FROM ZERO TO ONE

Part 2

***Digital Design and Computer Architecture*, 2nd Edition**

David Money Harris and Sarah L. Harris

ELSEVIER

Base Conversions

- From base-r >> decimal is easy
  - expand the number in power series and add all the terms
- Decimal >> base-r requires division
- Simple idea:
  - divide the decimal number successively by r
  - accumulate the remainders
- If there is a fraction, then integer part and fraction part are handled separately.

Base Conversion Examples 1/3

- Example 1:
  - 55
  - (decimal to binary)

55	1	1
27	1	2
13	1	4
6	0	
3	1	16
1=	1	32

- Example 2:
  - 144
  - (decimal to octal)

144	0	$0x8^0$
18	2	$2x8^1$
2=	2	$2x8^2$

Base Conversion Examples 2/3

- Example 1: 0.6875 (decimal to binary)
  - When dealing with fractions, instead of dividing by r, we multiply by r until we get an integer
  - $0.6875 \times 2 = 1.3750 = 1 + 0.375 \rightarrow a_{-1} = 1$
  - $0.3750 \times 2 = 0.7500 = 0 + 0.750 \rightarrow a_{-2} = 0$
  - $0.7500 \times 2 = 1.5000 = 1 + 0.500 \rightarrow a_{-3} = 1$
  - $0.5000 \times 2 = 1.0000 = 1 + 0.000 \rightarrow a_{-4} = 1$

  - $(0.6875)_{10} = (0.1011)_2$

Base Conversion Examples 3/3

- We are not always that lucky
- Example 2: (144.478) to octal
  - Treat the integer part and fraction part separately
  - $0.478 \times 8 = 3.824 = 3 + 0.824 \rightarrow a_{-1} = 3$
  - $0.824 \times 8 = 6.592 = 6 + 0.592 \rightarrow a_{-2} = 6$
  - $0.592 \times 8 = 4.736 = 4 + 0.736 \rightarrow a_{-3} = 4$
  - $0.736 \times 8 = 5.888 = 5 + 0.888 \rightarrow a_{-4} = 5$
  - $0.888 \times 8 = 7.104 = 7 + 0.104 \rightarrow a_{-5} = 7$
  - $0.104 \times 8 = 0.832 = 0 + 0.832 \rightarrow a_{-6} = 0$
  - $0.832 \times 8 = 6.656 = 6 + 0.656 \rightarrow a_{-7} = 6$
  - $144.478 = (220.3645706\ldots)_8$

Conversions between Binary, Octal and Hexadecimal

- r = 2 (binary), r = 8 (octal), r = 16 (hexadecimal)

```
1011000110100 1.101100010111

10 110 001 101 001.101 100 010 111

10 1100 0110 1001.1011 0001 0111
```

- Octal and hexadecimal representations are more compact.
- Therefore, we use them in order to communicate with computers directly using their internal representation

What if you can't make exact groups?

For example:

```
10110001101001.10110001011101

10 110 001 101 001.101 100 010 111 010

10 1100 0110 1001.1011 0001 0111 0100
```

Subtraction with 2's Complements

- Example:
  - X = 101 0100 (84) and Y = 100 0011 (67)
  - X-Y = ? and Y-X = ? in 8 bits

```
                    X      01010100   84
2's complement of   Y     +10111101   2's comp of 67
                          ---------
                          100010001   17
```

throw out the carry out

```
                    Y      01000011   67
2's complement of   X     +10101100   2's comp of 84
                          ---------
                          11101111   -17
```

2's complement of X-Y

Arithmetic with 2's complement

- Examples:

```
+11      00001011        -11      11110101
 +9    + 00001001         +9    + 00001001
```

```
+11      00001011        -11      11110101
 -9    + 11110111         -9    + 11110111
```

- No special treatment for sign bits

Arithmetic with 2's complement

- Examples:

```
+11      00001011        -11      11110101
 +9    + 00001001         +9    + 00001001
      ----------               ----------
      00010100                 11111110
```

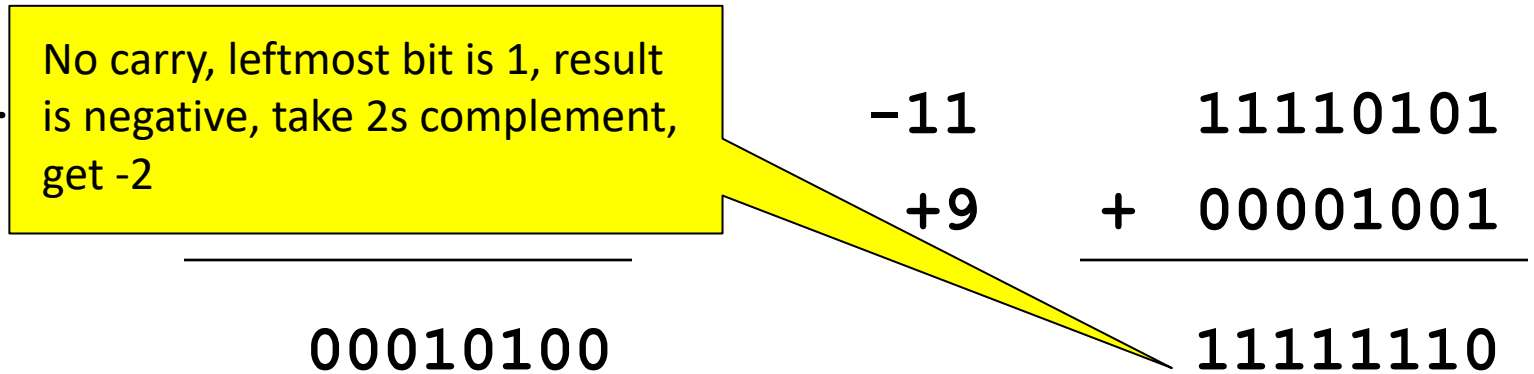
No carry, leftmost bit is 0, result is what you want

```
+11      00001011                      1
 -9    + 11110111         -9    + 11110111
      ----------               ----------
     100000010               111101100
```

- No special treatment for sign bits

Arithmetic with 2's complement

- Examples:




```
 +11      00001011      -11      11110101
  -9    + 11110111       -9    + 11110111
 ----    --------      ----    --------
        100000010              111101100

 +11      00001011      -11      11110101
  +9    + 00001001       +9    + 00001001
 ----    --------      ----    --------
        00010100               11111110
```

- No special treatment for sign bits

11

Arithmetic with 2's complement

- Examples:

```
+11      00001011        -11      11110101
 +9    + 00001001         +9    + 00001001
                                 ----------
                          -11      11111110
```

Carry=1, leftmost bit is 0, result is what you want

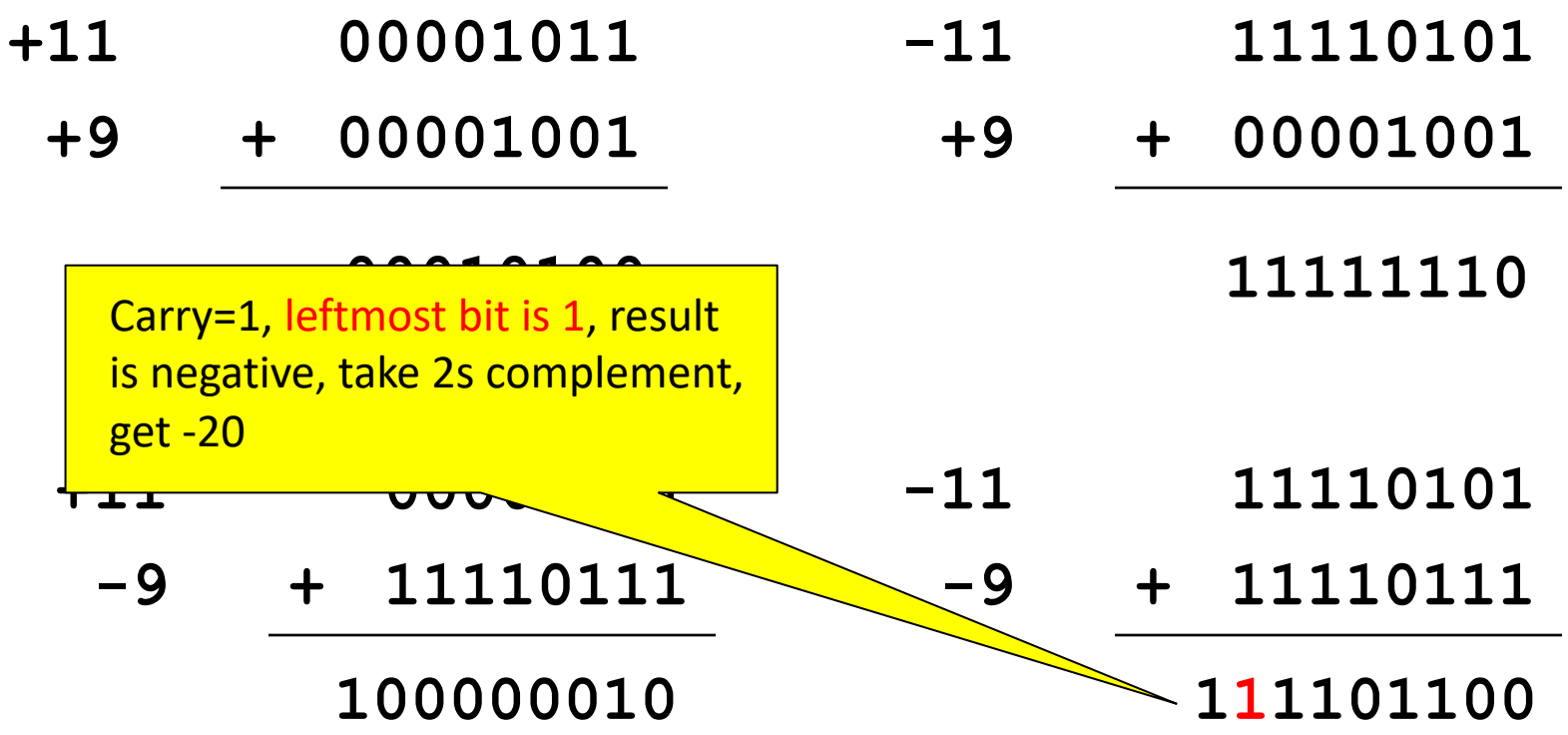
```
+11      00001011        -11      11110101
 -9    + 11110111         -9    + 11110111
                                 ----------
         100000010                111101100
```

- No special treatment for sign bits

12

Arithmetic with 2's complement

- Examples:



- No special treatment for sign bits

13

Arithmetic Overflow

- In hardware, we have limited resources to accommodate numbers
  - Computers use 8-bit, 16-bit, 32-bit, and 64-bit registers for the operands in arithmetic operations.
  - Sometimes the result of an arithmetic operation gets too large to fit in a register.

