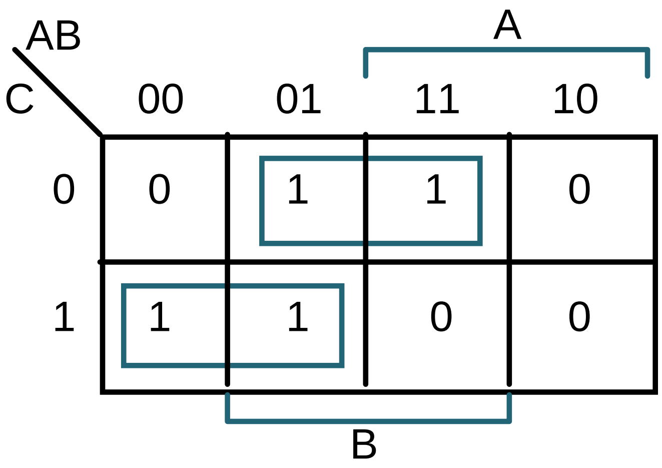
F(A,B,C) =



C \ AB	00	01	11	10
0	1	0	0	1
1	0	0	1	1

F(A,B,C) = Σm(0,4,5,7)

F =



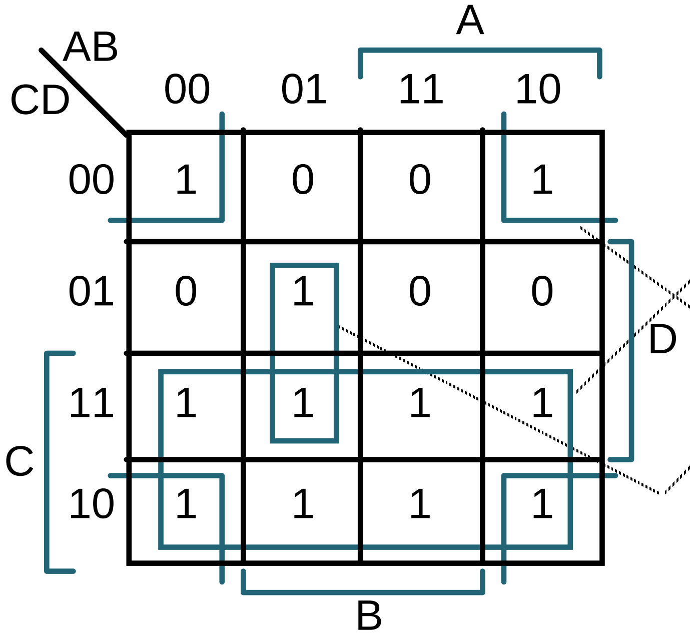
C \ AB	00	01	11	10
0	0	1	1	0
1	1	1	0	0

F' simply replace 1's with 0's and vice versa

F'(A,B,C) = Σm(1,2,3,6)

