Gates and Logic: From switches to Transistors, Logic Gates and Logic Circuits

Hakim Weatherspoon

CS 3410, Spring 2013

Computer Science

Cornell University

See: P&H Appendix C.2 and C.3 (Also, see C.0 and C.1)

iClicker

Lab0 was

a) Too easy

b) Too hard

c) Just right

d) Have not done lab yet

Goals for Today

From Switches to Logic Gates to Logic Circuits

Logic Gates

- From switches
- Truth Tables

Logic Circuits

- Identity Laws
- From Truth Tables to Circuits (Sum of Products)

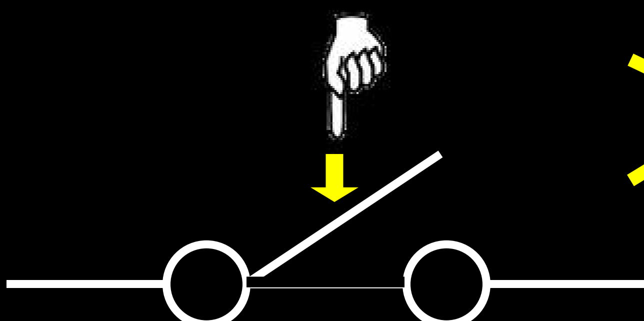
Logic Circuit Minimization

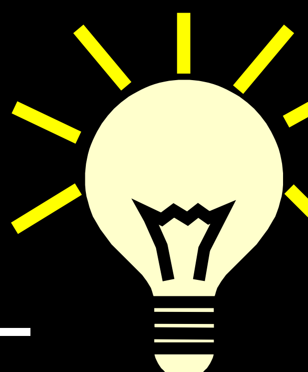
- Algebraic Manipulations
- Truth Tables (Karnaugh Maps)

Transistors (electronic switch)

A switch

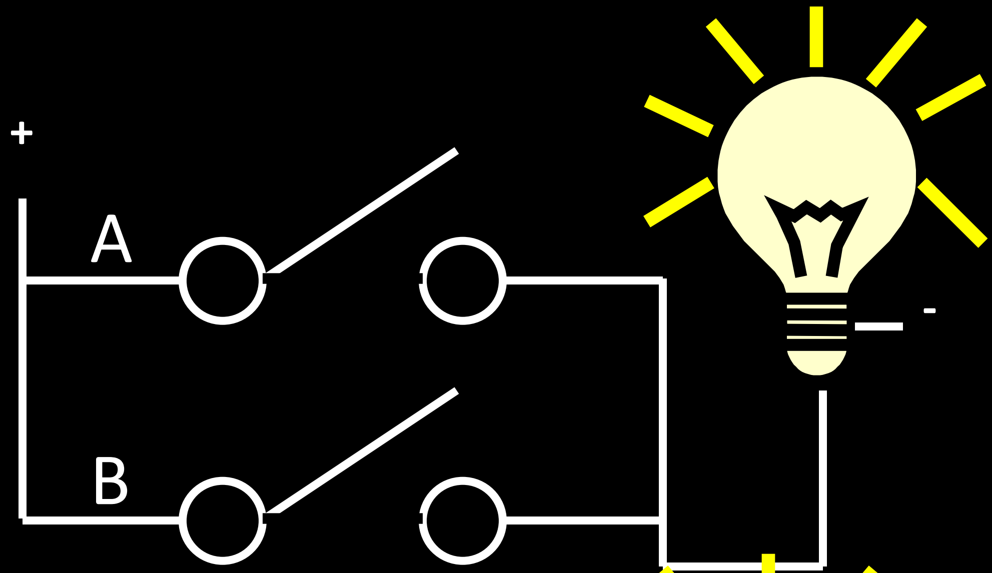
- Acts as a *conductor* or *insulator*
- Can be used to build amazing things…





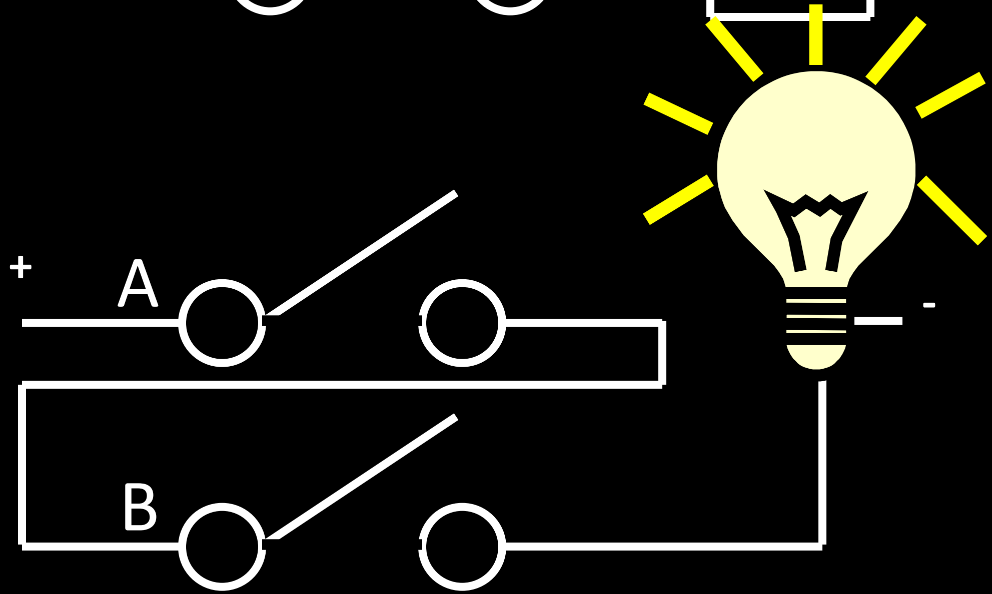
The Bombe used to break the German Enigma machine during World War II

Basic Building Blocks: Switches to Logic Gates



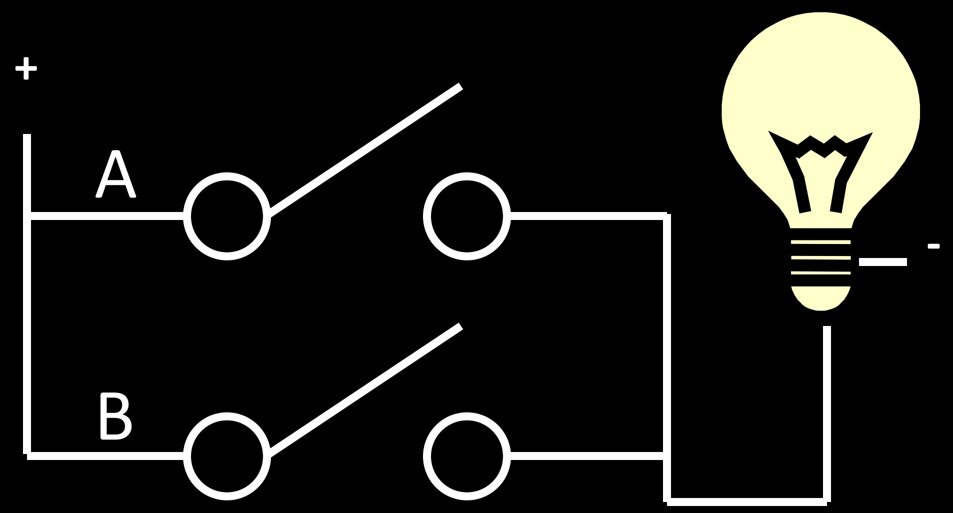
Truth Table

A	B	Light
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	



A	B	Light
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	

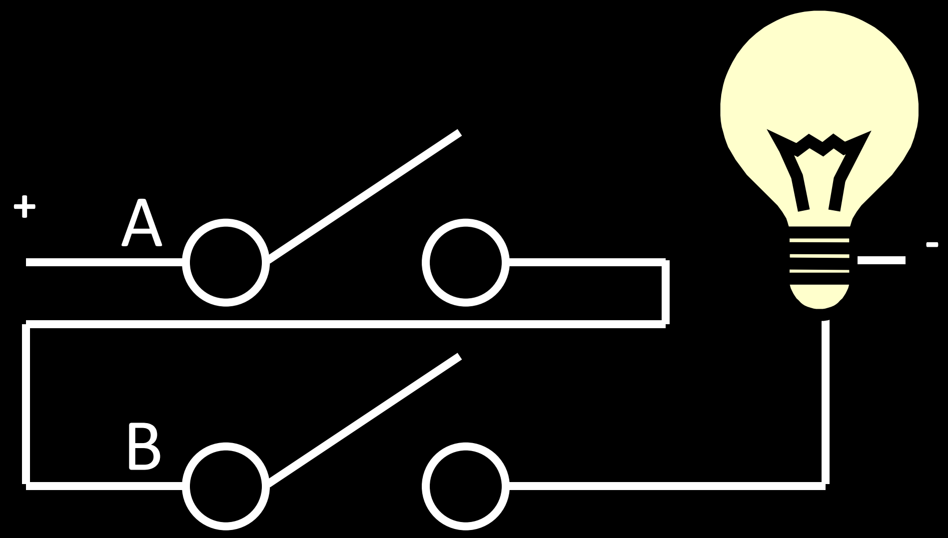
Basic Building Blocks: Switches to Logic Gates



Either (OR)

Truth Table

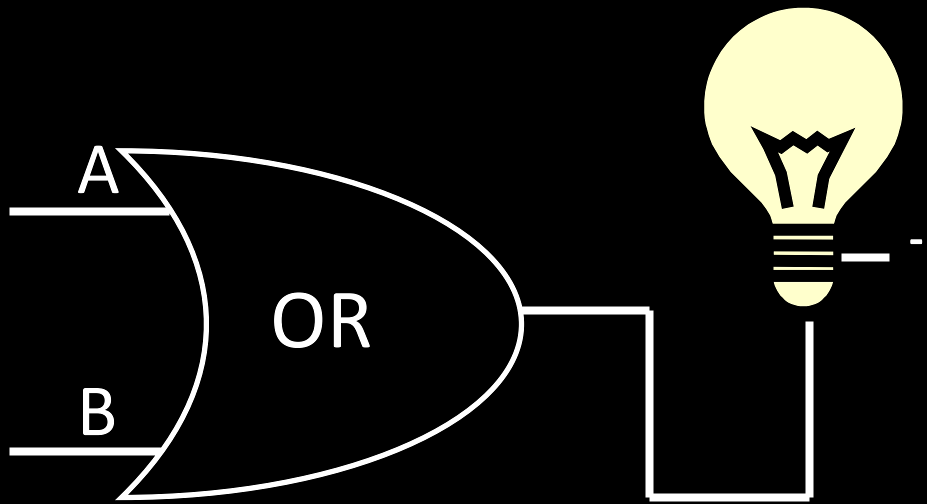
A	B	Light
OFF	OFF	OFF
OFF	ON	ON
ON	OFF	ON
ON	ON	ON



Both (AND)