Detecting Overflow

- The rules for detecting overflow in a 2's complement sum are simple:
  - If the sum of two positive numbers yields a negative result, the sum is overflowed.
  - If the sum of two negative numbers yields a positive result, the sum is overflowed.
  - Otherwise, the sum is not overflowed.

Arithmetic Overflow (2's complement)

- Example:

```
+2    0010          -3    1101
+4  + 0100          -5  + 1011
------              ------
  0110                11000

+7    0111          -3    1101
+6  + 0110          -6  + 1010
------              ------
  1101                10111
```

Examples with 2's complement

-19 + (-7) = -26:

1	1	1	1	1			1	
	1	1	1	0	1	1	0	1
+	1	1	1	1	1	0	0	1
	1	1	1	0	0	1	1	0

Carryout without overflow. Sum is correct.

Examples with 2's complement

-75 + 59 = -16:

		1	1	1	1	1	1	
	1	0	1	1	0	1	0	1
+	0	0	1	1	1	0	1	1
	1	1	1	1	0	0	0	0

No overflow nor carryout.

Examples with 2's complement

-103 + -69 = -172:

$$
\begin{array}{cccccccccc}
\boxed{1} & & 1 & 1 & 1 & & 1 & 1 & & \\
 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \\
+ & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & \\
\hline
 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 &
\end{array}
$$

Overflow, with incidental carryout. Sum is not correct.

Examples

104 + 45 = 149:

	1	1		1				
	0	1	1	0	1	0	0	0
+	0	0	1	0	1	1	0	1
	1	0	0	1	0	1	0	1

Overflow, no carryout. Sum is not correct.

Examples with 2's complement

-39 + 92 = 53:

1	1		1	1				
	1	1	0	1	1	0	0	1
+	0	1	0	1	1	1	0	0
	0	0	1	1	0	1	0	1

Carryout without overflow. Sum is correct.

Examples

10 + -3 = 7:

	1	1	1	1				
	0	0	0	0	1	0	1	0
+	1	1	1	1	1	1	0	1
	0	0	0	0	0	1	1	1

Carryout without overflow. Sum is correct.

Examples

127 + 1 = 128:

	1	1	1	1	1	1	1	
	0	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1
	1	0	0	0	0	0	0	0

Overflow, no carryout. Sum is not correct.

Examples

-1 + 1 = 0:

	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0

Carryout without overflow. Sum is correct.

Alphanumeric Codes

- Besides numbers, we have to represent other types of information
  - letters of alphabet, mathematical symbols.
- For English, alphanumeric character set includes
  - 10 decimal digits
  - 26 letters of the English alphabet (both lowercase and uppercase)
  - several special characters
- We need an alphanumeric code
  - ASCII (American Standard Code for Information Interchange)
    - Uses 7 bits to encode 128 characters

ASCII Code

- 7 bits of ASCII Code
  - $(b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0)_2$
- Examples:
  - A → **65 = (1000001)**, …, Z → **90 = (1011010)**
  - a → **97 = (1100001)**, …, z → **122 = (1111010)**
  - **0 →48 = (0110000)**, …,**9 → 57 = (0111001)**
- 128 different characters
  - 26 + 26 + 10 = 62 (letters and decimal digits)
  - 32 special printable characters %, *, $
  - 34 special control characters (non-printable): BS, CR, etc.

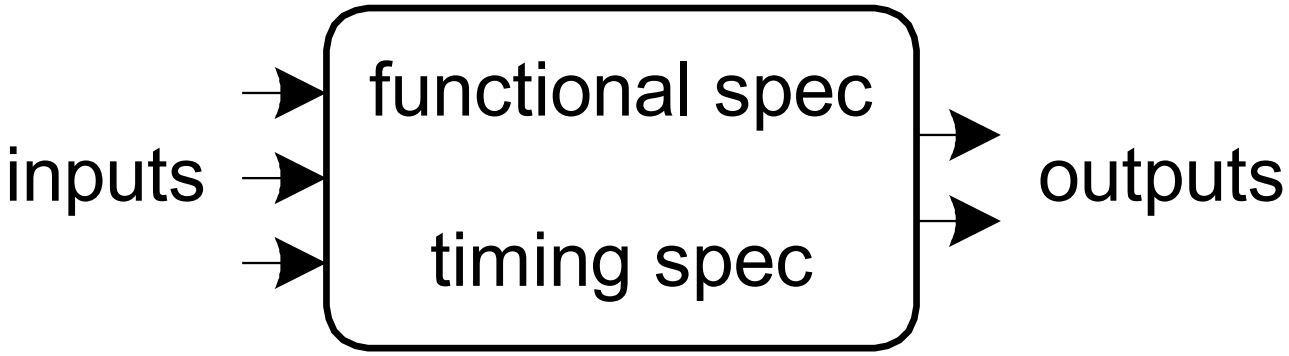
Representing ASCII Code

- 7-bit
- Most computers manipulate 8-bit quantity as a single unit (byte)
  - One ASCII character is stored using one byte
  - One unused bit can be used for other purposes such as representing Greek alphabet, italic type font, etc.
- The eighth bit can be used for error-detection
  - parity of seven bits of ASCII code is prefixed as a bit to the ASCII code.
  - even parity : A → (0 1000001)
    - 1 added to make number of 1s even, 0 if already even
  - odd parity : A → (1 1000001)
    - 1 added to make number of 1s odd, 0 if already odd
  - Detects one, three, and any odd number of bit errors

Logic Circuit

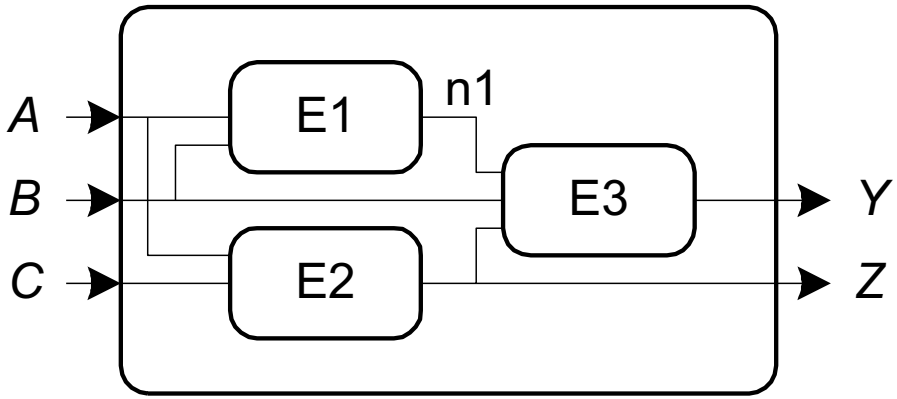
A logic circuit is composed of:

- Inputs
- Outputs
- Functional specification
- Timing specification



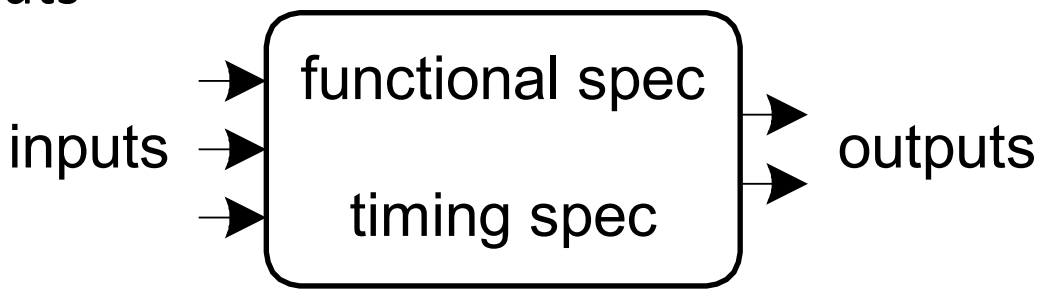
Circuits

- Nodes
  - Inputs: *A*, *B*, *C*
  - Outputs: *Y*, *Z*
  - Internal: n1
- Circuit elements
  - E1, E2, E3
  - Each a circuit



Types of Logic Circuits

- **Combinational Logic**
  - Memoryless
  - Outputs determined by current values of inputs
- **Sequential Logic**
  - Has memory
  - Outputs determined by previous and current values of inputs

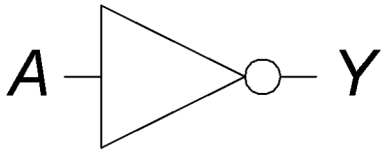


Logic Gates

- **Perform logic functions:**
  - inversion (NOT), AND, OR, NAND, NOR, etc.
- **Single-input:**
  - NOT gate, buffer
- **Two-input:**
  - AND, OR, XOR, NAND, NOR, XNOR
- **Multiple-input**

