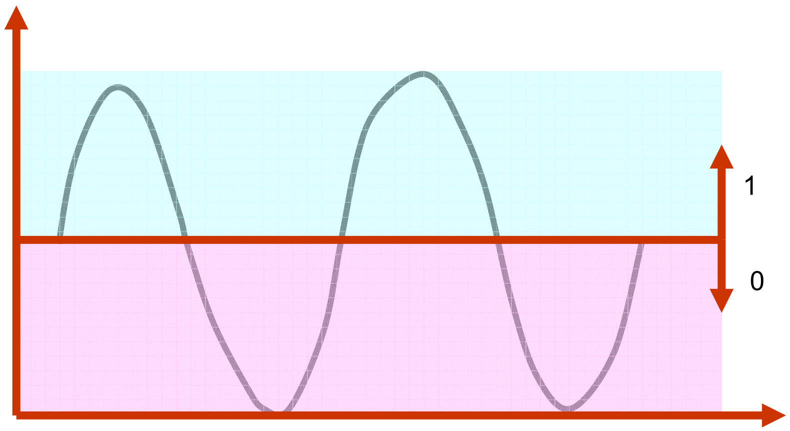
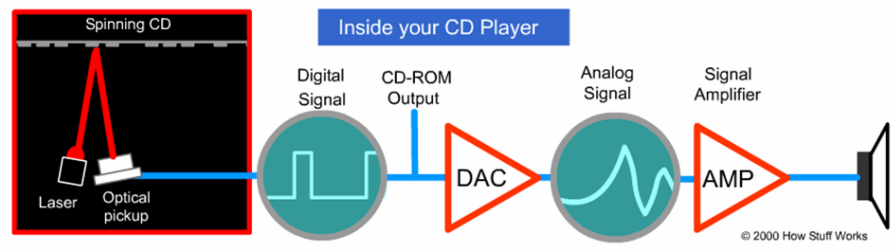
CS 130 : Computer Systems - II

Shankar Balachandran (shankar@cse.iitm.ac.in)
Dept. of Computer Science & Engineering
IIT Madras

Recap



Differentiate Between 0's and 1's



Truth Tables

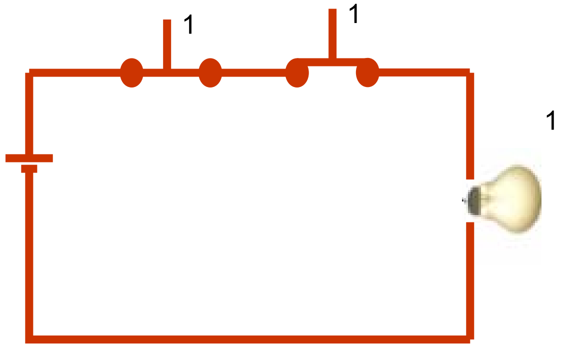
a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0