F' =

Four Variable Karnaugh Map

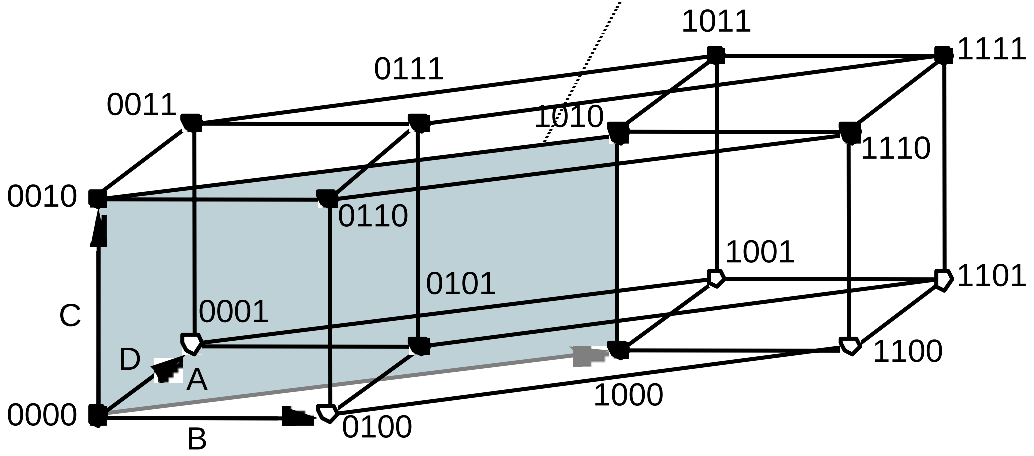


CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

$F(A,B,C,D) = \Sigma m(0,2,3,5,6,7,8,10,11,14,15)$

$F = C + \overline{A}\,B\,D + \overline{B}\,\overline{D}$

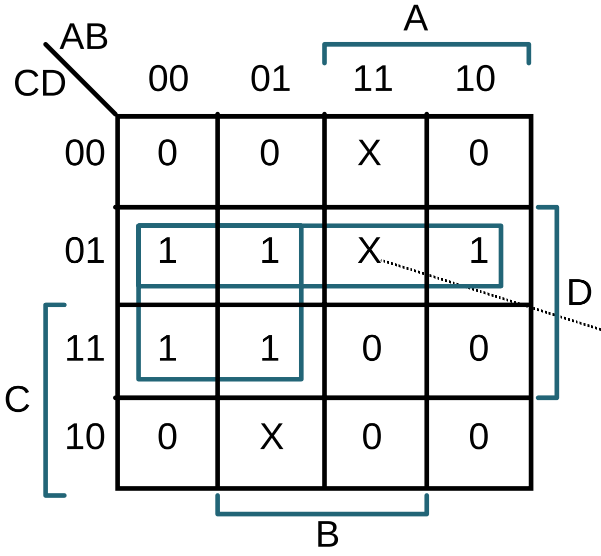
Find the smallest number of the largest possible subcubes that cover the ON-set



K-map Corner Adjacency Illustrated in the 4-Cube

K-Map Example: Don’t Cares

Don't Cares can be treated as 1's or 0's if it is advantageous to do so



CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

$F(A,B,C,D) = \Sigma m(1,3,5,7,9) + \Sigma d(6,12,13)$

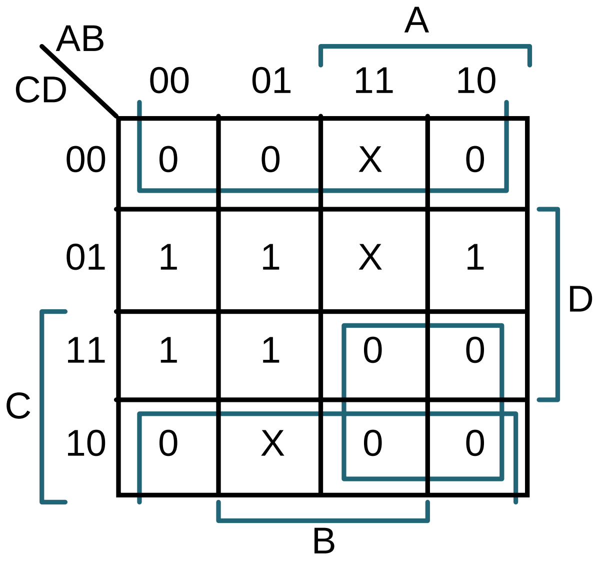
$F = \overline{A}\,D + \overline{B}\;\overline{C}\;D$ w/o don't cares