Single-Input Logic Gates

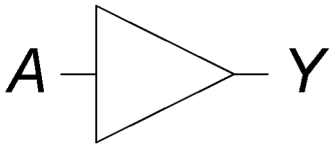
NOT



$Y = \overline{A}$

A	Y
0	
1	

BUF

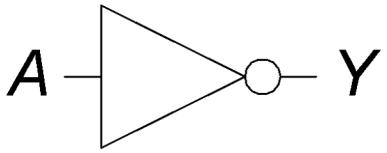


$Y = A$

A	Y
0	
1	

Single-Input Logic Gates

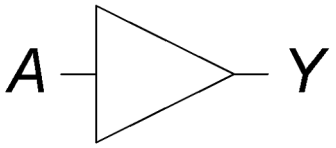
NOT



$Y = \overline{A}$

A	Y
0	1
1	0

BUF

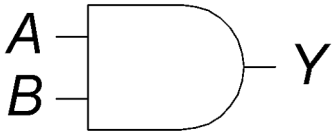


$Y = A$

A	Y
0	0
1	1

Two-Input Logic Gates

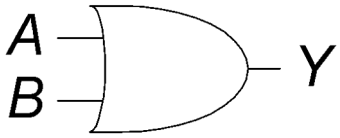
AND



$Y = AB$

A	B	Y
0	0	
0	1	
1	0	
1	1	

OR



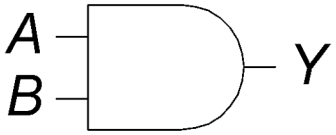
$Y = A + B$

A	B	Y
0	0	
0	1	
1	0	
1	1	

Suppose you want to ensure that you take your credit card with you when you go out. If you ensure your credit card is in your wallet **AND** your wallet is in your bag then taking your bag with you will suffice:
credit card in wallet & wallet in bag ⇒ credit card in bag

Two-Input Logic Gates

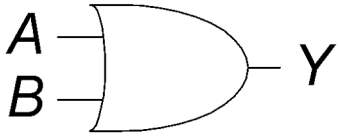
AND



$Y = AB$

A	B	Y
0	0	
0	1	
1	0	
1	1	

OR



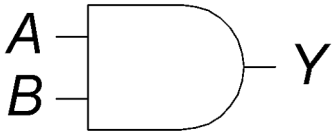
$Y = A + B$

A	B	Y
0	0	
0	1	
1	0	
1	1	

Suppose you want to ensure that you can come back to your house when you go out. If you ensure your main keys are in your bag **OR** your spare keys are in your car then having access to either your bag or your car will mean you have access to your house as well:
keys in bag || spare keys in car ⇒ access to house

Two-Input Logic Gates

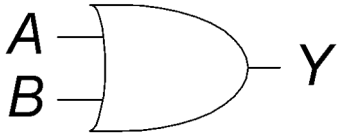
AND



$Y = AB$

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

OR



$Y = A + B$

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

Positive & Negative Logic

- In digital circuits, we have two digital signal levels:
  - H (higher signal level; e.g. 3 ~ 5 V)
  - L (lower signal level; e.g. 0 ~ 1 V)
- There is no logic-1 or logic-0 at the circuit level
- We can do any assignment we wish
  - For example:
    - H → logic-1
    - L → logic-0

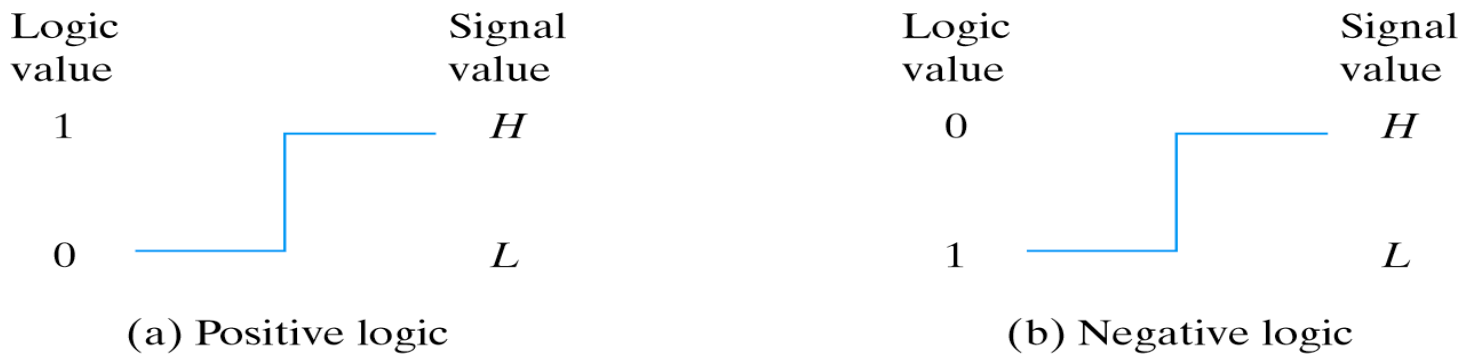


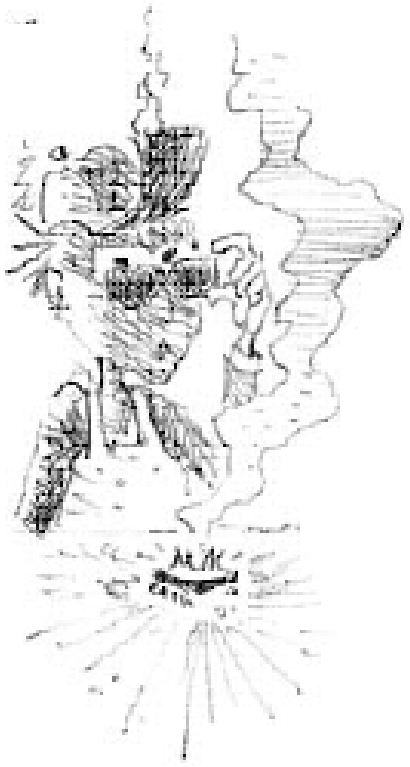
Fig. 2-9 signal assignment and logic polarity

Logic Levels

- Discrete voltages represent 1 and 0
- For example:
  - 0 = *ground* (GND) or 0 volts
  - 1 = V_{DD} or 5 volts
- What about 4.99 volts? Is that a 0 or a 1?
- What about 3.2 volts?

V_{DD} Scaling

- In 1970's and 1980's, V_{DD} = 5 V
- V_{DD} has dropped as chips got smaller to
  - avoid frying tiny transistors
  - save power
- 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V, 1.0 V, …
- Be careful connecting chips with different supply voltages



Chips operate because they contain magic smoke

Proof:

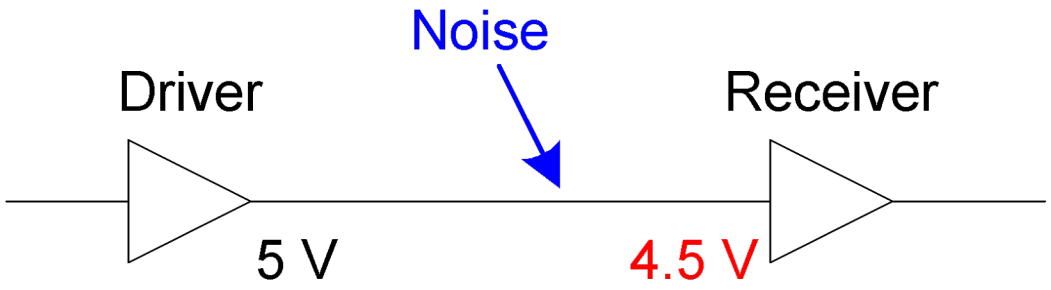
- if the magic smoke is let out, the chip stops working