A	B	Light
OFF	OFF	OFF
OFF	ON	OFF
ON	OFF	OFF
ON	ON	ON

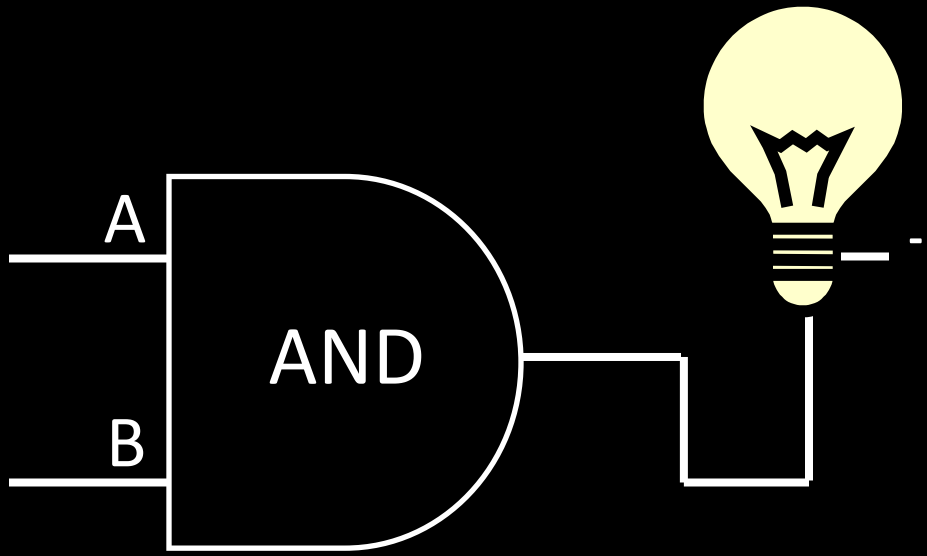
Basic Building Blocks: Switches to Logic Gates



Either (OR)

Truth Table

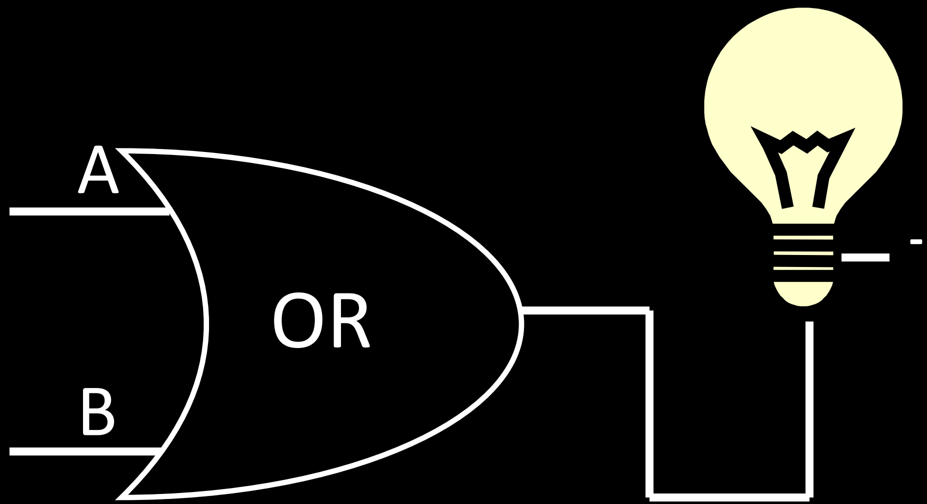
A	B	Light
OFF	OFF	OFF
OFF	ON	ON
ON	OFF	ON
ON	ON	ON



Both (AND)

A	B	Light
OFF	OFF	OFF
OFF	ON	OFF
ON	OFF	OFF
ON	ON	ON

Basic Building Blocks: Switches to Logic Gates

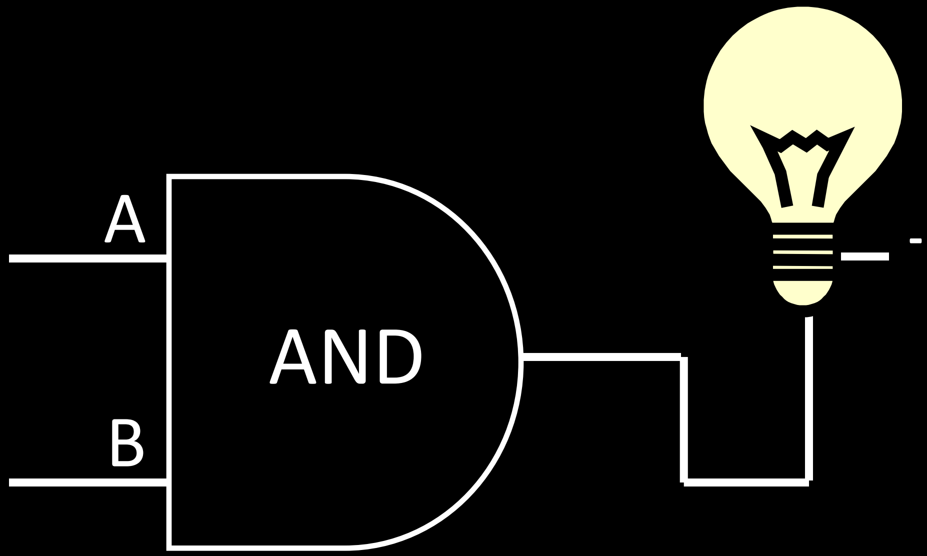


Either (OR)

Truth Table

A	B	Light
0	0	0
0	1	1
1	0	1
1	1	1

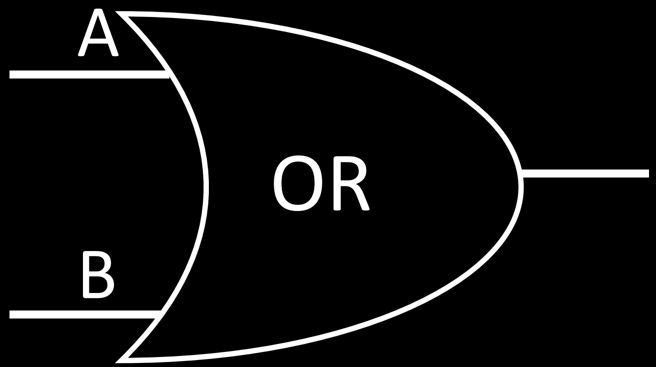
0 = OFF
1 = ON

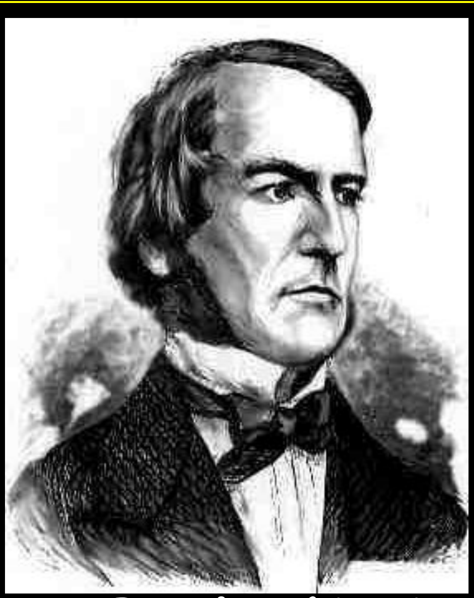


Both (AND)

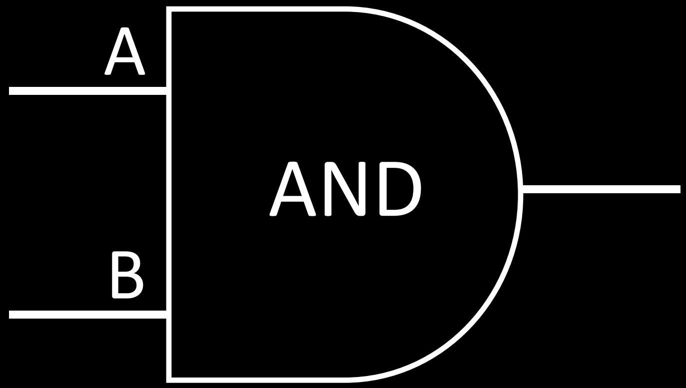
A	B	Light
0	0	0
0	1	0
1	0	0
1	1	1

Basic Building Blocks: Switches to Logic Gates





George Boole,(1815-1864)



Did you know?

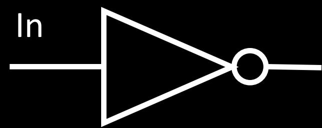
George Boole Inventor of the idea of logic gates. He was born in Lincoln, England and he was the son of a shoemaker in a low class family.

Takeaway

Binary (two symbols: true and false) is the basis of Logic Design

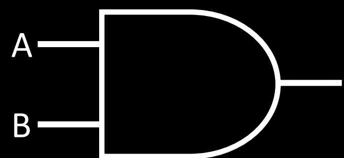
Building Functions: Logic Gates

NOT:



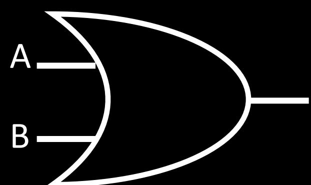
A	Out

AND:



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

OR:



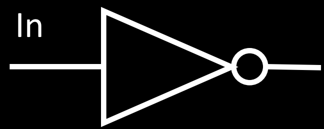
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates

- digital circuit that either allows a signal to pass through it or not.
- Used to build logic functions
- There are seven basic logic gates:
  AND, OR, **NOT**,
  NAND (not AND), NOR (not OR), XOR, and XNOR (not XOR) [later]

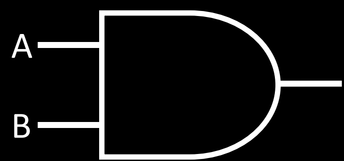
Building Functions: Logic Gates

NOT:



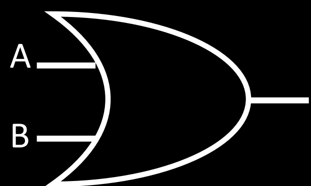
A	Out
0	1
1	0

AND:



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

OR:



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates

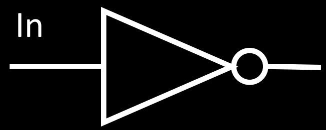
- digital circuit that either allows a signal to pass through it or not.
- Used to build logic functions
- There are seven basic logic gates:

  AND, OR, **NOT**,

  NAND (not AND), NOR (not OR), XOR, and XNOR (not XOR) [later]

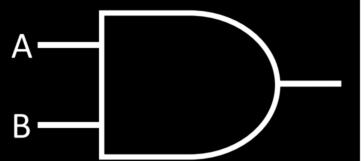
Building Functions: Logic Gates

NOT:



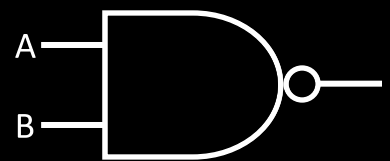
A	Out
0	1
1	0

AND:



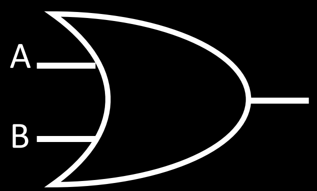
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

NAND:



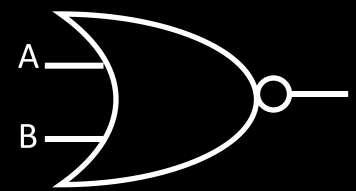
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

OR:



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

NOR:



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gates

- digital circuit that either allows a signal to pass through it or not.
- Used to build logic functions
- There are seven basic logic gates:

  AND, OR, **NOT**,

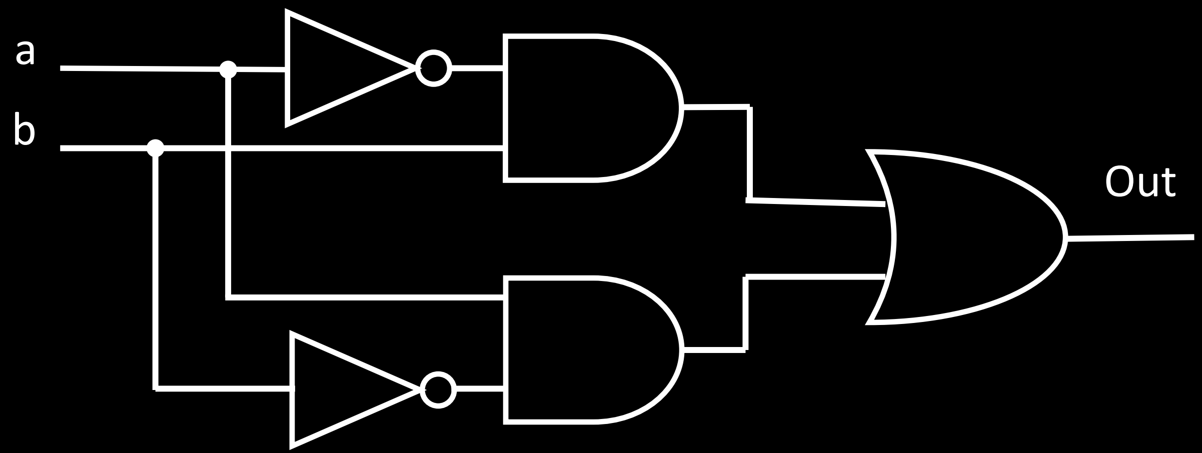
  NAND (not AND), NOR (not OR), XOR, and XNOR (not XOR) [later]

Activity#1.A: Logic Gates

Fill in the truth table, given the following Logic Circuit made from Logic AND, OR, and NOT gates.

What does the logic circuit do?

a	b	Out



Activity#1.A: Logic Gates

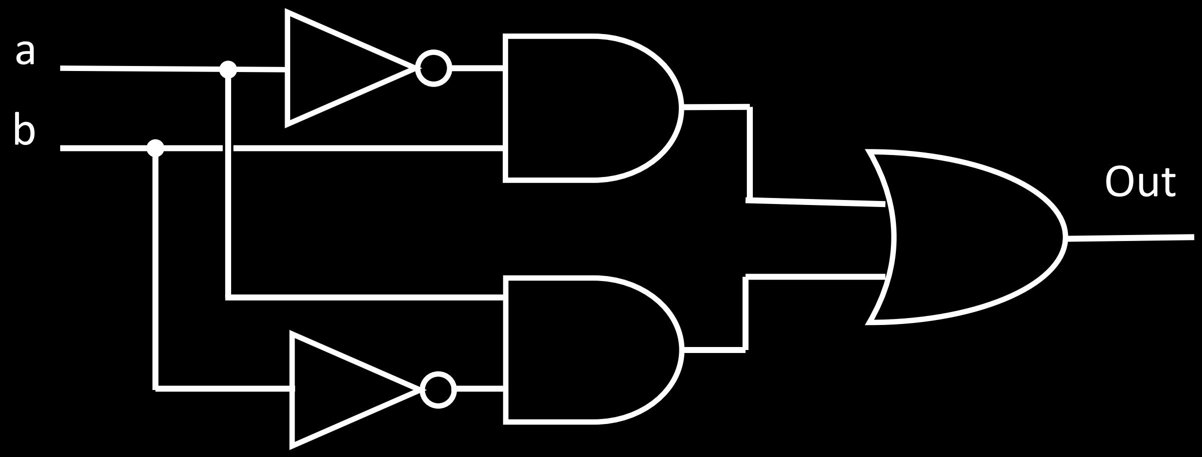
XOR: out = 1 if a or b is 1, but not both;

out = 0 otherwise.

out = 1, only if a = 1 AND b = 0

OR a = 0 AND b = 1

a	b	Out
0	0	0
0	1	1
1	0	1
1	1	0



Activity#1.A: Logic Gates

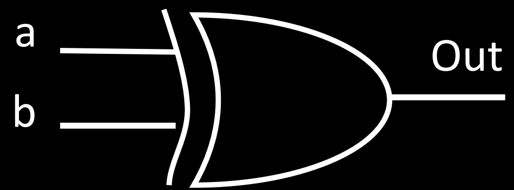
XOR: out = 1 if a or b is 1, but not both;

out = 0 otherwise.

out = 1, only if a = 1 AND b = 0

OR a = 0 AND b = 1

a	b	Out
0	0	0
0	1	1
1	0	1
1	1	0

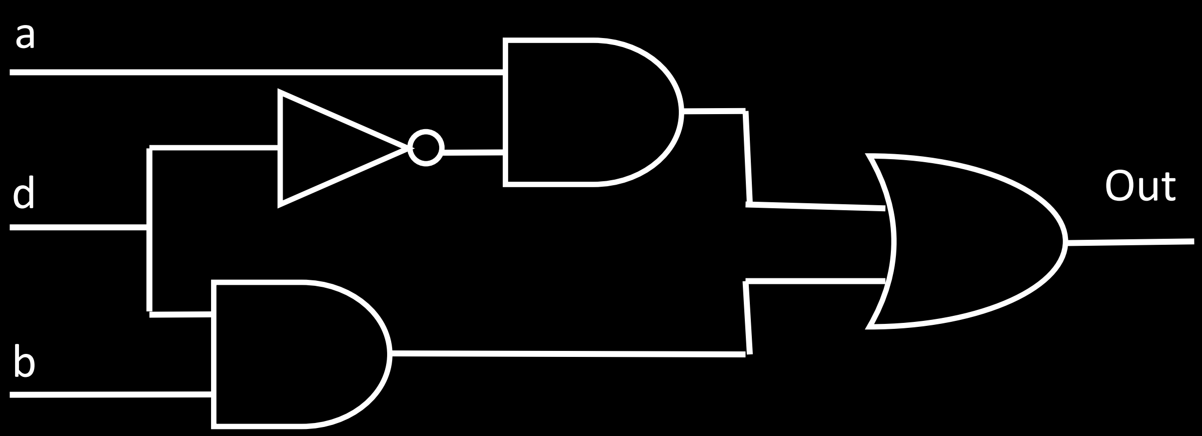


Activity#1: Logic Gates

Fill in the truth table, given the following Logic Circuit made from Logic AND, OR, and NOT gates.

What does the logic circuit do?

a	b	d	Out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



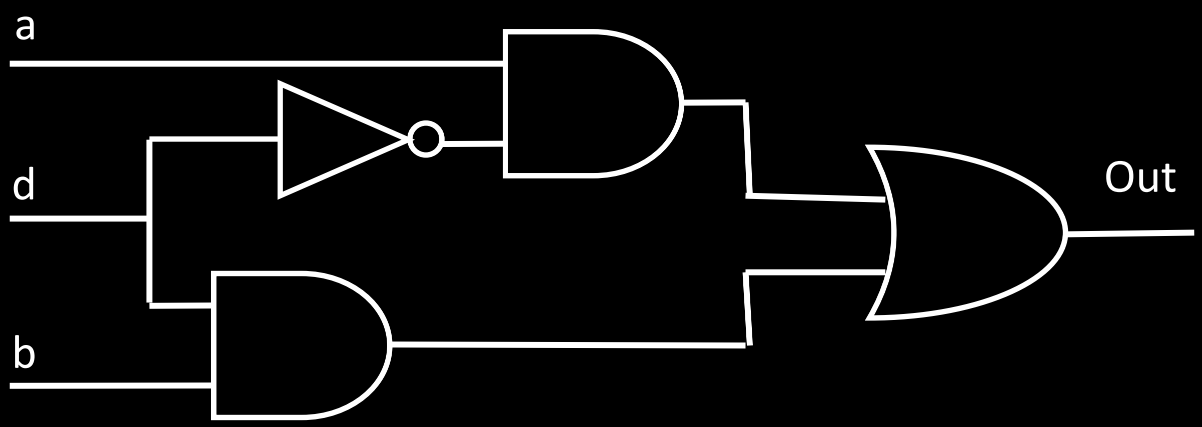
Activity#1: Logic Gates

Multiplexor: select (d) between two inputs (a and b) and set one as the output (out)?

out = a, if d = 0

out = b, if d = 1

a	b	d	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



Goals for Today

From Switches to Logic Gates to Logic Circuits

Logic Gates

- From switches
- Truth Tables

Logic Circuits

- Identity Laws
- From Truth Tables to Circuits (Sum of Products)

Logic Circuit Minimization

- Algebraic Manipulations
- Truth Tables (Karnaugh Maps)

Transistors (electronic switch)

Next Goal

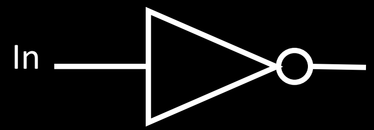
Given a Logic function, create a Logic Circuit that implements the Logic Function…

…and, *with the minimum number of logic gates*

Fewer gates: A cheaper ($$$) circuit!