$F = \overline{C}\;D + \overline{A}\;D$ w/ don't cares

By treating this DC as a "1", a 2-cube can be formed rather than one 0-cube

In PoS form: $F = D\,(\overline{A} + \overline{C})$

Equivalent answer as above, but fewer literals

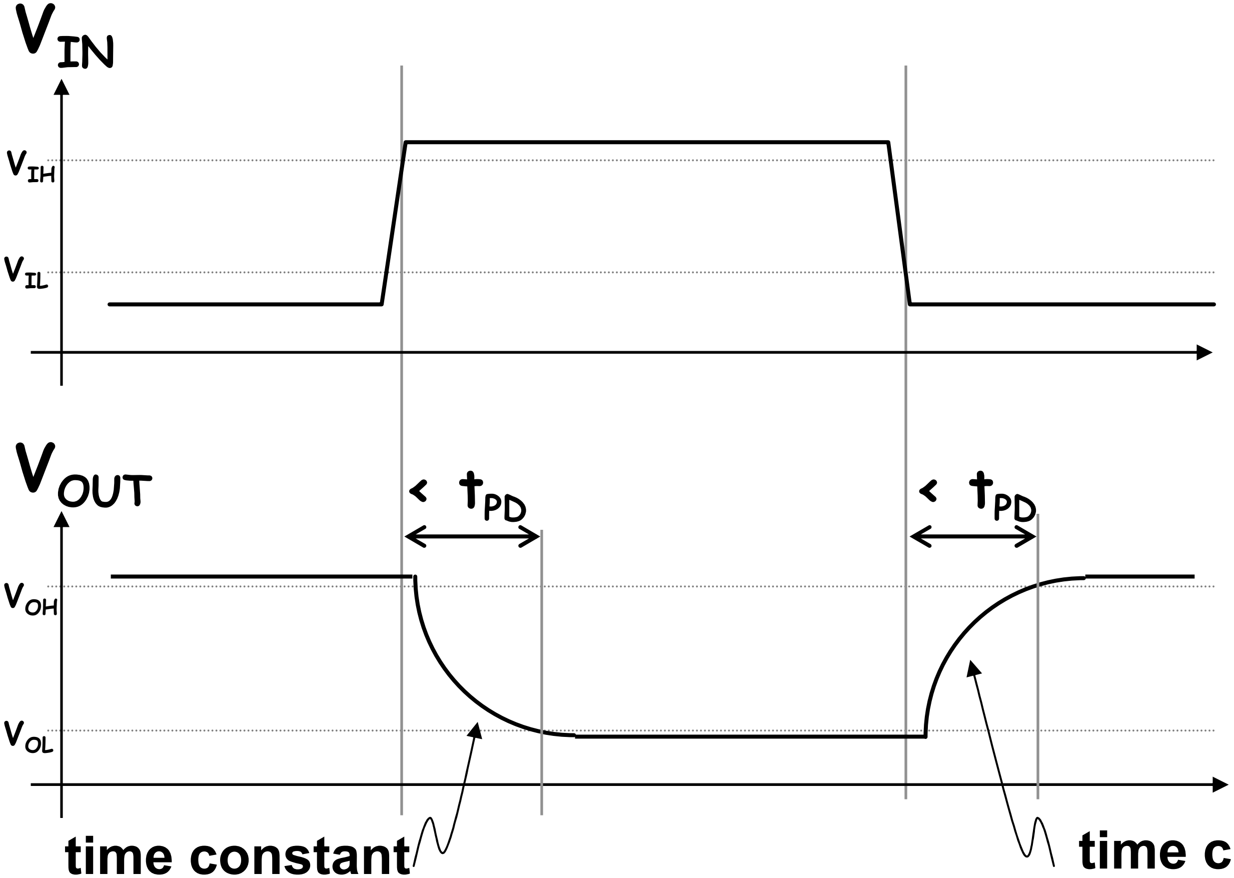


CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

Due to unavoidable delays…

Propagation delay (t_{PD}):

An UPPER BOUND on the delay from valid inputs to valid outputs.



GOAL:

minimize **propagation delay!**

ISSUE:

keep Capacitances low and transistors fast

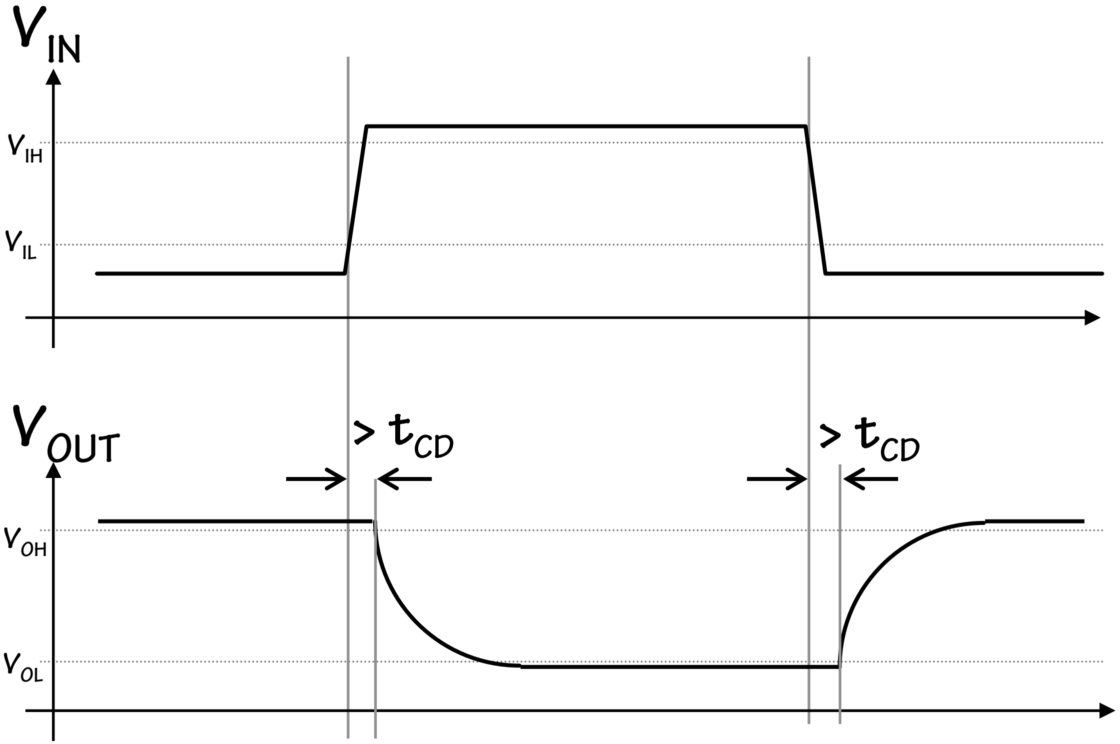
time constant
$\tau = R_{PD} \bullet C_L$

time constant
$\tau = R_{PU} \bullet C_L$

Contamination Delay

an optional, additional timing spec

INVALID inputs take time to propagate, too...



Do we really need t_{CD}?

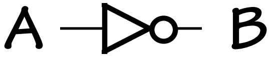
Usually not… it'll be important when we design circuits with registers (coming soon!)

If t_{CD} is not specified, safe to assume it's 0.