Logic Levels

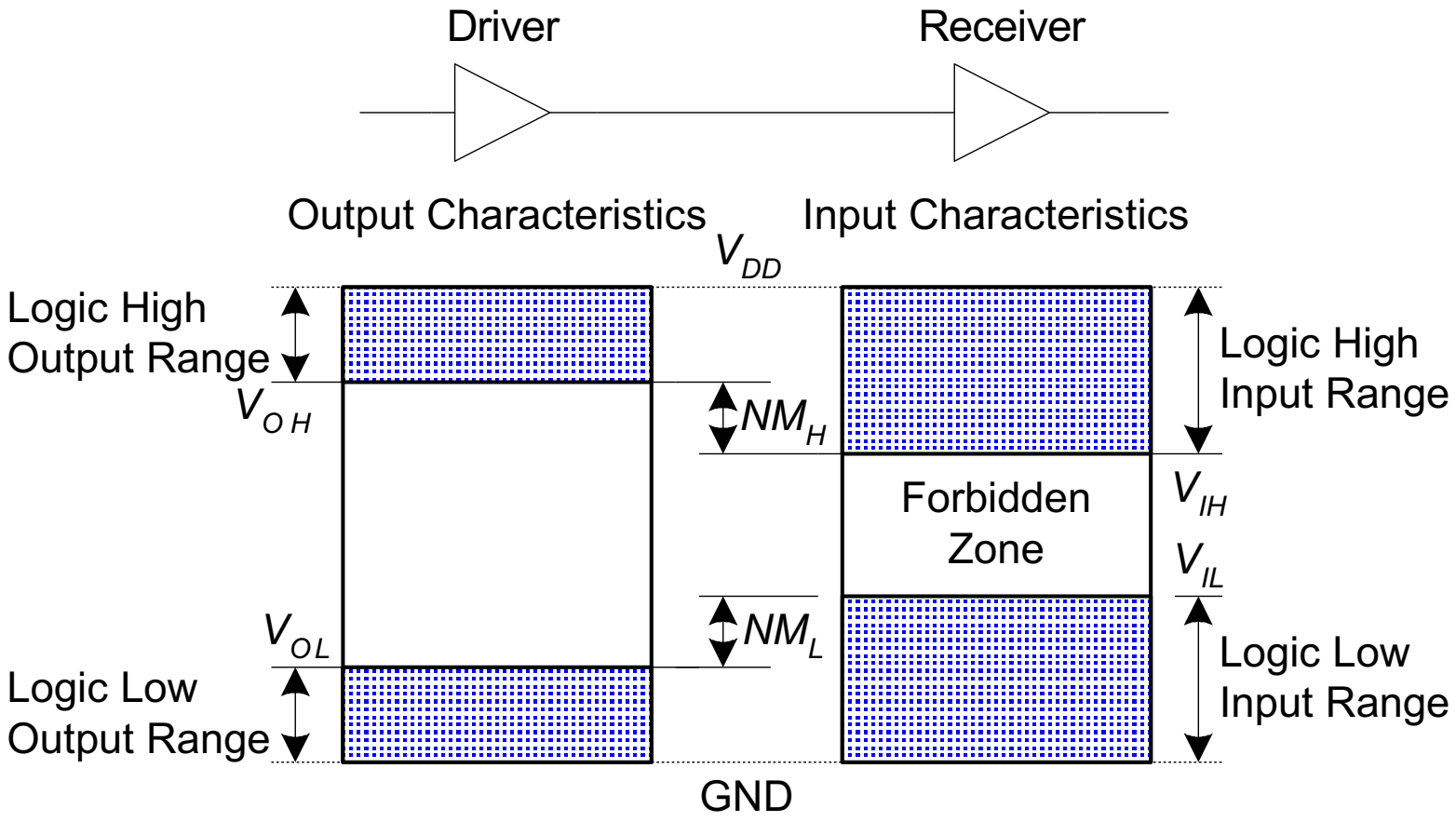
- *Range* of voltages for 1 and 0
- Different ranges for inputs and outputs to allow for *noise*

What is Noise?

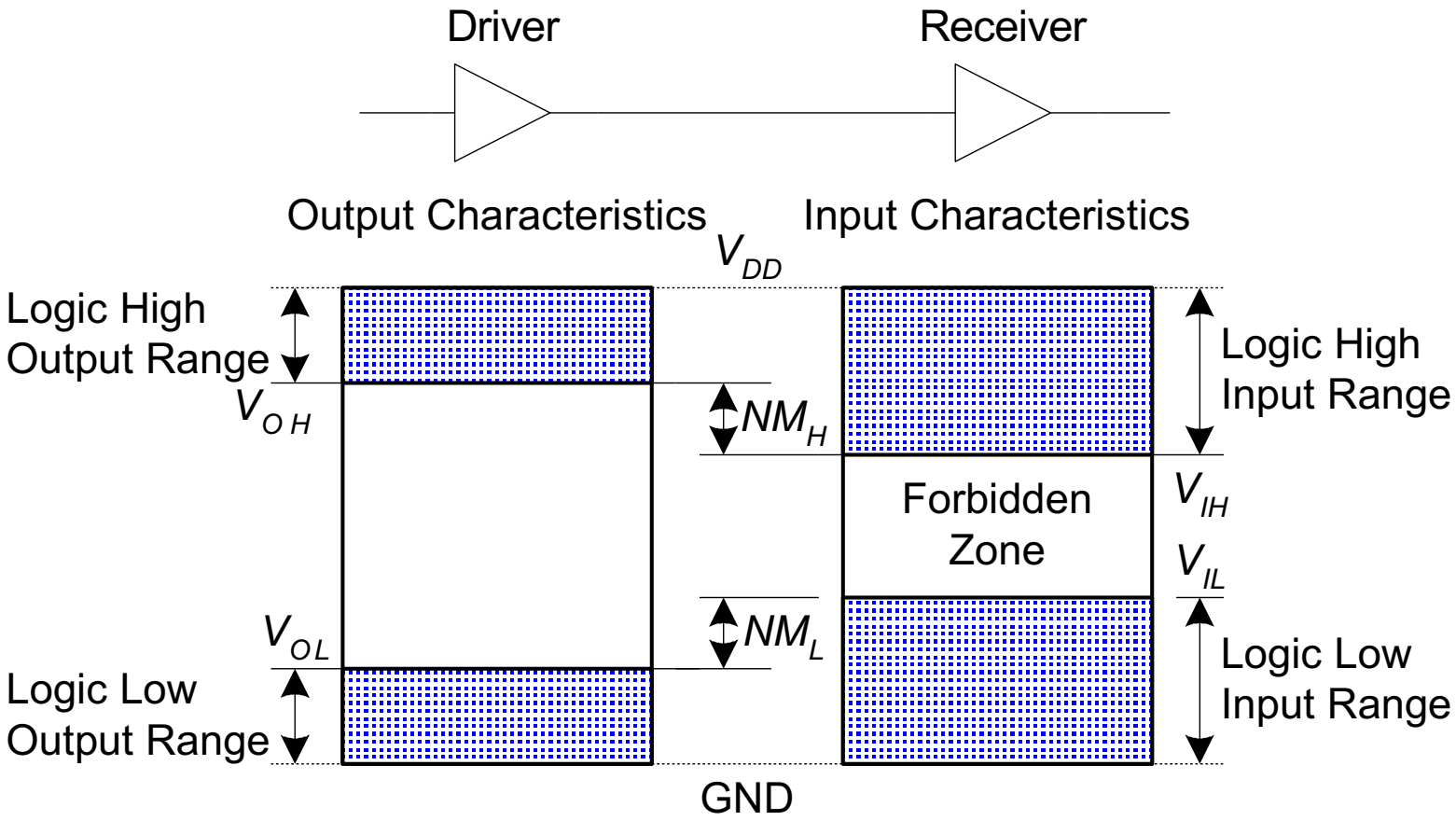
- **Anything that degrades the signal**
  - e.g., resistance, power supply noise, coupling to neighboring wires, etc.
- **Example:** a gate (driver) outputs 5 V but, because of resistance in a long wire, receiver gets 4.5 V



Logic Levels



Noise Margins

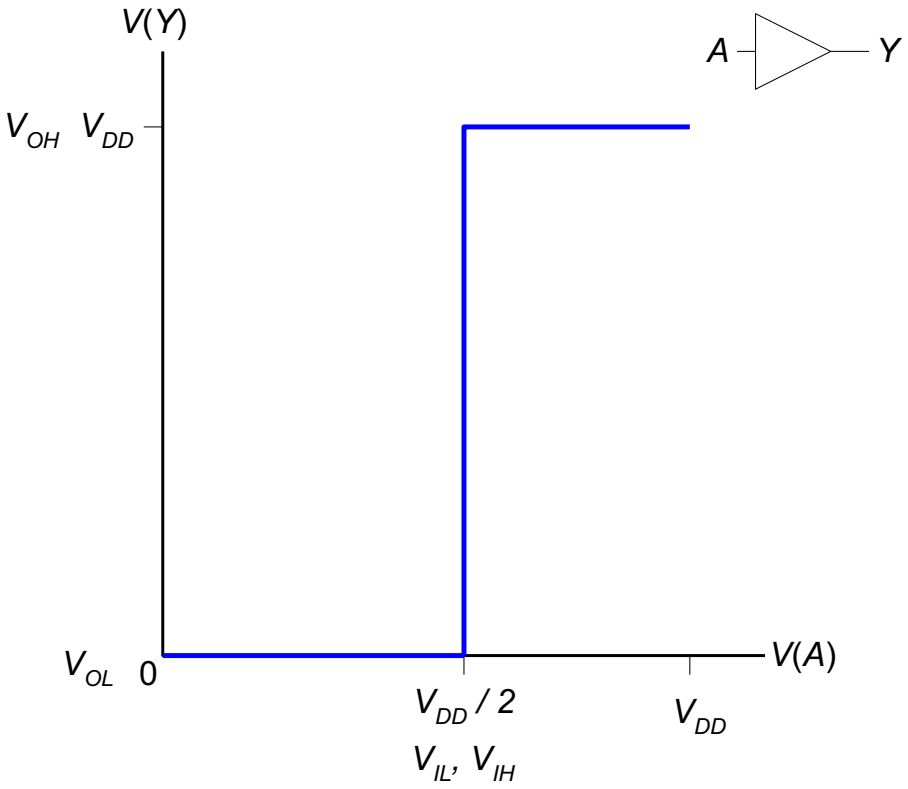


$$NM_H = V_{OH} - V_{IH}$$

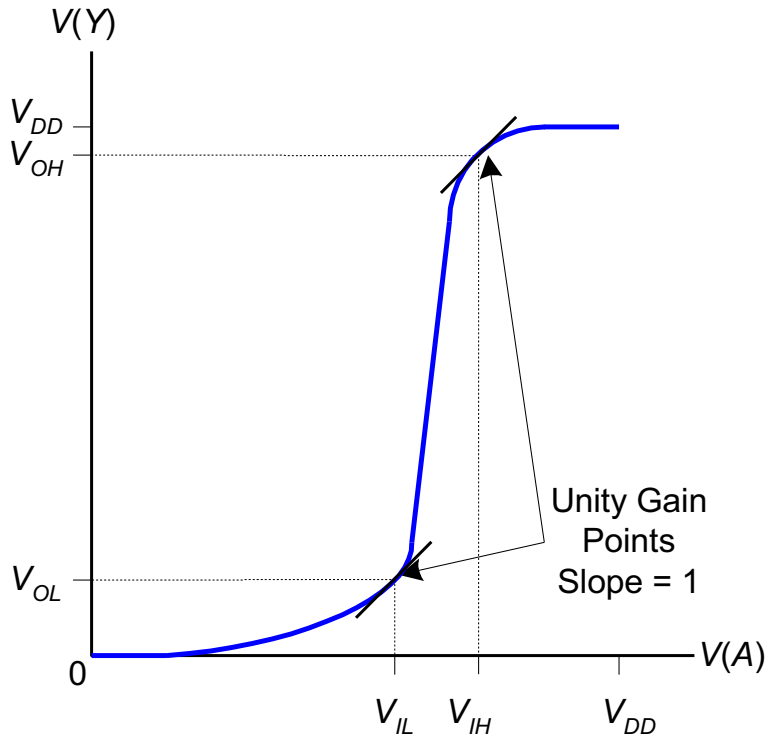
$$NM_L = V_{IL} - V_{OL}$$

DC Transfer Characteristics

Ideal Buffer:



Real Buffer:



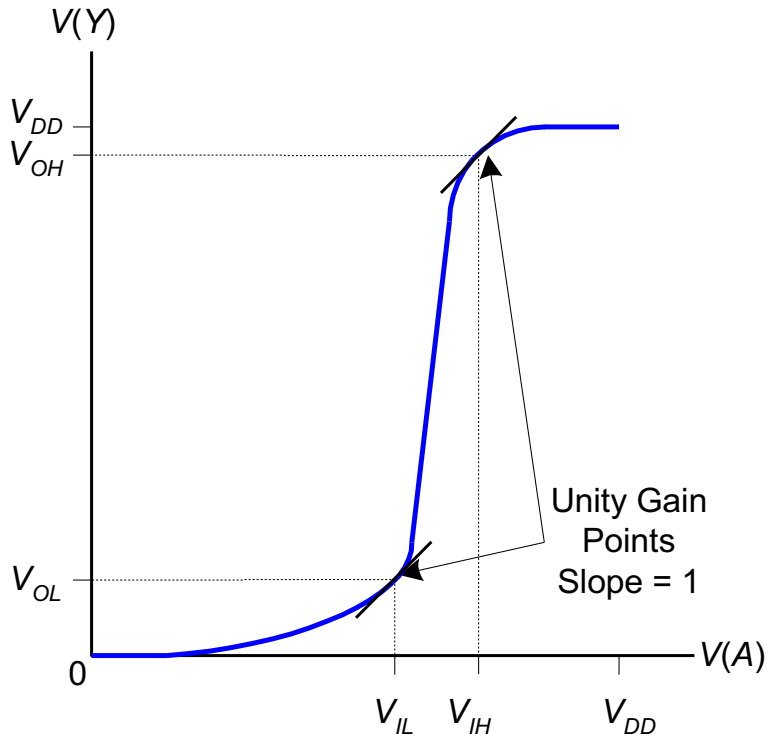
$NM_H = NM_L = V_{DD}/2$

$NM_H , NM_L < V_{DD}/2$

DC Transfer Characteristics

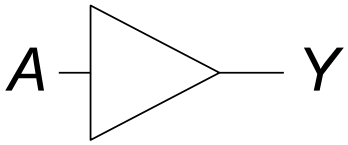
A reasonable place to choose the logic levels is where the slope of the transfer characteristic dV(Y) / dV(A) is −1. These two points are called the unity gain points. Choosing logic levels at the unity gain points usually maximizes the noise margins
>> If V_{IL} were reduced, V_{OH} would only increase by a small amount. But if V_{IL} were increased, V_{OH} would drop precipitously.

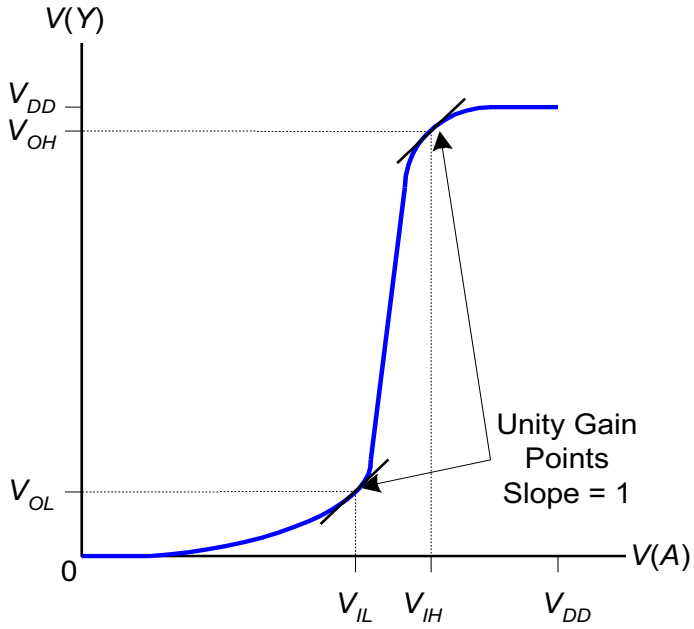
Real Buffer:

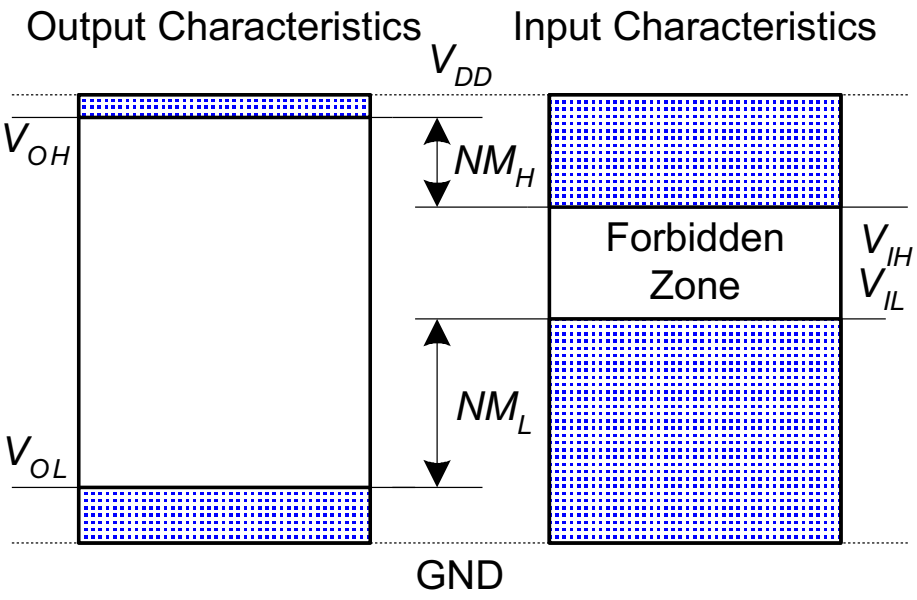


$$NM_H, NM_L < V_{DD}/2$$

DC Transfer Characteristics







The Static Discipline

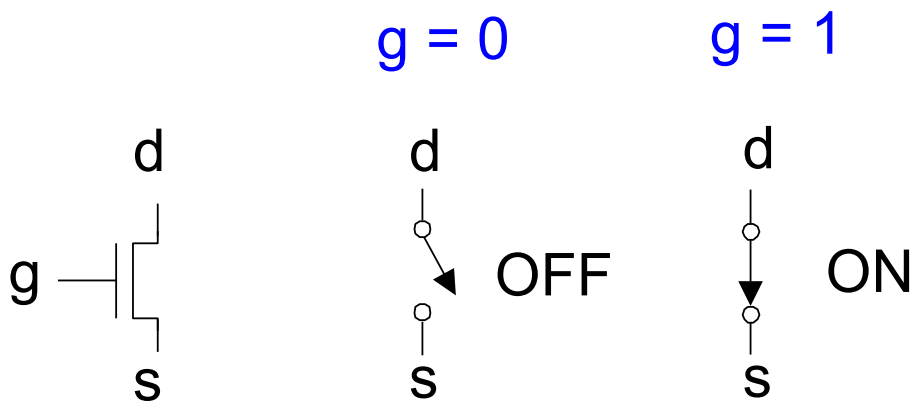
- To avoid inputs falling into the forbidden zone, digital logic gates are designed to conform to the *static discipline*.
- The static discipline requires that:
  - With logically valid inputs, every circuit element must produce logically valid outputs
  - Use limited ranges of voltages to represent discrete values

Transistors

- Babbage's Analytical Engine was built from gears, and early electrical computers used relays or vacuum tubes.
- Modern computers use transistors because they are cheap, small, and reliable.
- Transistors are electrically controlled switches that turn ON or OFF when a voltage or current is applied to a control terminal.
- Logic gates built from transistors