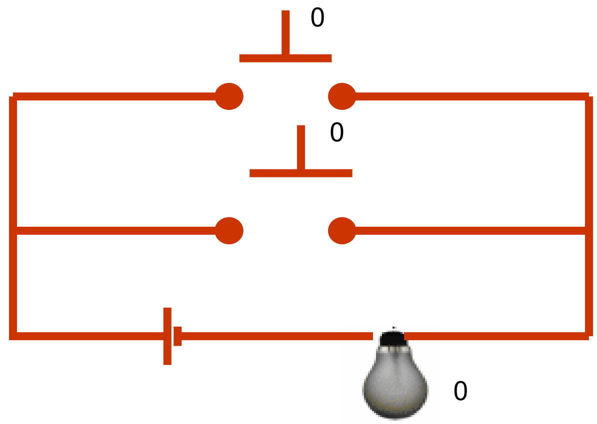
Recap

Logical AND



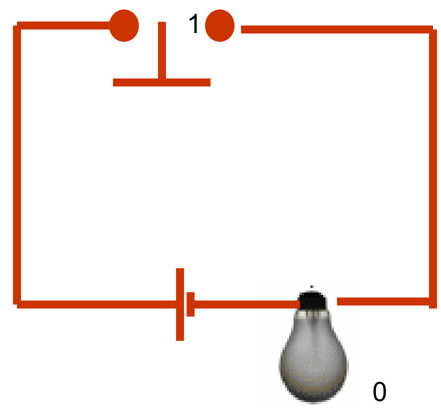
Only configuration which turns bulb ON

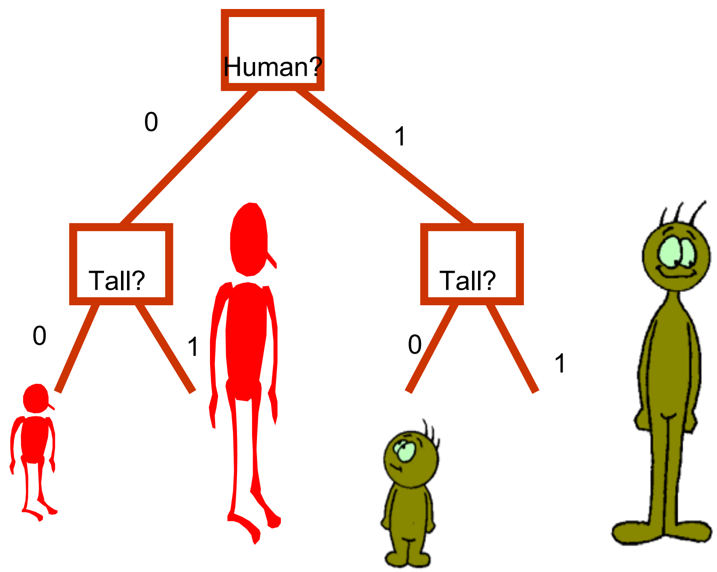
Logical OR



Only configuration in which bulb is OFF

Logical NOT





Algorithm

Logic Functions

- Truth Table for *n* inputs
- Just like mathematical functions that you have learnt
- AND, OR and NOT are the most basic logic functions
- Two other important functions
  - XOR – pronounced ex-or
    - When both are same, output = 0
  - XNOR – pronounced ex-nor

		XOR
0	0	0
0	1	1
1	0	1
1	1	0

		XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Relationship Between XOR and XNOR

		XOR
0	0	0
0	1	1
1	0	1
1	1	0

		XNOR
0	0	1
0	1	0
1	0	0
1	1	1

- Whenever XOR is 0, XNOR is 1
- Whenever XOR is 1, XNOR is 0
- XOR(a,b) = NOT (XNOR(a,b))
- Also
  - XNOR(a,b) = NOT (XOR(a,b))

Let's Enumerate Logic Functions

- One variable : *a*
- What are the possible functions on *a* ?
- Let's do the truth tables

a	f0
0	0
1	0

a	f1
0	0
1	1

a	f2
0	1
1	0

a	f3
0	1
1	1

Four Functions of One Variable

- f0 : Equivalent to **0**
- f1 : Equivalent to *a*
- f2 : Equivalent to NOT(a)
- f3 : Equivalent to **1**
- Notation :
  - NOT(A) is denoted as $\bar{a}$
  - Pronounced a-bar
- Let's rewrite f's as functions of *a*
  - f0 = **0**
  - f1 = *a*
  - f2 = $\bar{a}$
  - f3 = **1**

Functions of Two Variables

- Variables *a* and *b*
- Let's enumerate all functions
  - In Truth Table form

a	b	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

0 **AND** **XOR** **OR** **XNOR** **1**

Functions of Two Variables (Contd.)

- A few tough functions

a	b	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0			0	0	0	0			1		1	1	1	1	1	
0	1			0	0	1	1			0		0	0	1	1	1	
1	0			1	1	0	0			0		1	1	0	0	1	
1	1			0	1	0	1			0		0	1	0	1	0	
					a		b					$\bar{b}$		$\bar{a}$			

Functions of Two Variables (contd.)

- A few complex functions

a	b	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0			0		0				1			1		1	1	
0	1			0		1				0			0		1	1	
1	0			1		0				0			1		0	1	
1	1			0		0				0			1		1	0	

NOT(OR) NOT(AND)

- NOT(AND) = NAND
- NOT (OR) = NOR

Functions of Two Variables (contd.)