CONTAMINATION DELAY, t_{CD}

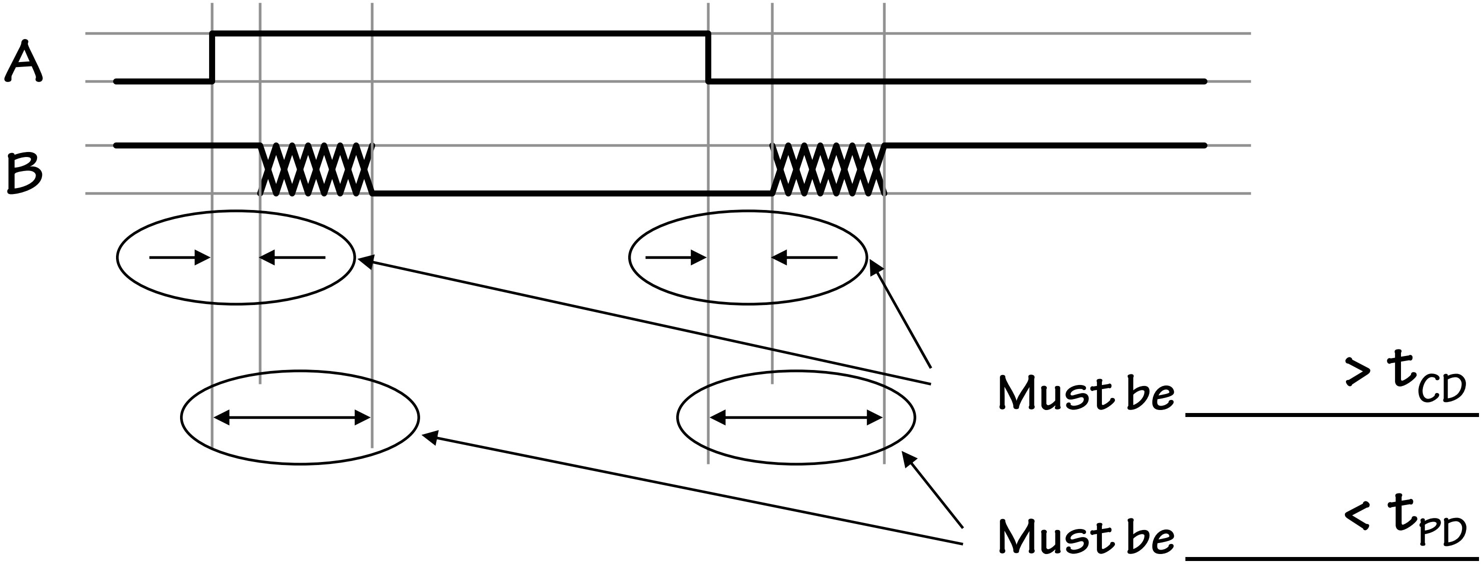
A LOWER BOUND on the delay from any invalid input to an invalid output

The Combinational Contract



A	B
0	1
1	0

t_{PD} propagation delay
t_{CD} contamination delay



Must be $> t_{CD}$

Must be $< t_{PD}$

Note:

1. *No Promises* during
2. Default (conservative) spec: $t_{CD} = 0$

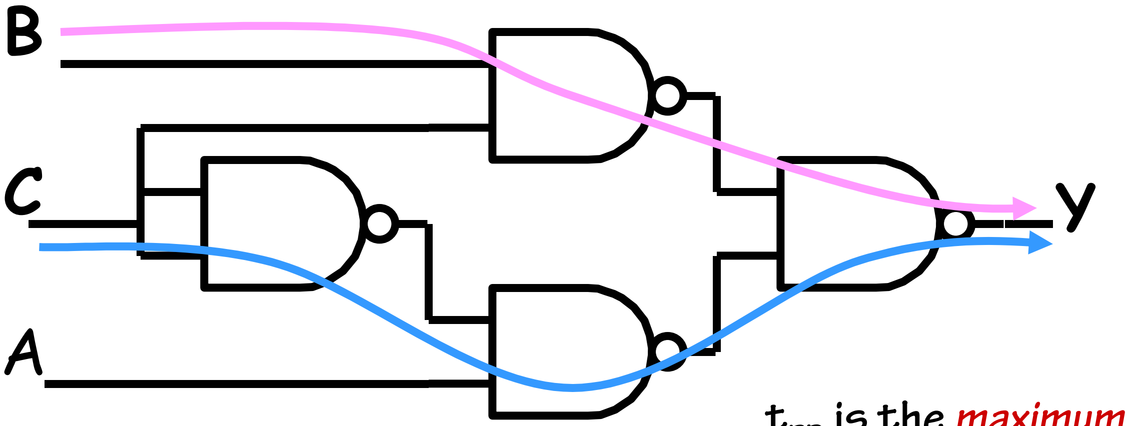
Example: Timing Analysis

If NAND gates have a t_{PD} = 4nS and t_{CD} = 1nS

t_{PD} = ____12____ nS

t_{CD} = ____2____ nS

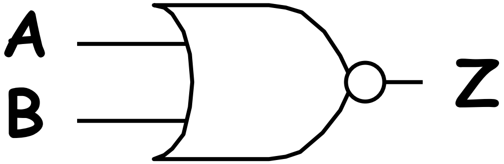
t_{CD} is the *minimum* cumulative contamination delay over all paths from inputs to outputs