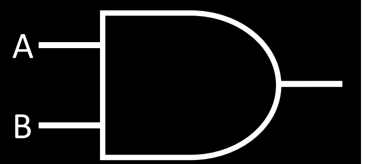
Logic Gates

NOT:



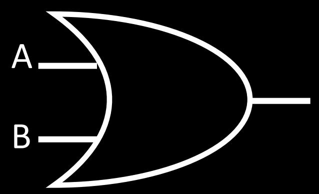
A	Out
0	1
1	0

AND:



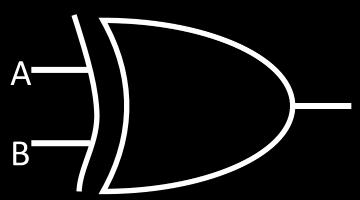
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

OR:



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

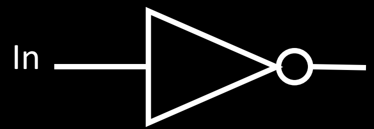
XOR:



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

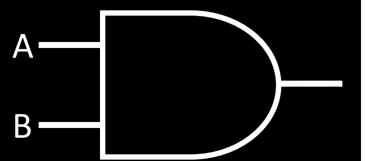
Logic Gates

NOT:



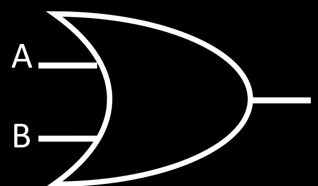
A	Out
0	1
1	0

AND:



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

OR:

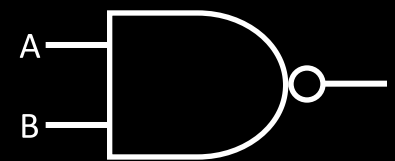


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

XOR:

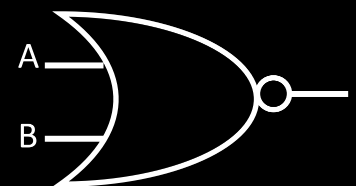
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

NAND:



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

NOR:



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

XNOR:

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

Logic Equations

NOT:

- out = $\bar{a}$ = !a = $\neg a$

AND:

- out = $a \cdot b$ = a & b = $a \wedge b$

OR:

- out = a + b = a | b = $a \vee b$

XOR:

- out = $a \oplus b = a\bar{b} + \bar{a}b$

Logic Equations

- Constants: true = 1, false = 0
- Variables: a, b, out, …
- Operators (above): AND, OR, NOT, etc.

Logic Equations

NOT:

- out = $\bar{a}$ = !a = $\neg a$

AND:

- out = $a \cdot b$ = a & b = $a \wedge b$

OR:

- out = $a + b$ = a | b = $a \vee b$

XOR:

- out = $a \oplus b = a\bar{b} + \bar{a}b$

NAND:

- out = $\overline{a \cdot b}$ = !(a & b) = $\neg (a \wedge b)$

NOR:

- out = $\overline{a + b}$ = !(a | b) = $\neg (a \vee b)$

XNOR:

- out = $\overline{a \oplus b} = ab + \overline{ab}$

Logic Equations

- Constants: true = 1, false = 0
- Variables: a, b, out, …
- Operators (above): AND, OR, NOT, etc.

Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$a + 0 =$

$a + 1 =$

$a + \bar{a} =$

$a \cdot 0 =$

$a \cdot 1 =$

$a \cdot \bar{a} =$

Identities

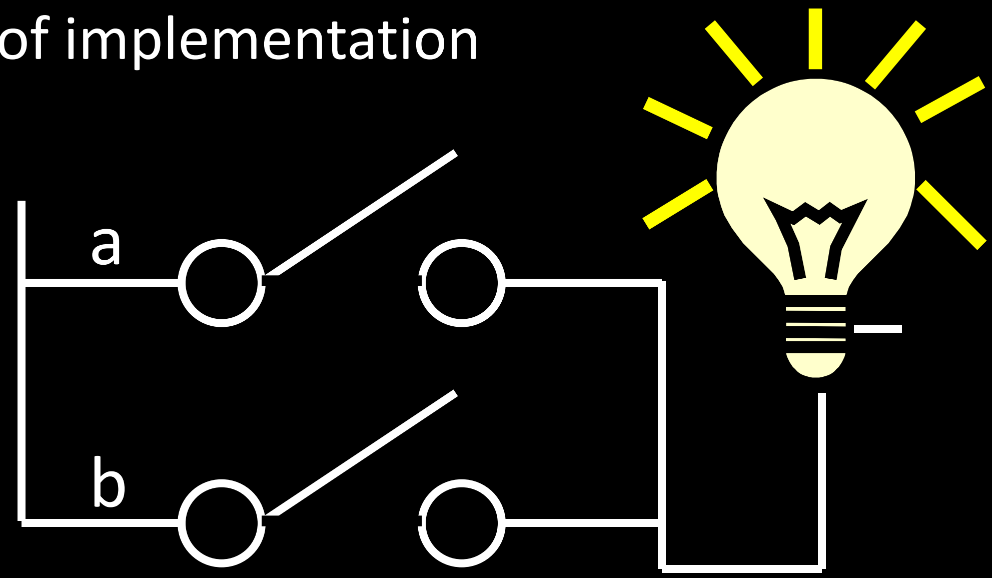
Identities useful for manipulating logic equations

– For optimization & ease of implementation

$a + 0 = a$

$a + 1 = 1$

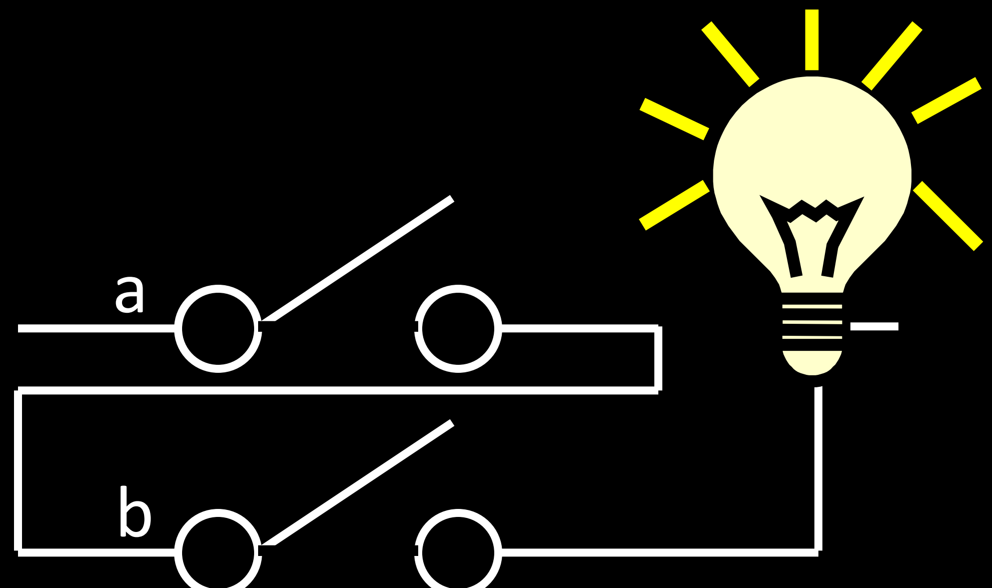
$a + \bar{a} = 1$



$a \cdot 0 = 0$

$a \cdot 1 = a$

$a \cdot \bar{a} = 0$



Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$\overline{(a + b)}$ =

$\overline{(a \cdot b)}$ =

a + a b =

a(b+c) =

$\overline{a(b + c)}$ =

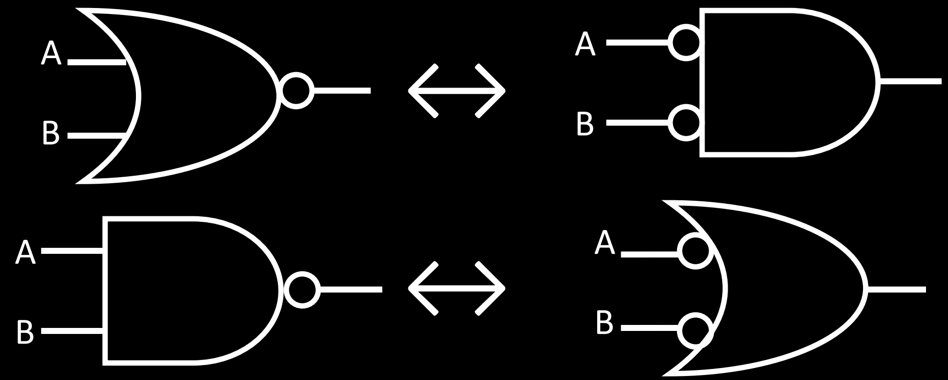
Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

$\overline{(a + b)} = \bar{a} \cdot \bar{b}$

$\overline{(a \cdot b)} = \bar{a} + \bar{b}$



$a + a\,b = a$

$a(b+c) = ab + ac$

$\overline{a(b + c)} = \bar{a} + \bar{b} \cdot \bar{c}$

Activity #2: Identities

$a + 0 = a$

$a + 1 = 1$

$a + \bar{a} = 1$

$a\,0 = 0$

$a\,1 = a$

$a\,\bar{a} = 0$

$\overline{(a + b)} = \bar{a}\,\bar{b}$

$\overline{(a\,b)} = \bar{a} + \bar{b}$

$a + a\,b = a$

$a(b+c) = ab + ac$

$\overline{a(b + c)} = \bar{a} + \overline{bc}$

Show that the Logic equations below are equivalent.

$(a+b)(a+c) = a + bc$

$(a+b)(a+c) =$

Activity #2: Identities

$a + 0 = a$

$a + 1 = 1$

$a + \bar{a} = 1$

$a\,0 = 0$

$a\,1 = a$

$a\,\bar{a} = 0$

$\overline{(a + b)} = \bar{a}\,\bar{b}$

$\overline{(a\,b)} = \bar{a} + \bar{b}$

$a + a\,b = a$

$a(b+c) = ab + ac$

$\overline{a(b + c)} = \bar{a} + \overline{bc}$

Show that the Logic equations below are equivalent.

$(a+b)(a+c) = a + bc$

$$
\begin{aligned}
(a+b)(a+c) &= aa + ab + ac + bc \\
&= a + a(b+c) + bc \\
&= a(1 + (b+c)) + bc \\
&= a + bc
\end{aligned}
$$

Logic Manipulation

- functions: gates ↔ truth tables ↔ equations
- Example: (a+b)(a+c) = a + bc

a	b	c					
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Logic Manipulation

- functions: gates ↔ truth tables ↔ equations
- Example: (a+b)(a+c) = a + bc

a	b	c	a+b	a+c	LHS	bc	RHS
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Takeaway

Binary (two symbols: true and false) is the basis of Logic Design

More than one Logic Circuit can implement same Logic function. Use Algebra (Identities) or Truth Tables to show equivalence.

Goals for Today

From Switches to Logic Gates to Logic Circuits

Logic Gates

- From switches
- Truth Tables

Logic Circuits

- Identity Laws
- From Truth Tables to Circuits (Sum of Products)

Logic Circuit Minimization

- Algebraic Manipulations
- Truth Tables (Karnaugh Maps)

Transistors (electronic switch)

Next Goal

How to standardize minimizing logic circuits?