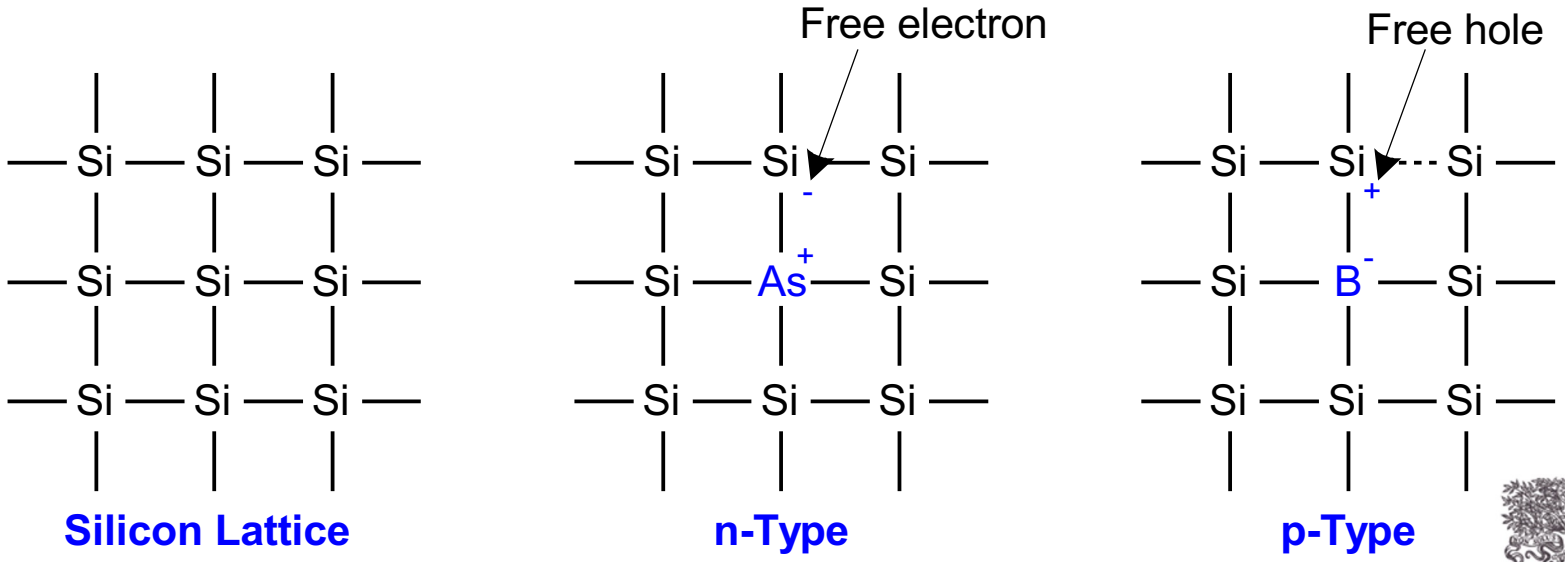
Transistors

- 3-ported voltage-controlled switch
  - 2 ports connected depending on voltage of 3rd
  - d and s are connected (ON) when g is 1



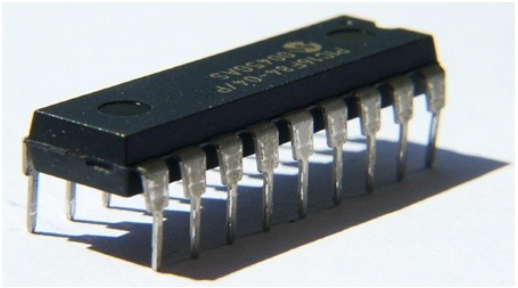
Silicon

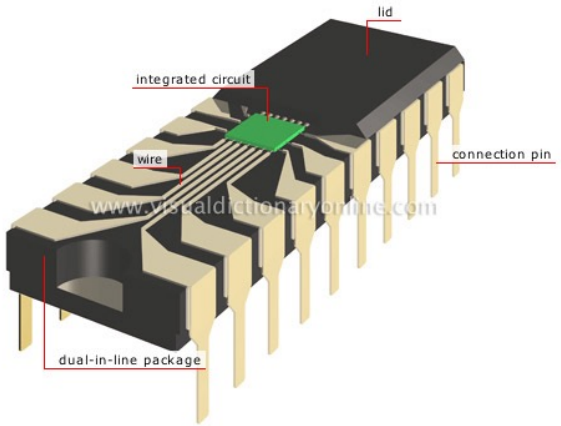
- Transistors built from silicon, a semiconductor
- Pure silicon is a poor conductor (no free charges)
- Doped silicon is a good conductor (free charges)
  - n-type (free *n*egative charges, electrons)
  - p-type (free *p*ositive charges, holes)

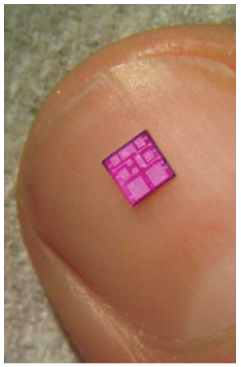


Integrated Circuits

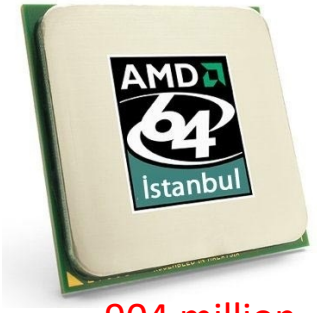
- IC – silicon semiconductor crystal (“chip”) that contains gates.
  - gates are interconnected inside to implement a Boolean function
  - Chip is mounted in a ceramic or plastic container
  - Inputs & outputs are connected to the external pins of the IC.
  - Many external pins (14 to hundreds)







Levels of Integration



904 million transistors

- SSI (small-scale integration):
  - inputs and outputs of the gates are connected directly to the pins in the package.
  - The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.
- MSI (medium-scale integration):
  - From 10 to 1,000 gates per chip
  - usually perform specific elementary digital operations.
  - Usual MSI digital functions: decoders, adders, multiplexers, registers and counters.
- LSI (large-scale integration):
  - 1,000s of gates per chip
  - include digital systems such as processors, memory chips, and programmable logic devices.
- VLSI (very large-scale integration) and ULSI (ultra large-scale integration):
  - devices now contain millions of gates within a single package.
  - Examples are large memory arrays and complex microcomputer chips.
  - Because of their small size and low cost, VLSI devices have revolutionized the computer system design technology, giving the designer the capability to create structures that were previously uneconomical to build.

Digital Logic Families

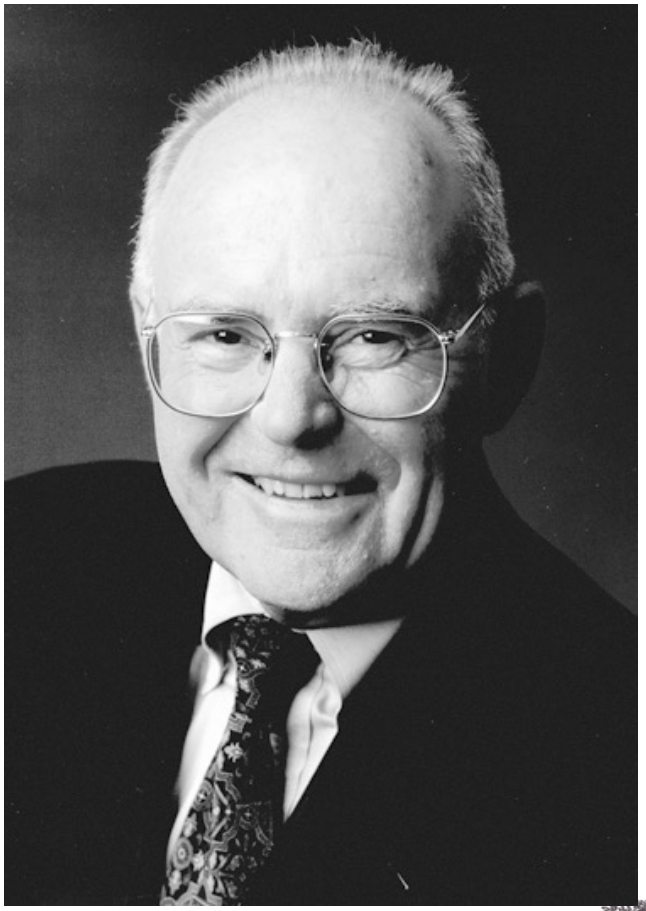
- Circuit Technologies
  - TTL → transistor-transistor logic
    - has been in use for 50 years and is considered to be standard.
  - ECL → Emitter-coupled logic
    - Fast. Has an advantage in systems requiring **high-speed operation**.
  - MOS → metal-oxide semiconductor
    - suitable for circuits that need **high component density**,
  - CMOS → Complementary MOS
    - preferable in systems requiring **low power consumption**, such as digital cameras, personal media players, and other handheld portable devices.
  - Low power consumption is essential for VLSI design; therefore, CMOS has become the dominant logic family, while TTL and ECL continue to decline in use.