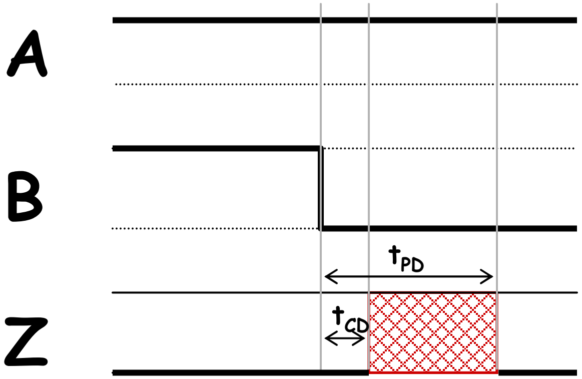
t_{PD} is the *maximum* cumulative propagation delay over all paths from inputs to outputs

Oh yeah... one last issue

NOR:



A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0



Recall the rules for *combinational devices*:

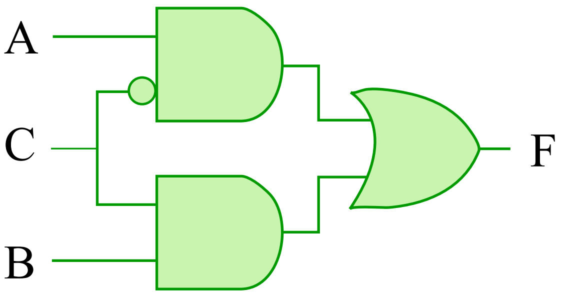
Output guaranteed to be valid when all inputs have been valid for at least t_{PD}, and, outputs may become invalid no earlier than t_{CD} after an input changes!

Many gate implementations--e.g., CMOS—
adhere to even tighter restrictions.

Hazards

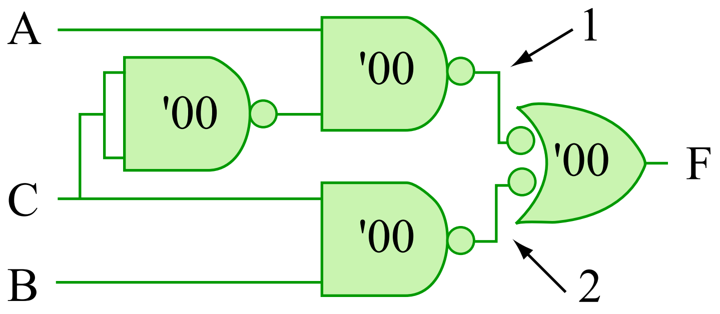
Static hazards: Consider this function:

$$F = A * \overline{C} + B * C$$



C \ AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Implemented with MSI gates:



A = B = 1

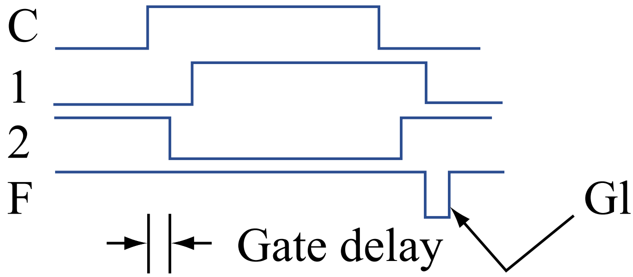
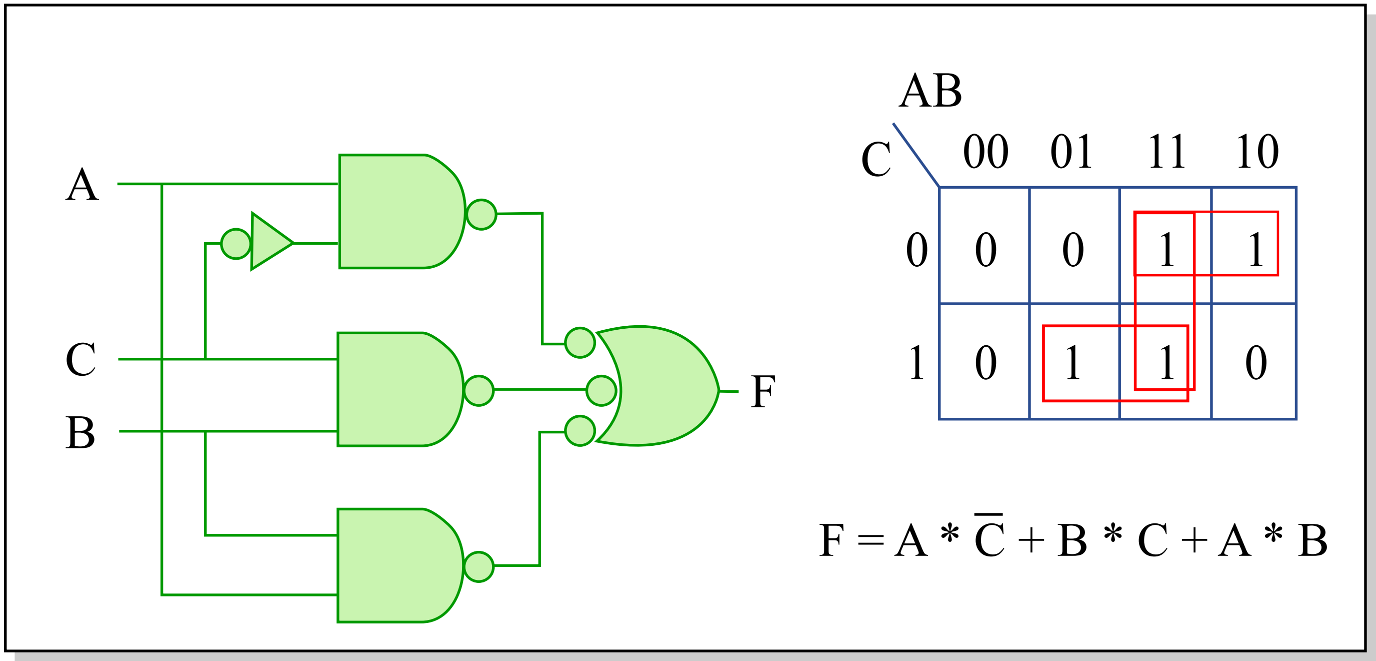


Figure by MIT OpenCourseWare.

Fixing Hazards

The glitch is the result of timing differences in parallel data paths. It is associated with the function jumping between groupings or product terms on the K-map. To fix it, cover it up with another grouping or product term!



C \ AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$F = A * \overline{C} + B * C + A * B$$

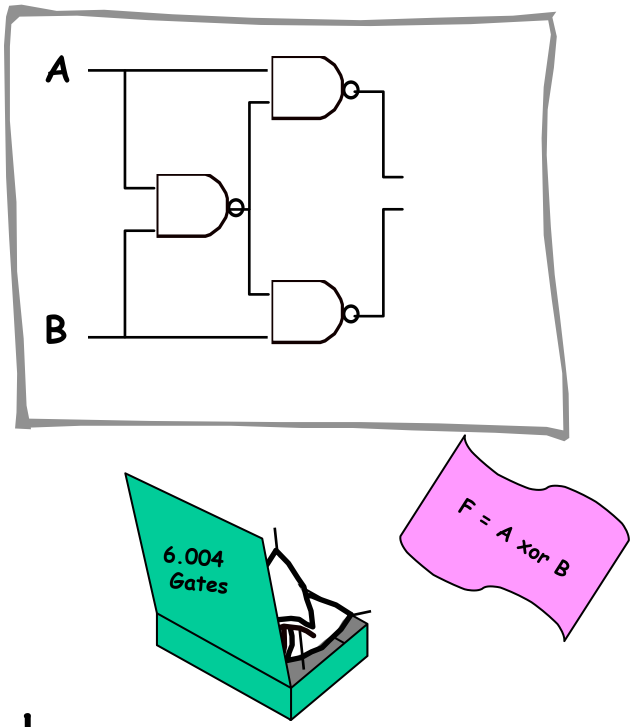
Figure by MIT OpenCourseWare.

- **In general, it is difficult to avoid hazards – need a robust design methodology to deal with hazards.**

Lets design stuff!

Where do we start?

We have a bag of gates.



We have a spec.

What do we do?

Did I mention we have gates?