Logic Minimization

How to implement a desired logic function?

a	b	c	out
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**0**

Logic Minimization

How to implement a desired logic function?

a	b	c	out	minterm
0	**0**	**0**	**0**	$\bar{a}\,\bar{b}\,\bar{c}$
0	**0**	**1**	**1**	$\bar{a}\,\bar{b}\,c$
0	**1**	**0**	**0**	$\bar{a}\,b\,\bar{c}$
0	**1**	**1**	**1**	$\bar{a}\,b\,c$
1	**0**	**0**	**0**	$a\,\bar{b}\,\bar{c}$
1	**0**	**1**	**1**	$a\,\bar{b}\,c$
1	**1**	**0**	**0**	$a\,b\,\bar{c}$
1	**1**	**1**	**0**	$a\,b\,c$

1) Write minterm's
2) sum of products:

- OR of all minterms where out=1

Logic Minimization

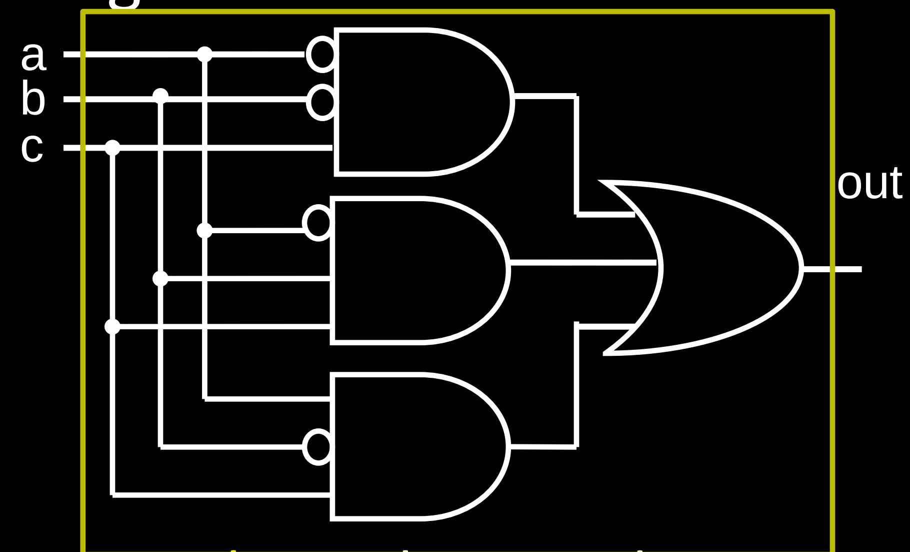
How to implement a desired logic function?

a	b	c	out	minterm
0	**0**	**0**	**0**	$\bar{a}\,\bar{b}\,\bar{c}$
0	**0**	**1**	**1**	$\bar{a}\,\bar{b}\,c$
0	**1**	**0**	**0**	$\bar{a}\,b\,\bar{c}$
0	**1**	**1**	**1**	$\bar{a}\,b\,c$
1	**0**	**0**	**0**	$a\,\bar{b}\,\bar{c}$
1	**0**	**1**	**1**	$a\,\bar{b}\,c$
1	**1**	**0**	**0**	$a\,b\,\bar{c}$
1	**1**	**1**	**0**	$a\,b\,c$

1) Write minterm's
2) sum of products:

- OR of all minterms where out=1

  - E.g. out = $\overline{ab}c + \bar{a}bc + a\bar{b}c$



corollary: *any* combinational circuit *can be* implemented in two levels of logic (ignoring inverters)

Karnaugh Maps

How does one find the most efficient equation?

- Manipulate algebraically until…?
- Use Karnaugh maps (optimize visually)
- Use a software optimizer

For large circuits

- Decomposition & reuse of building blocks

Minimization with Karnaugh maps (1)

a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

- Sum of minterms yields
  - out =

Minimization with Karnaugh maps (2)

- Sum of minterms yields
  - out = $\overline{ab}c + \bar{a}bc + a\overline{bc} + a\bar{b}c$

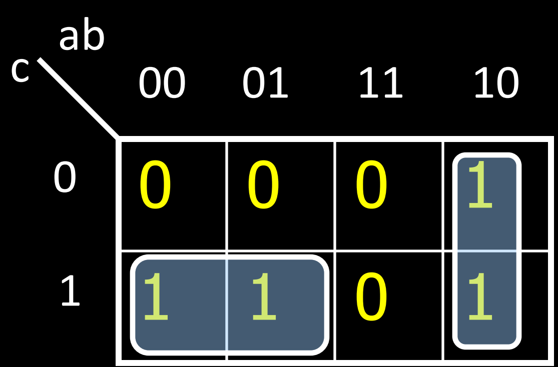
- Karnaugh maps identify which inputs are (ir)relevant to the output

a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

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

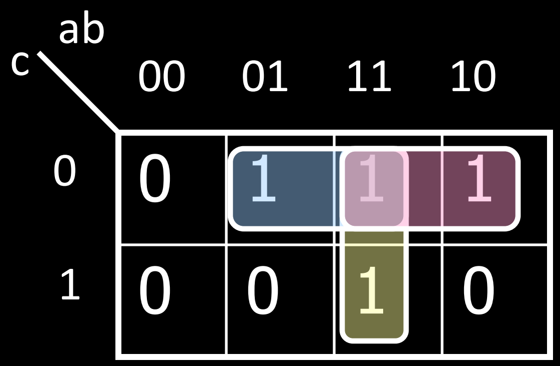
Minimization with Karnaugh maps (2)

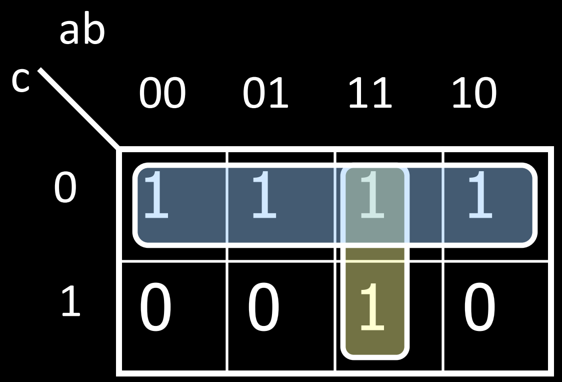
a	b	c	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



- Sum of minterms yields
  - out = $\overline{ab}c + \bar{a}bc + a\overline{bc} + a\bar{b}c$
- Karnaugh map minimization
  - Cover all 1's
  - Group adjacent blocks of 2^n 1's that yield a rectangular shape
  - Encode the common features of the rectangle
    - out = $a\bar{b} + \bar{a}c$

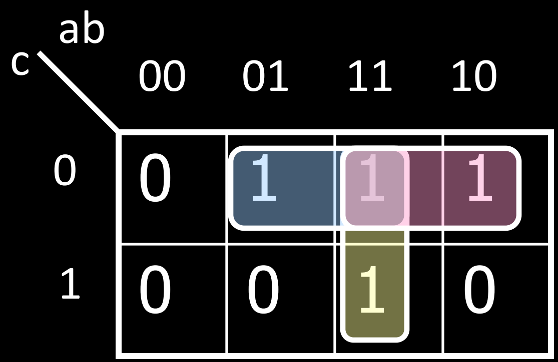
Karnaugh Minimization Tricks (1)

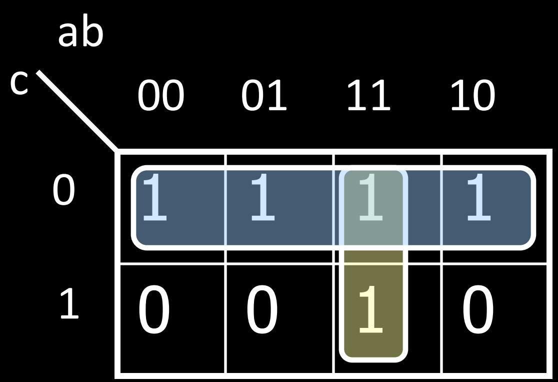




- Minterms can overlap
  - out =
- Minterms can span 2, 4, 8 or more cells
  - out =

Karnaugh Minimization Tricks (1)





- Minterms can overlap
  - out = $b\bar{c} + a\bar{c} + ab$
- Minterms can span 2, 4, 8 or more cells
  - out = $\bar{c} + ab$

Karnaugh Minimization Tricks (2)

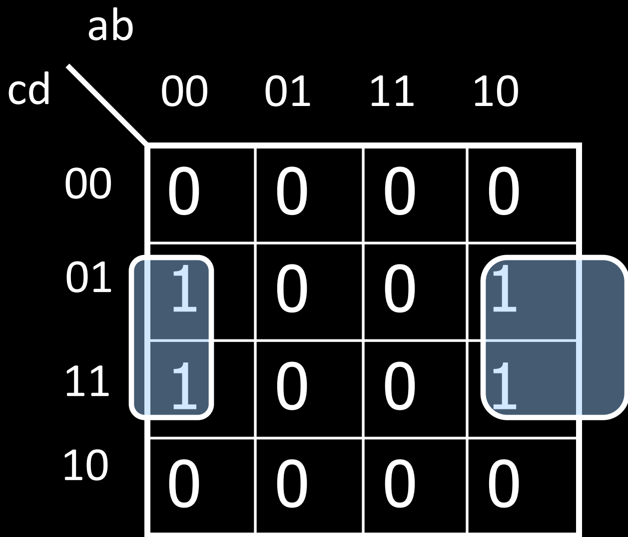
cd \ ab	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

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

The map wraps around

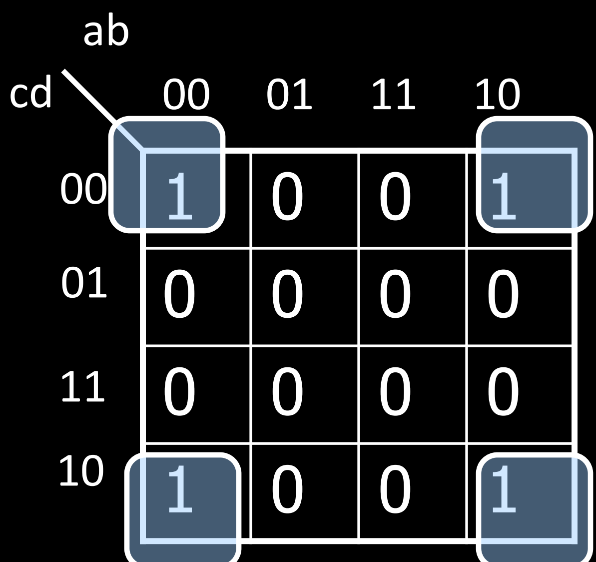
- out =

Karnaugh Minimization Tricks (2)



The map wraps around

- out = $\bar{b}d$



- out = $\overline{bd}$

Karnaugh Minimization Tricks (3)

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

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

"Don't care" values can be interpreted individually in whatever way is convenient

- assume all x's = 1
- out =

- assume middle x's = 0
- assume 4th column x = 1
- out =

Karnaugh Minimization Tricks (3)

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

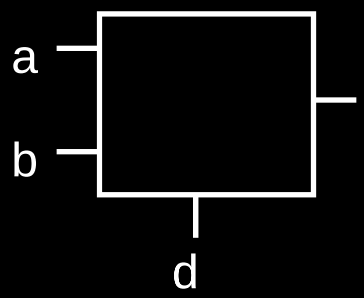
"Don't care" values can be interpreted individually in whatever way is convenient

- assume all x's = 1
- out = d

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

- assume middle x's = 0
- assume 4th column x = 1
- out = $\overline{bd}$

Multiplexer



A multiplexer selects between multiple inputs

- out = a, if d = 0
- out = b, if d = 1

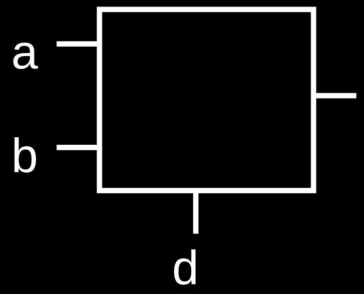
a	b	d	out
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Build truth table