Logic Family Examples

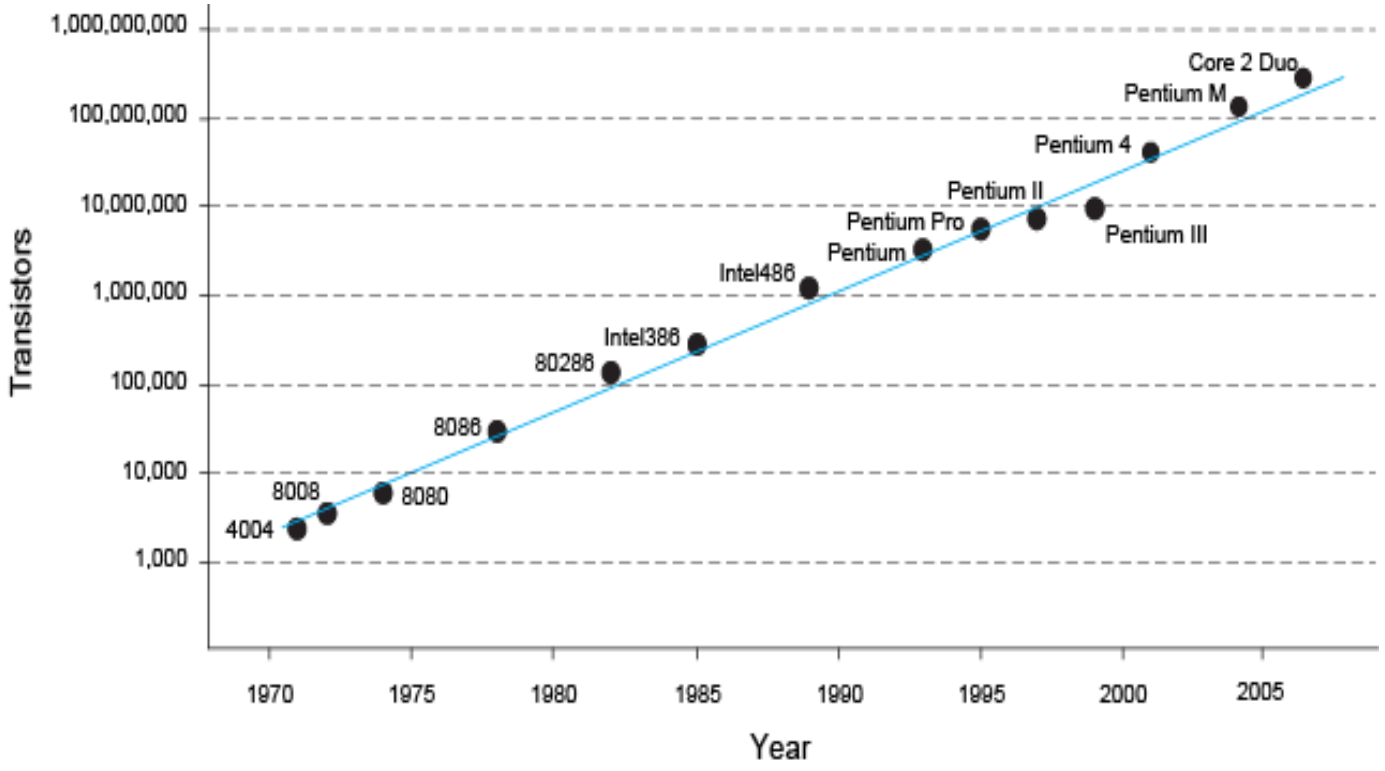
Logic Family	V_{DD}	V_{IL}	V_{IH}	V_{OL}	V_{OH}
TTL	5 (4.75 - 5.25)	0.8	2.0	0.4	2.4
CMOS	5 (4.5 - 6)	1.35	3.15	0.33	3.84
LVTTL	3.3 (3 - 3.6)	0.8	2.0	0.4	2.4
LVCMOS	3.3 (3 - 3.6)	0.9	1.8	0.36	2.7

Gordon Moore, 1929-

- Cofounded Intel in 1968 with Robert Noyce.
- **Moore's Law:** number of transistors on a computer chip doubles every year (observed in 1965)
- Since 1975, transistor counts have doubled every two years.



Moore's Law



- *"If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get one million miles to the gallon, and explode once a year . . ."*

  – Robert Cringley

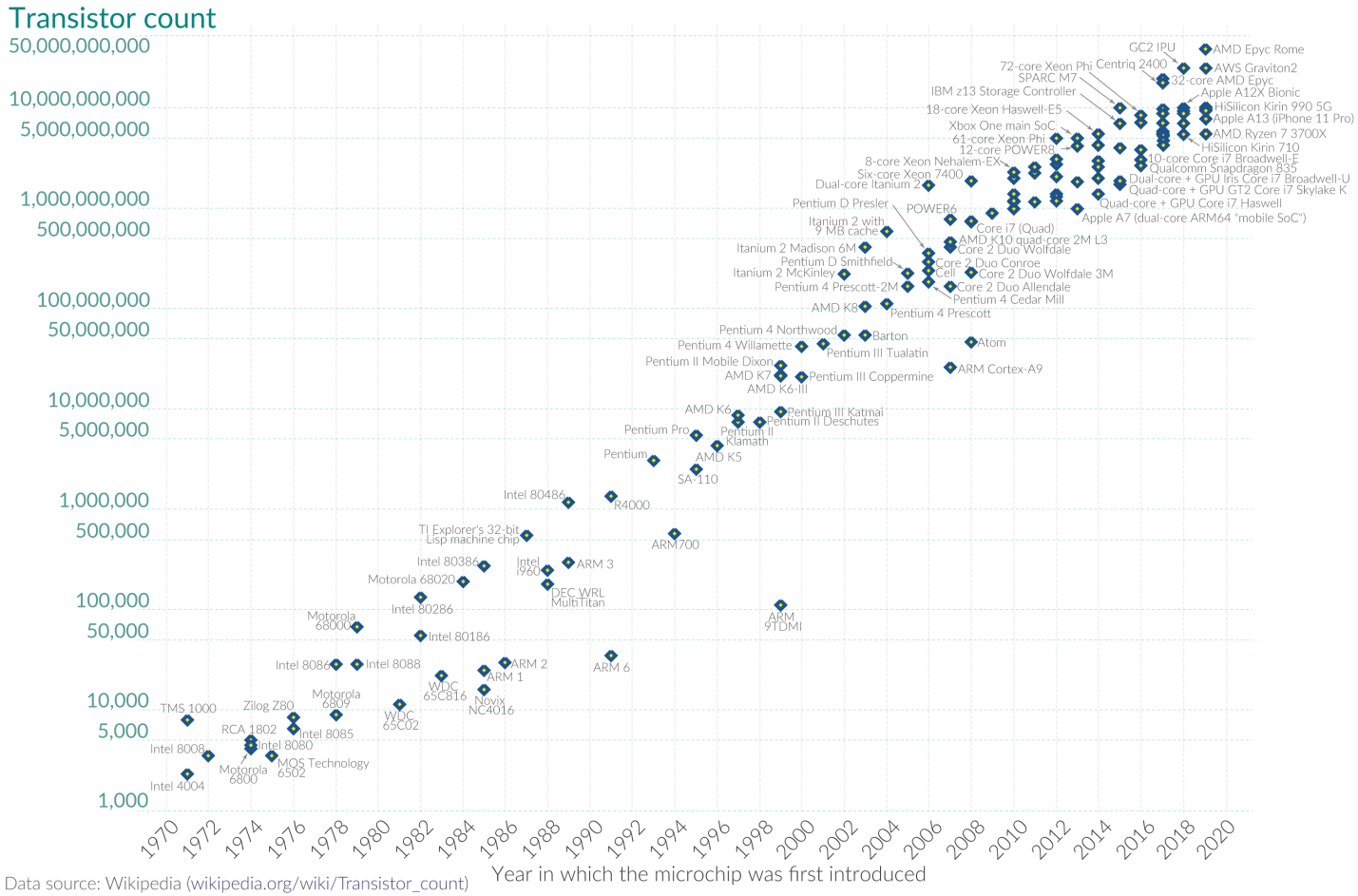
FROM ZERO TO ONE

Moore's Law

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

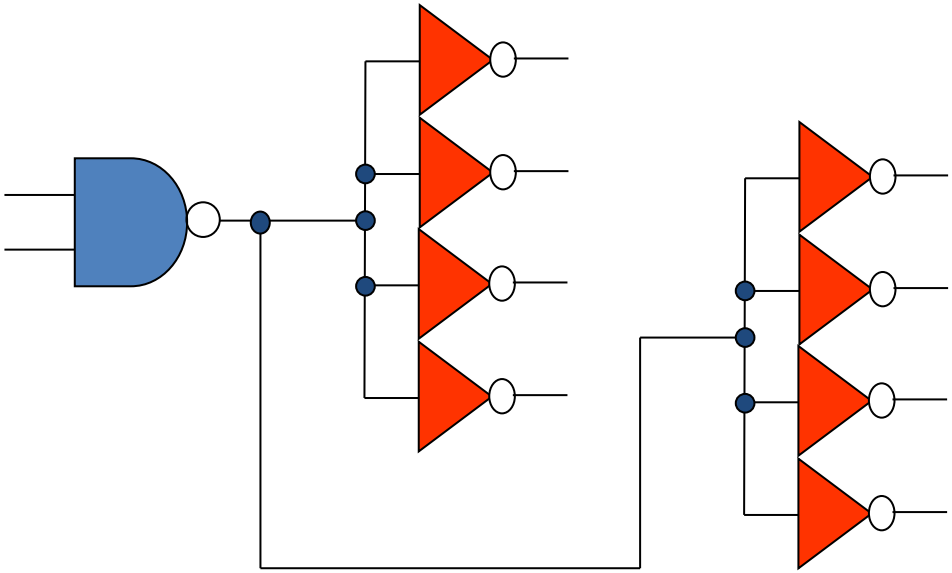


Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

ELSEVIER

Parameters of Logic Gates - 1

- **Fan-out**
  - Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation.
  - A **standard load** is usually defined as the amount of current needed by an input of another similar gate in the same family.