We can build ANY Combinational Device… *can't we????*

We need

… a systematic approach for designing logic

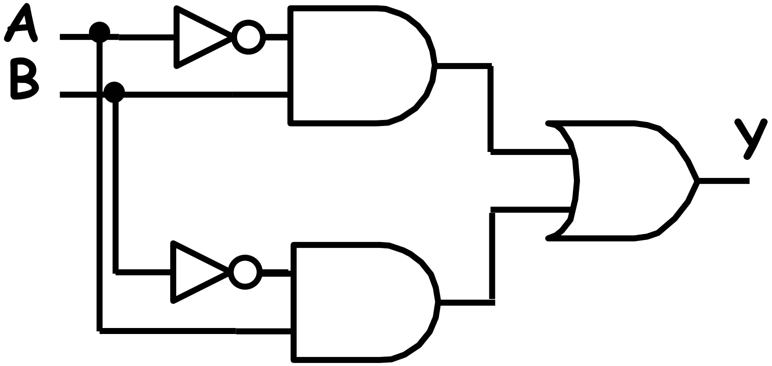
We can make most gates out of others

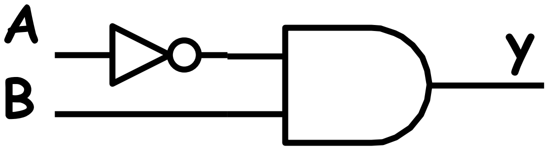
B>A

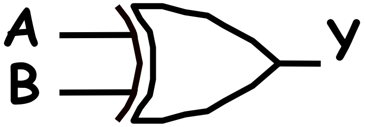
AB	Y
00	0
01	1
10	0
11	0

XOR

AB	Y
00	0
01	1
10	1
11	0



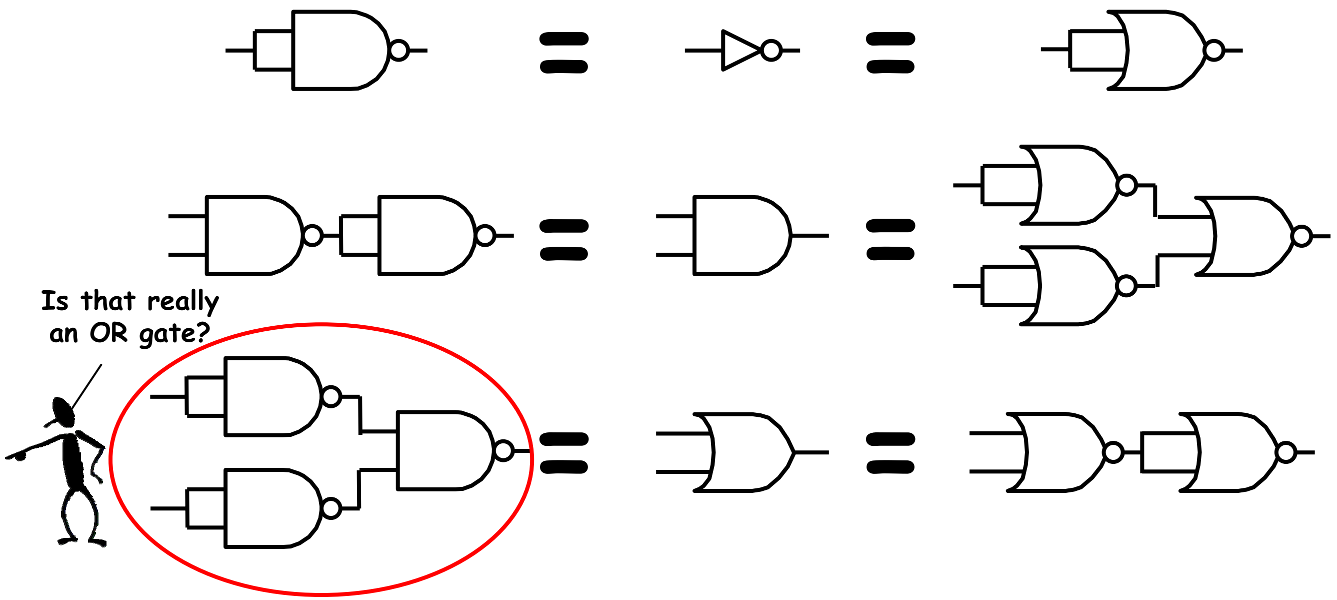




How many different *gates* do we *really* need?

One will do!

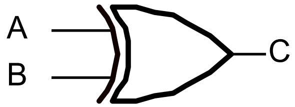
NANDs and NORs are universal



Ah!, but what if we want more than 2-inputs

Stupid Gate Tricks

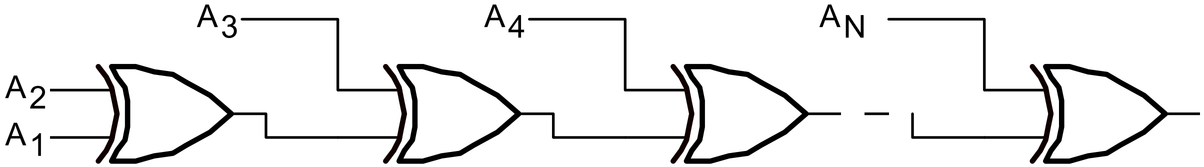
Suppose we have some 2-input XOR gates:



$t_{pd} = 1$
$t_{cd} = 0$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

And we want an N-input XOR:



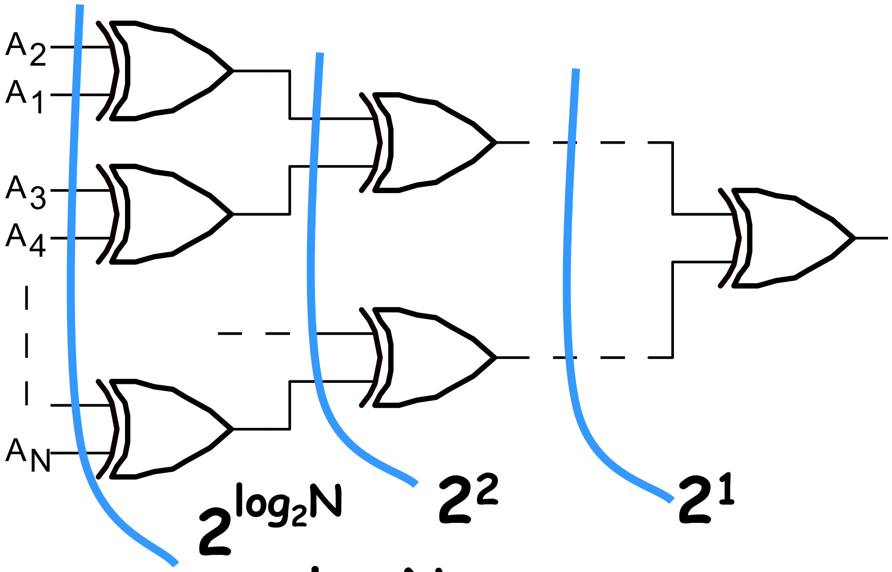
output = 1
iff number of 1s
input is ODD
("ODD PARITY")

t_{pd} = O(N) -- WORST CASE.

Can we compute N-input XOR faster?

I think that I shall never see

a circuit lovely as...



N-input TREE has O(log N) levels...

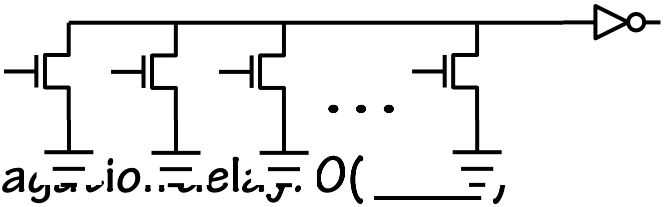
Signal propagation takes O(log N) gate delays.

Question: Can EVERY N-Input Boolean function be implemented as a tree of 2-input gates?

Are Trees Always Best?

Alternate Plan: Large Fan-in gates

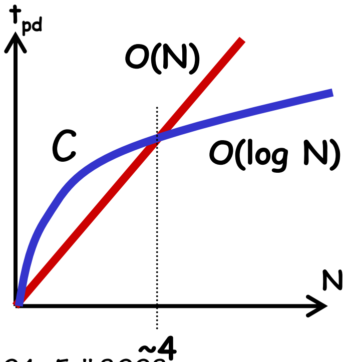
- N pulldowns with complementary pullups
- Output HIGH if any input is HIGH = "OR"



- Propagation delay: O(______)

  since each additional MOSFET adds

N



Don't be mislead by the "big O" stuff… the constants in this case can be much smaller… so for small N this plan might be the best.

Here's a Design Approach

Truth Table

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

1) Write out our functional spec as a truth table

2) Write down a Boolean expression for every '1' in the output

$$Y = \overline{C}\overline{B}A + \overline{C}BA + CB\overline{A} + CBA$$

3) Wire up the gates, call it a day, and declare success!

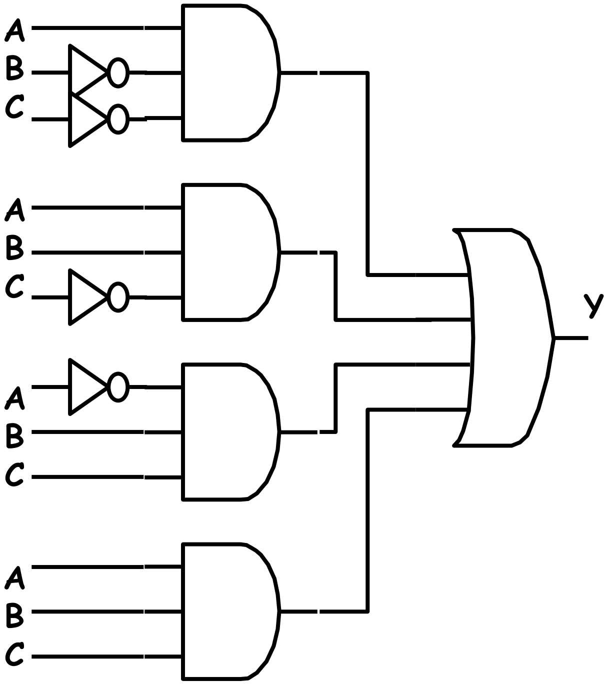
This approach will always give us Boolean expressions in a particular form: SUM-OF-PRODUCTS

Straightforward Synthesis

We can implement

SUM-OF-PRODUCTS

with just three levels of logic.



INVERTERS/AND/OR

Propagation delay --

No more than 3 gate delays

(ignoring fan-in)