Minimize diagram

Derive logic diagram

Multiplexer Implementation

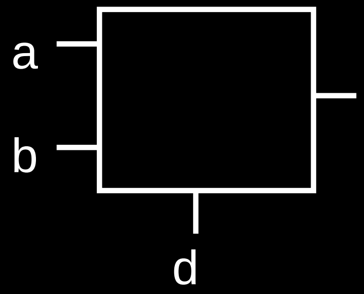


- Build a truth table

$out = \bar{a}bd + a\overline{bd} + ab\bar{d} + abd$

a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Multiplexer Implementation

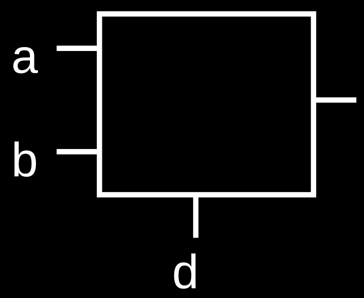


a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Build the Karnaugh map

d \ ab	00	01	11	10
0				
1				

Multiplexer Implementation

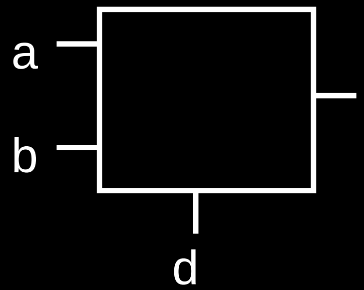


- Build the Karnaugh map

a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

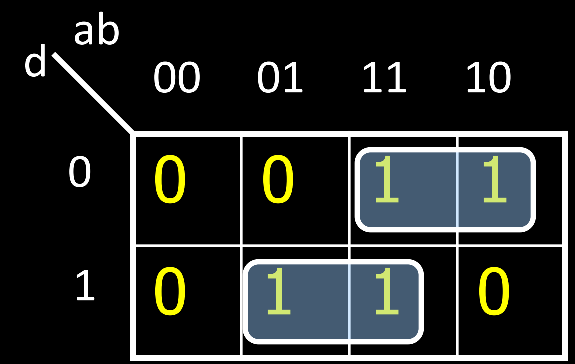
d \ ab	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Multiplexer Implementation

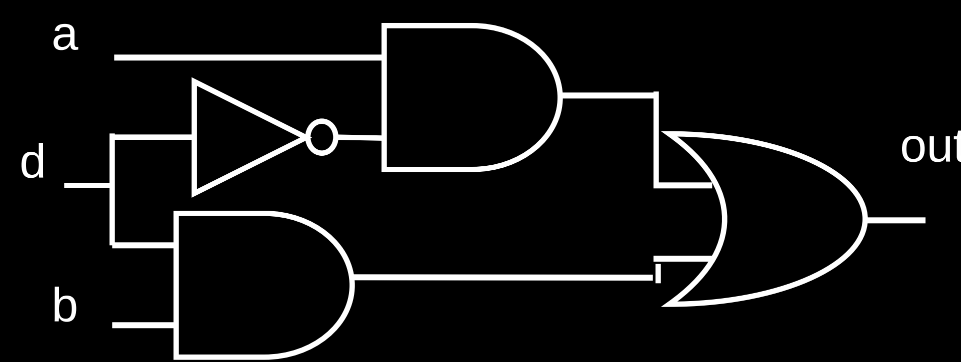


a	b	d	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Derive Minimal Logic Equation



- out = $a\bar{d} + bd$



Takeaway

Binary (two symbols: true and false) is the basis of Logic Design

More than one Logic Circuit can implement same Logic function. Use Algebra (Identities) or Truth Tables to show equivalence.

Any logic function can be implemented as "sum of products". Karnaugh Maps minimize number of gates.

Goals for Today

From Transistors to Gates to Logic Circuits

Logic Gates

- From transistors
- Truth Tables

Logic Circuits

- Identity Laws
- From Truth Tables to Circuits (Sum of Products)

Logic Circuit Minimization

- Algebraic Manipulations
- Truth Tables (Karnaugh Maps)

Transistors (electronic switch)

Activity#1 How do we build *electronic* switches?

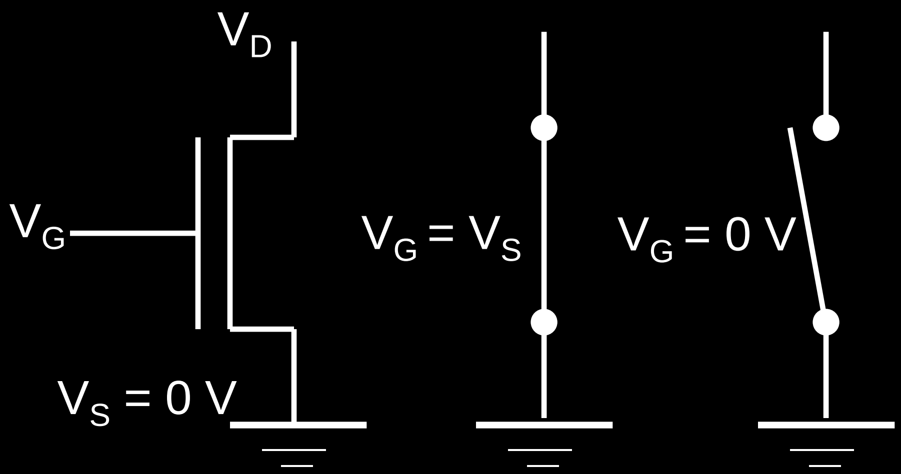
Transistors:

- 6:10 minutes (watch from from 41s to 7:00)
- http://www.youtube.com/watch?v=QO5FgM7MLGg

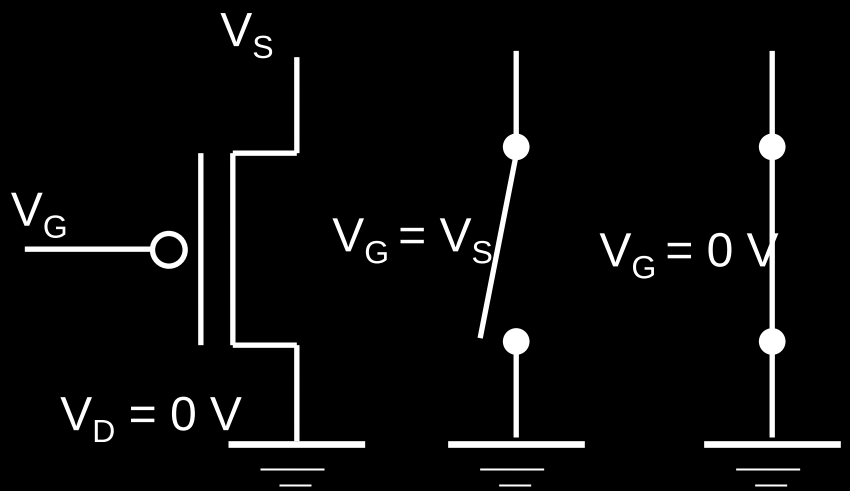
- Fill our Transistor Worksheet with info from Video

NMOS and PMOS Transistors

- NMOS Transistor



PMOS Transistor



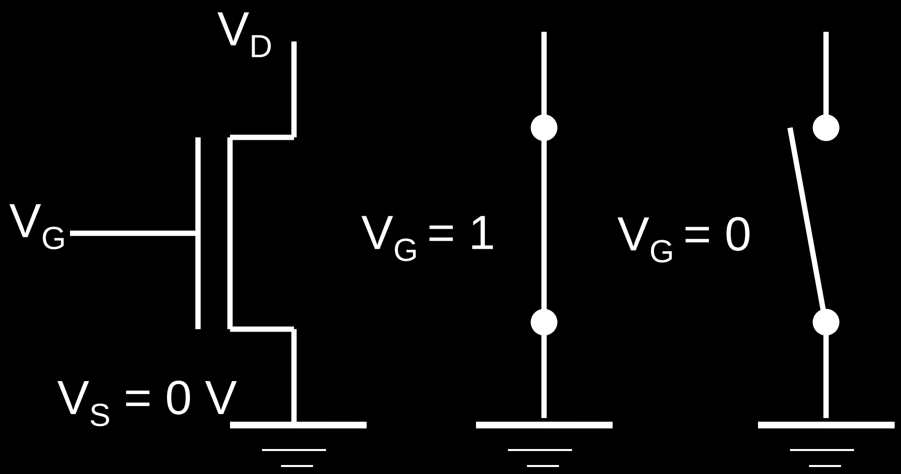
- Connect source to drain when gate = 1
- N-channel

Connect source to drain when gate = 0

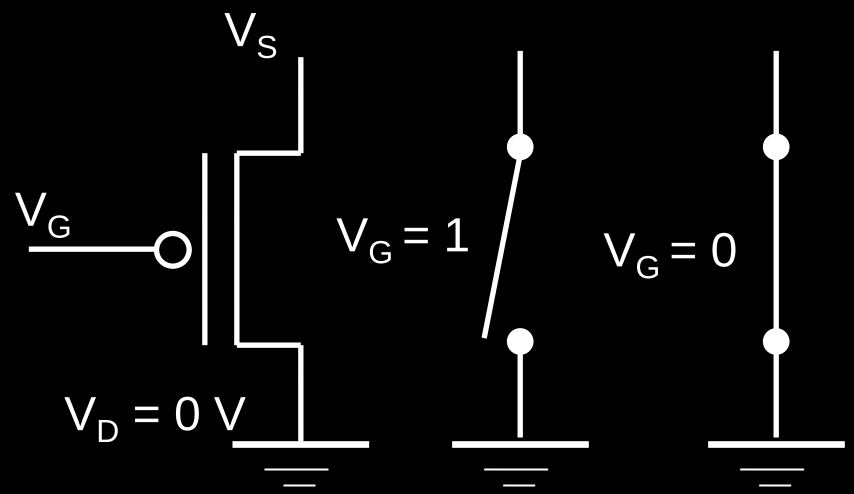
P-channel

NMOS and PMOS Transistors

- NMOS Transistor



PMOS Transistor

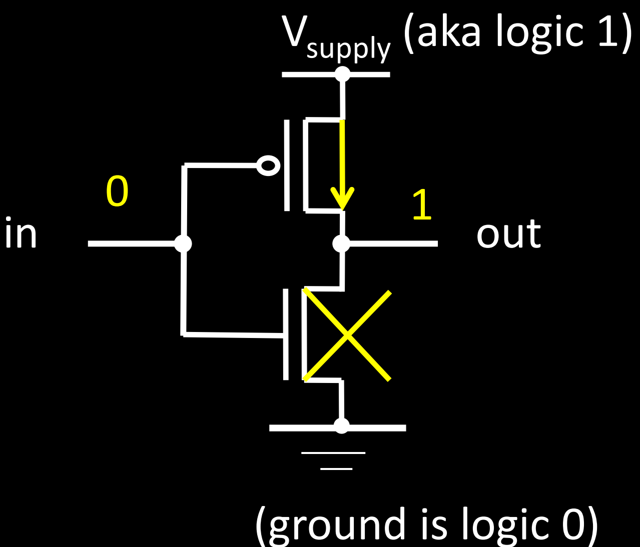


- Connect source to drain when gate = 1
- N-channel

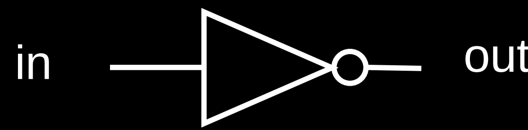
Connect source to drain when gate = 0

P-channel

Inverter



- Function: NOT
- Called an inverter
- Symbol:



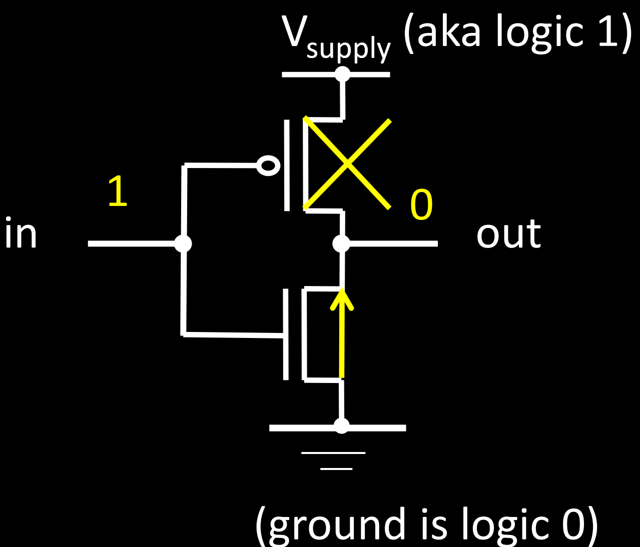
In	Out
0	1
1	0

Truth table

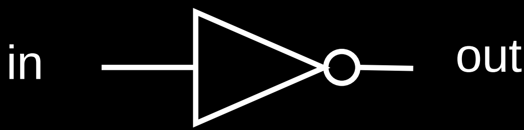
- Useful for taking the inverse of an input

- CMOS: complementary-symmetry metal**–oxide–semiconductor**

Inverter



- Function: NOT
- Called an inverter
- Symbol:

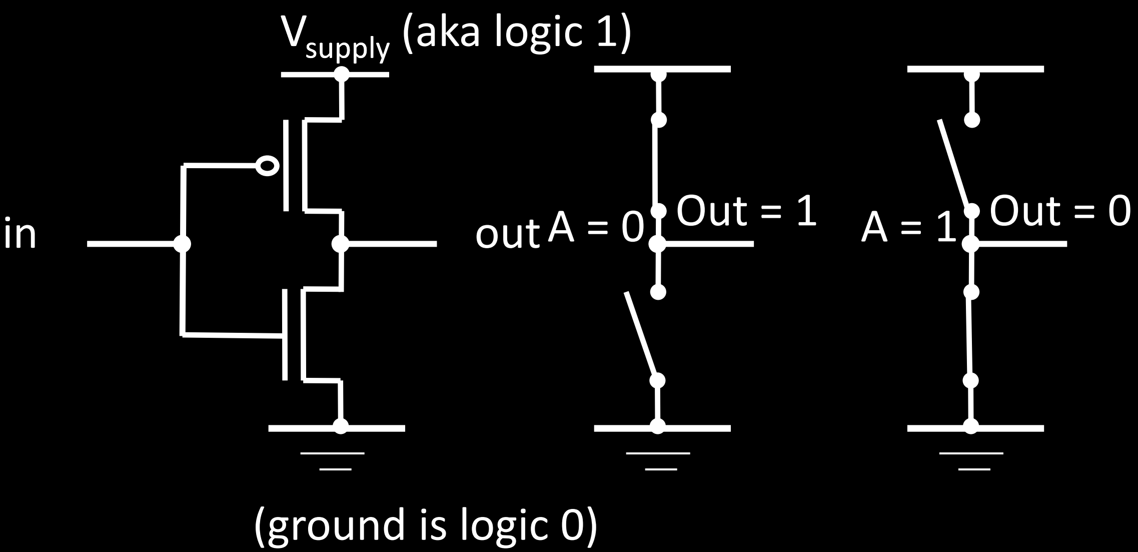


In	Out
0	1
1	0

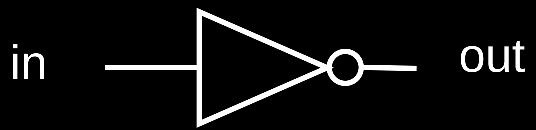
Truth table

- Useful for taking the inverse of an input
- CMOS: complementary-symmetry metal**–oxide–semiconductor**

Inverter



- Function: NOT
- Called an inverter
- Symbol:



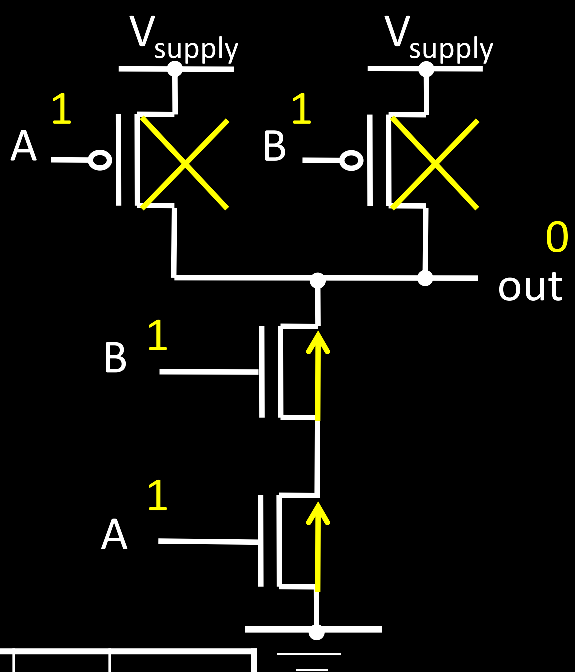
A	Out
0	1
1	0

Truth table

- Useful for taking the inverse of an input

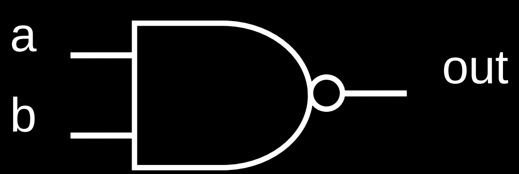
- CMOS: complementary-symmetry metal**–oxide–semiconductor**

NAND Gate

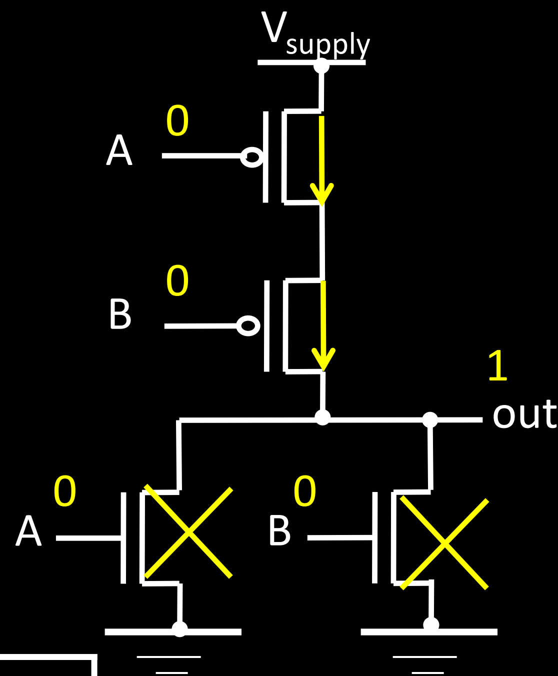


A	B	out
0	0	1
1	0	1
0	1	1
1	1	0

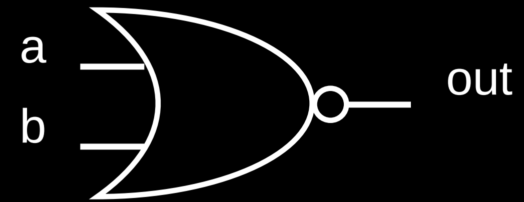
- Function: NAND
- Symbol:



NOR Gate



- Function: NOR
- Symbol:



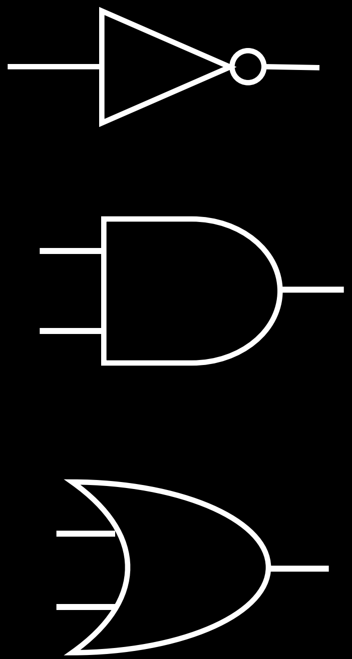
A	B	out
0	0	1
1	0	0
0	1	0
1	1	0

Building Functions (Revisited)

NOT:

AND:

OR:

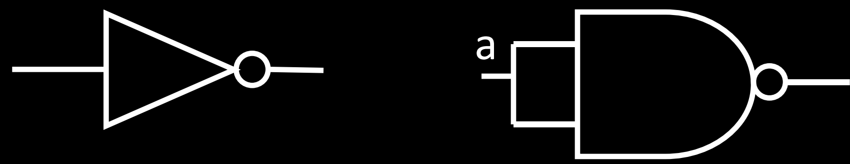


NAND and NOR are universal

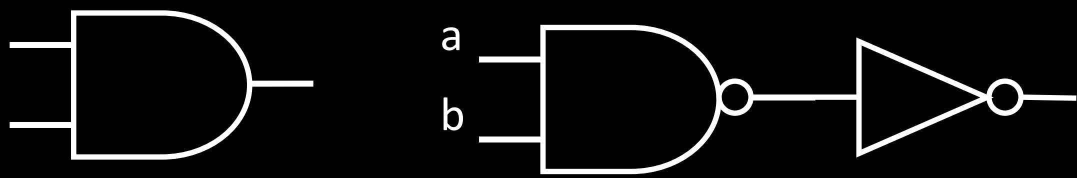
- Can implement any function with NAND or just NOR gates
- useful for manufacturing

Building Functions (Revisited)

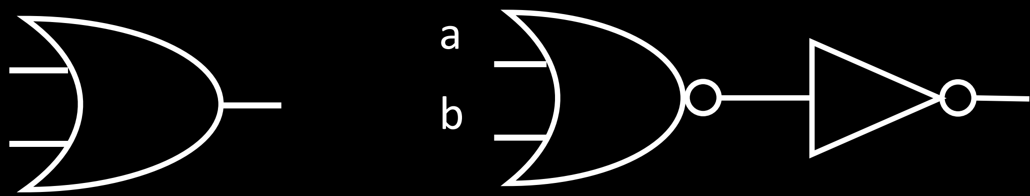
NOT:



AND:



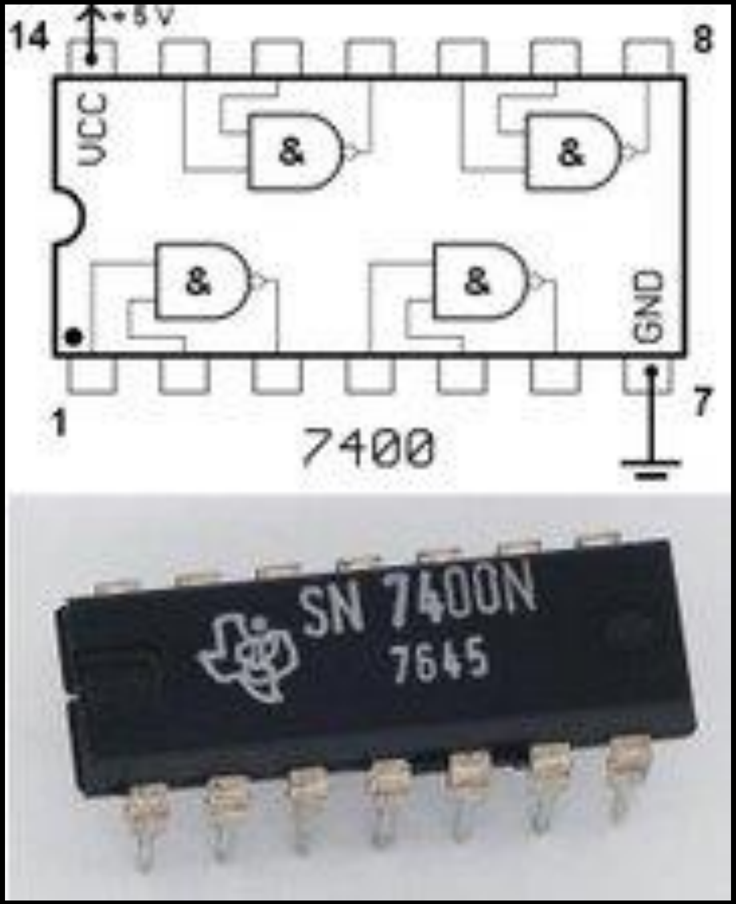
OR:



NAND and NOR are universal

- Can implement any function with NAND or just NOR gates
- useful for manufacturing

Logic Gates

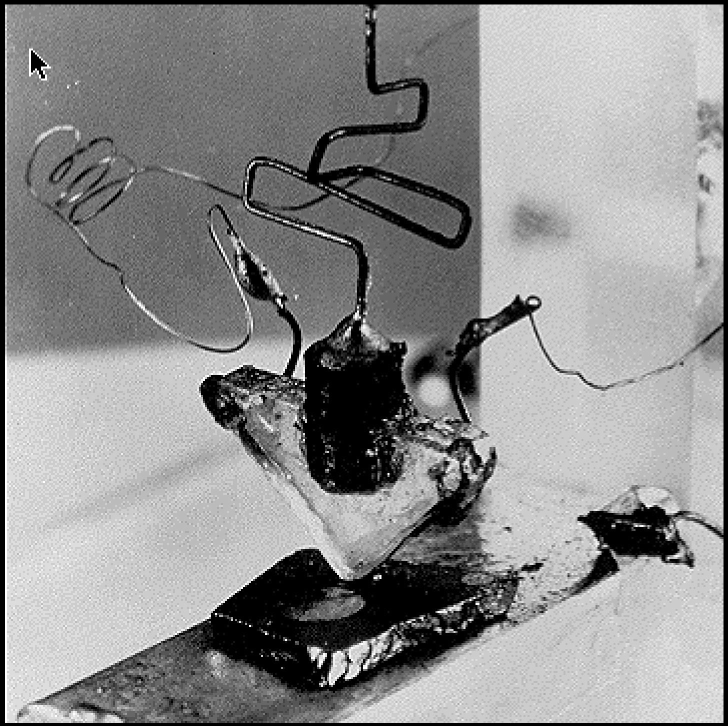


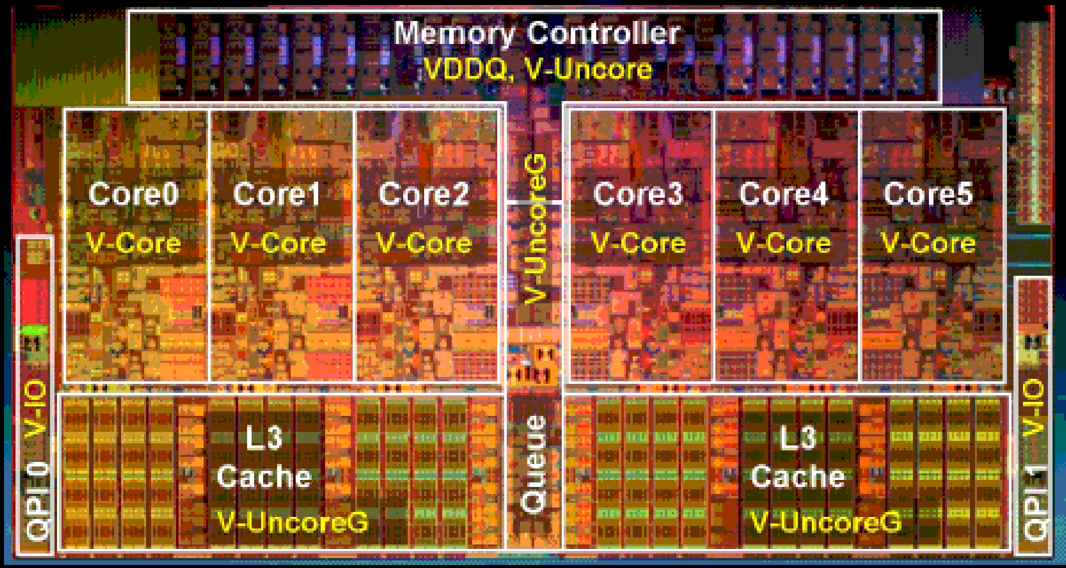
One can buy gates separately

- ex. 74xxx series of integrated circuits
- cost ~$1 per chip, mostly for packaging and testing

Cumbersome, but possible to build devices using gates put together manually

Then and Now





http://www.theregister.co.uk/2010/02/03/intel_westmere_ep_preview/

The first transistor

- on a workbench at

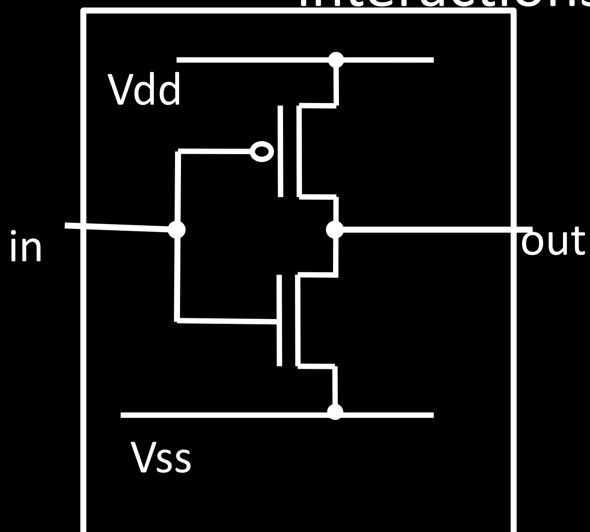
  AT&T Bell Labs in 1947
- Bardeen, Brattain, and Shockley

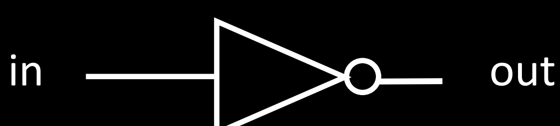
- An Intel Westmere
  - 1.17 billion transistors
  - 240 square millimeters
  - Six processing cores

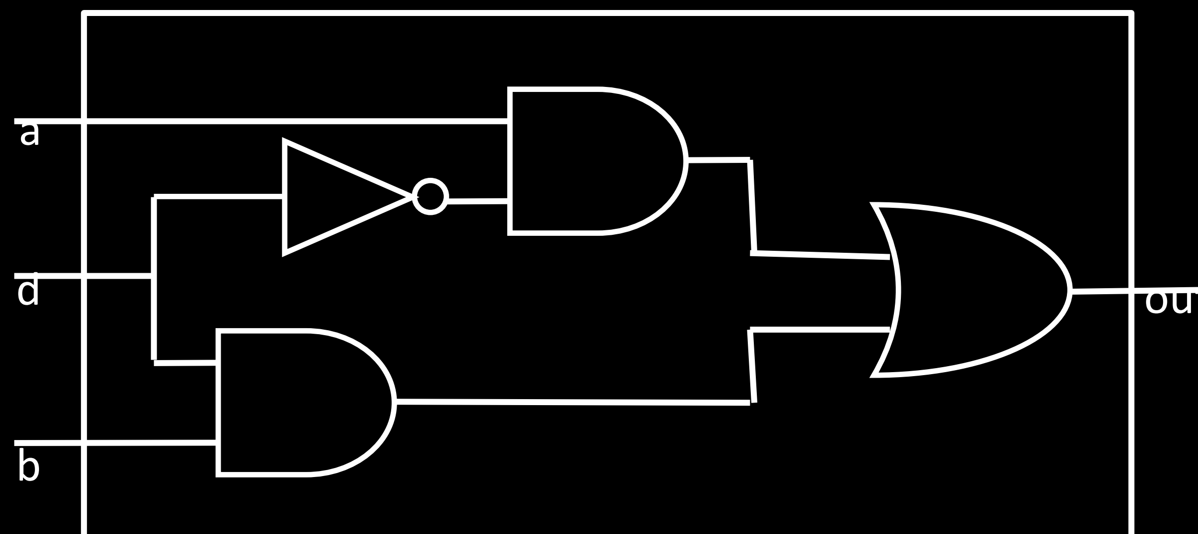
Big Picture: Abstraction

Hide complexity through simple abstractions

- Simplicity
  - Box diagram represents inputs and outputs
- Complexity
  - Hides underlying NMOS- and PMOS-transistors and atomic interactions







Summary

Most modern devices are made from billions of on /off switches called transistors

- We will build a processor in this course!
- Transistors made from semiconductor materials:
  - MOSFET – Metal Oxide Semiconductor Field Effect Transistor
  - NMOS, PMOS – Negative MOS and Positive MOS
  - CMOS – Complimentary MOS made from PMOS and NMOS transistors
- Transistors used to make logic gates and logic circuits

We can now implement any logic circuit

- Can do it efficiently, using Karnaugh maps to find the minimal terms required
- Can use either NAND or NOR gates to implement the logic circuit
- Can use P- and N-transistors to implement NAND or NOR gates