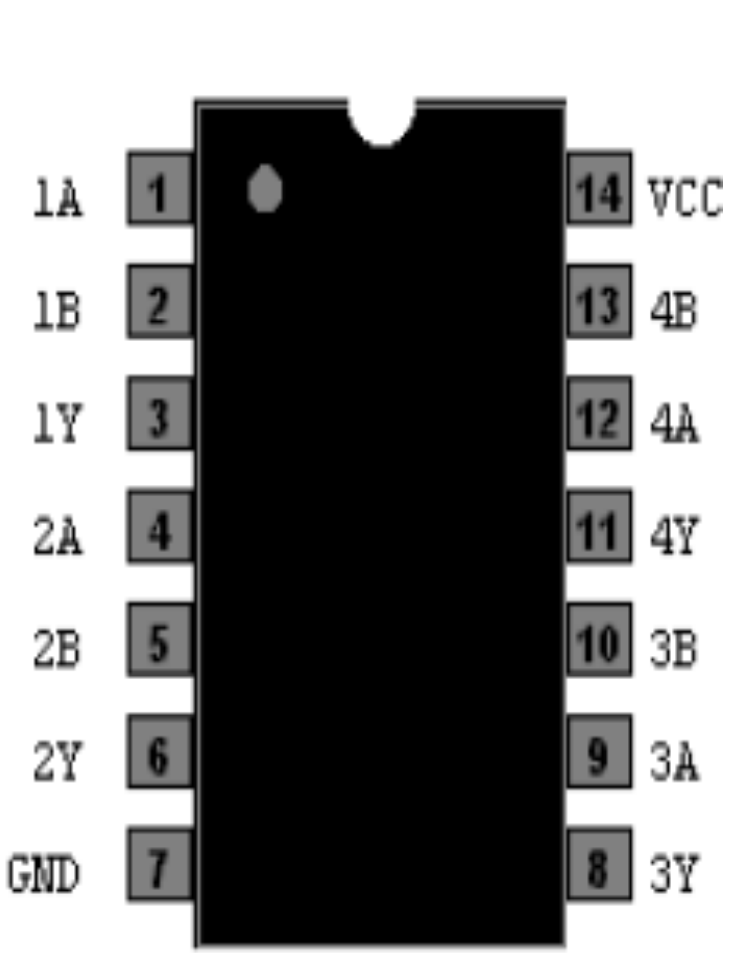


- For example, if a NAND gate drives 4 such inverters, then the fan-out is equal to 4.0 standard loads.

Parameters of Logic Gates - 2

- **Fan-in**
  - number of inputs that a gate can have in a particular logic family
  - In theory, we can design a CMOS NAND or NOR gate with a very large number of inputs
  - In practice, however, we have some limits
    - 4 for NOR gates
    - 6 for NAND gates
- **Power dissipation**
  - power consumed by the gate that must be available from the power supply

Power Dissipation



Difference in voltage level !

Power Consumption

- Power = Energy consumed per unit time
  - Dynamic power consumption
  - Static power consumption

Dynamic Power Consumption

- **Power to charge transistor gate capacitances**
  - Energy required to charge a capacitance, C, to V_{DD} is CV_{DD}^2
  - Circuit running at frequency f: transistors switch (from 1 to 0 or vice versa) at that frequency
  - Capacitor is charged $f/2$ times per second (discharging from 1 to 0 is free)
- Dynamic power consumption:

$$P_{dynamic} = \tfrac{1}{2}CV_{DD}^2 f$$

Static Power Consumption

- Power consumed when no gates are switching
- Caused by the *quiescent ("at rest") supply current*, I_{DD} (also called the *leakage current*)
- Static power consumption:

$$P_{static} = I_{DD}V_{DD}$$

Power Consumption Example

- Estimate the power consumption of a wireless handheld computer
  - V_{DD} = 1.2 V
  - C = 20 nF
  - f = 1 GHz
  - I_{DD} = 20 mA

Power Consumption Example

- Estimate the power consumption of a wireless handheld computer
  - V_{DD} = 1.2 V
  - C = 20 nF
  - f = 1 GHz
  - I_{DD} = 20 mA

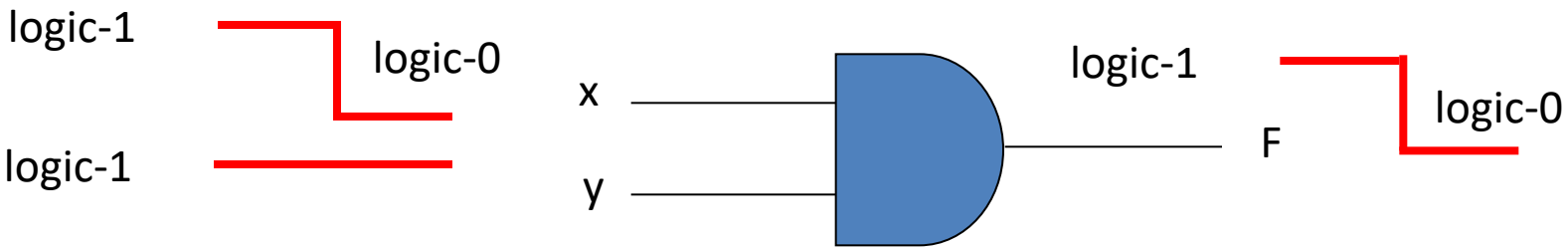
$$P = \tfrac{1}{2}CV_{DD}^2 f + I_{DD}V_{DD}$$

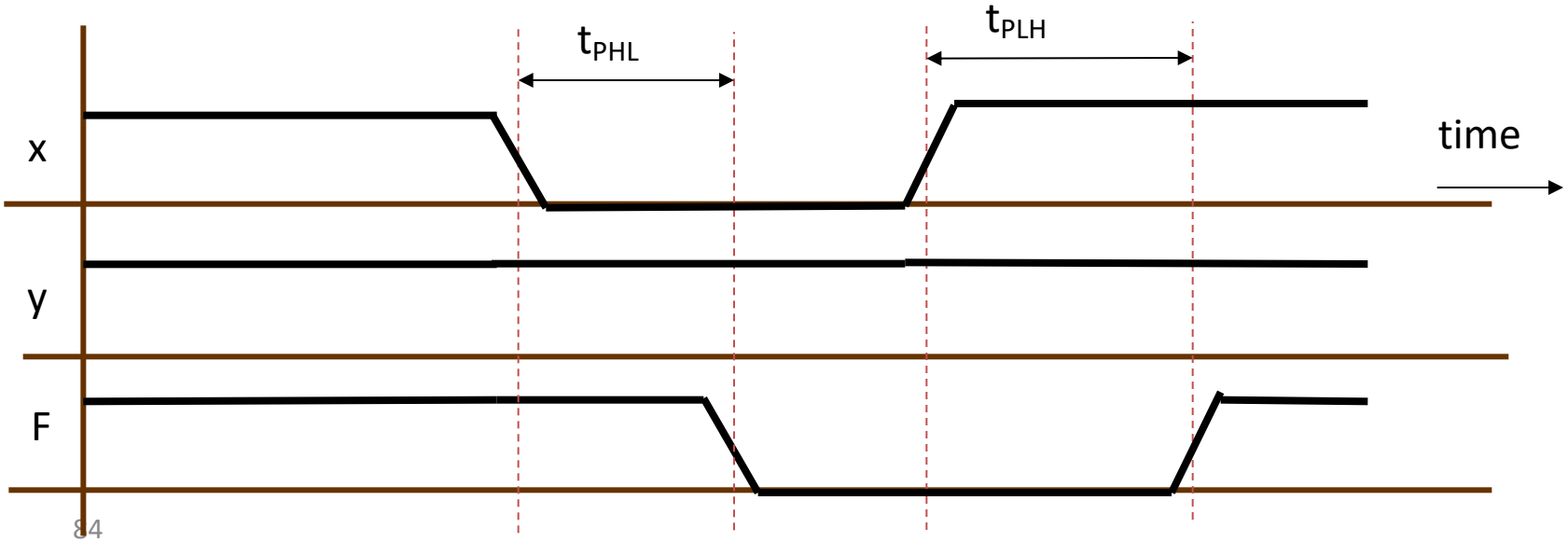
$$= \tfrac{1}{2}(20\text{ nF})(1.2\text{ V})^2(1\text{ GHz}) + (20\text{ mA})(1.2\text{ V})$$

$$= \mathbf{14.4\text{ W}}$$

Parameters of Logic Gates - 3

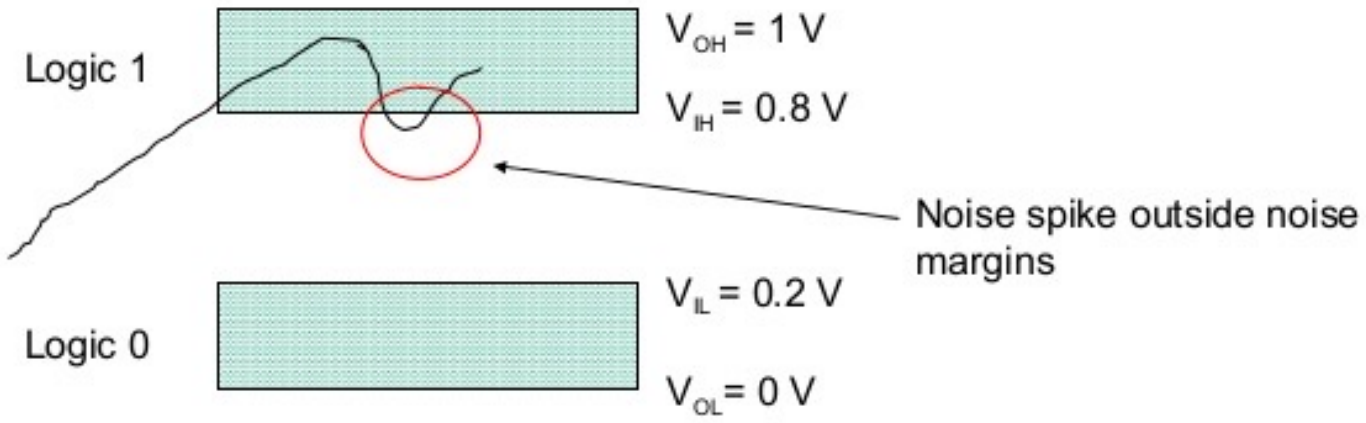
- **Propagation delay:**
  - the time required for a change in value of a signal to propagate from input to output.





Parameters of Logic Gates - 4

- ***Noise margin***
  - the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.



Noise tolerance is the degree to which a gate is impervious to noise spikes at its input.