- Most complex functions

a	b	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0			0		0							1		1		
0	1			0		1							0		1		
1	0			1		0							1		0		
1	1			0		0							1		1		

a AND (NOT b)

(NOT a) AND b

a OR (NOT b)

(NOT a) OR b

Summary of functions

- One variable – 4 functions
- Two variable – 16 functions
- N variables - 2^{2^N} functions
  - Can you prove this?
- Just like mathematical functions
  - Given a function *f* on N variables, $f(x_1, x_2, \ldots, x_N)$ is unique
    - $x_1, x_2, \ldots, x_N \in (0,1)$
- The functions can be enumerated
  - Gets very large soon though

AND, OR and NOT

- Form a Universal Set
  - All functions can be expressed as combinations of AND, OR and NOT
- Eg :XOR

		XOR
0	0	0
0	1	1
1	0	1
1	1	0

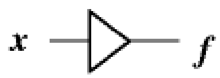
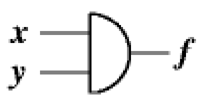
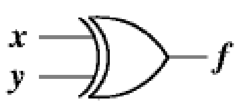
$XOR(a,b) = (\bar{a}\ AND\ b)\ OR\ (a\ AND\ \bar{b})$

- Eg :XNOR

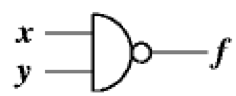
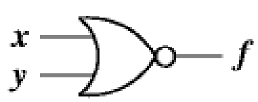
$XNOR(a,b) = (\bar{a}\ AND\ \bar{b})\ OR\ (a\ AND\ b)$

		XNOR
0	0	1
0	1	0
1	0	0
1	1	1

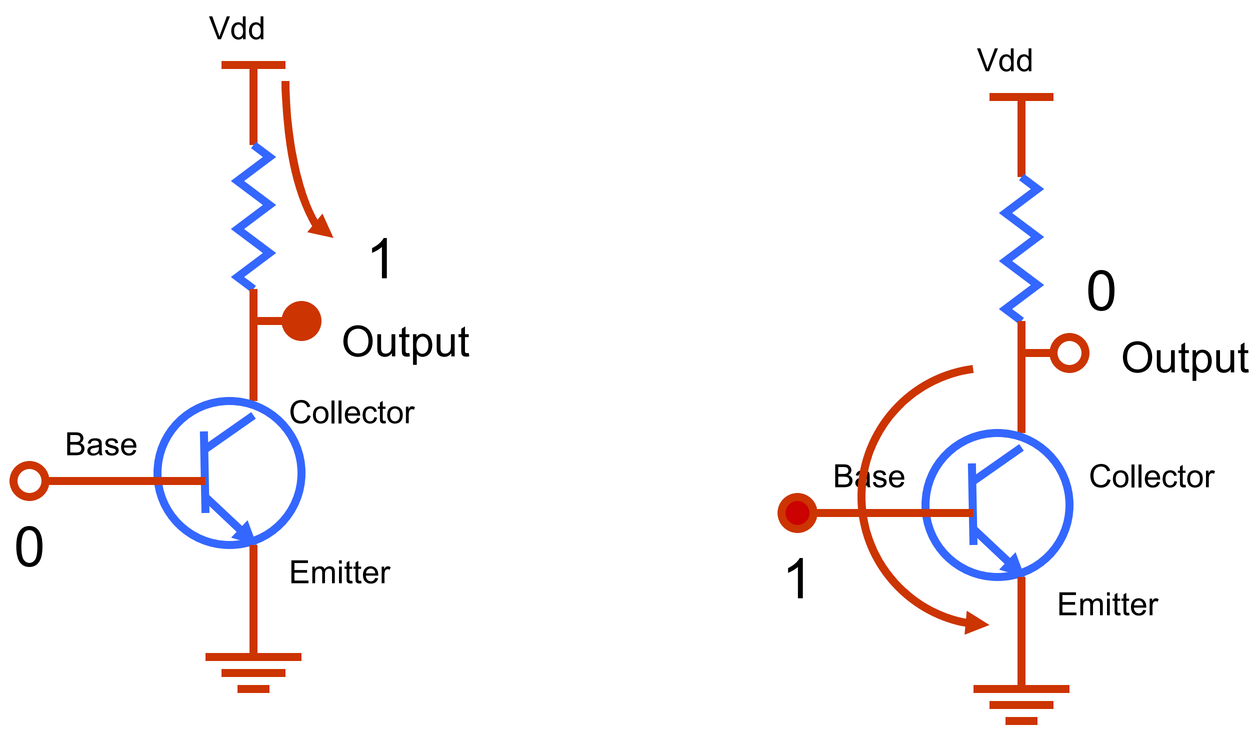
Abstraction Using Symbols

Gate	Schematic Symbol	Algebraic Function	Truth Table
BUFFER		$f = x$	x f 0 0 1 1
AND		$f = xy$	x y f 0 0 0 0 1 0 1 0 0 1 1 1
OR		$f = x + y$	x y f 0 0 0 0 1 1 1 0 1 1 1 1
XOR		$f = x \oplus y$	x y f 0 0 0 0 1 1 1 0 1 1 1 0

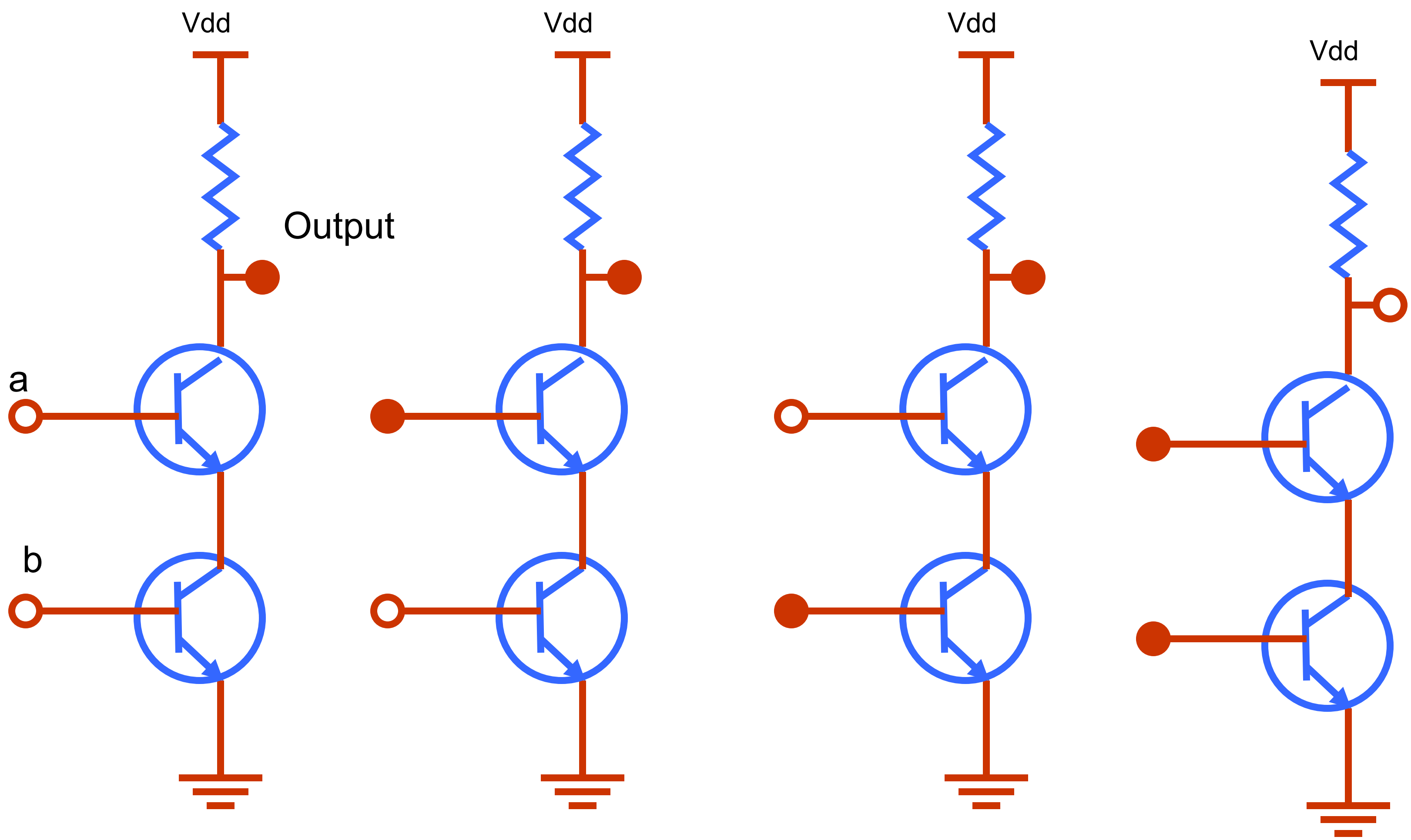
Abstraction Using Symbols

Gate	Schematic Symbol	Algebraic Function	Truth Table
NOT (Inverter)		$f = \overline{x}$	x f 0 1 1 0
NAND		$f = \overline{xy}$	x y f 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$f = \overline{x + y}$	x y f 0 0 1 0 1 0 1 0 0 1 1 0
XNOR (Equivalence)		$f = \overline{x \oplus y}$ $= x \odot y$	x y f 0 0 1 0 1 0 1 0 0 1 1 1

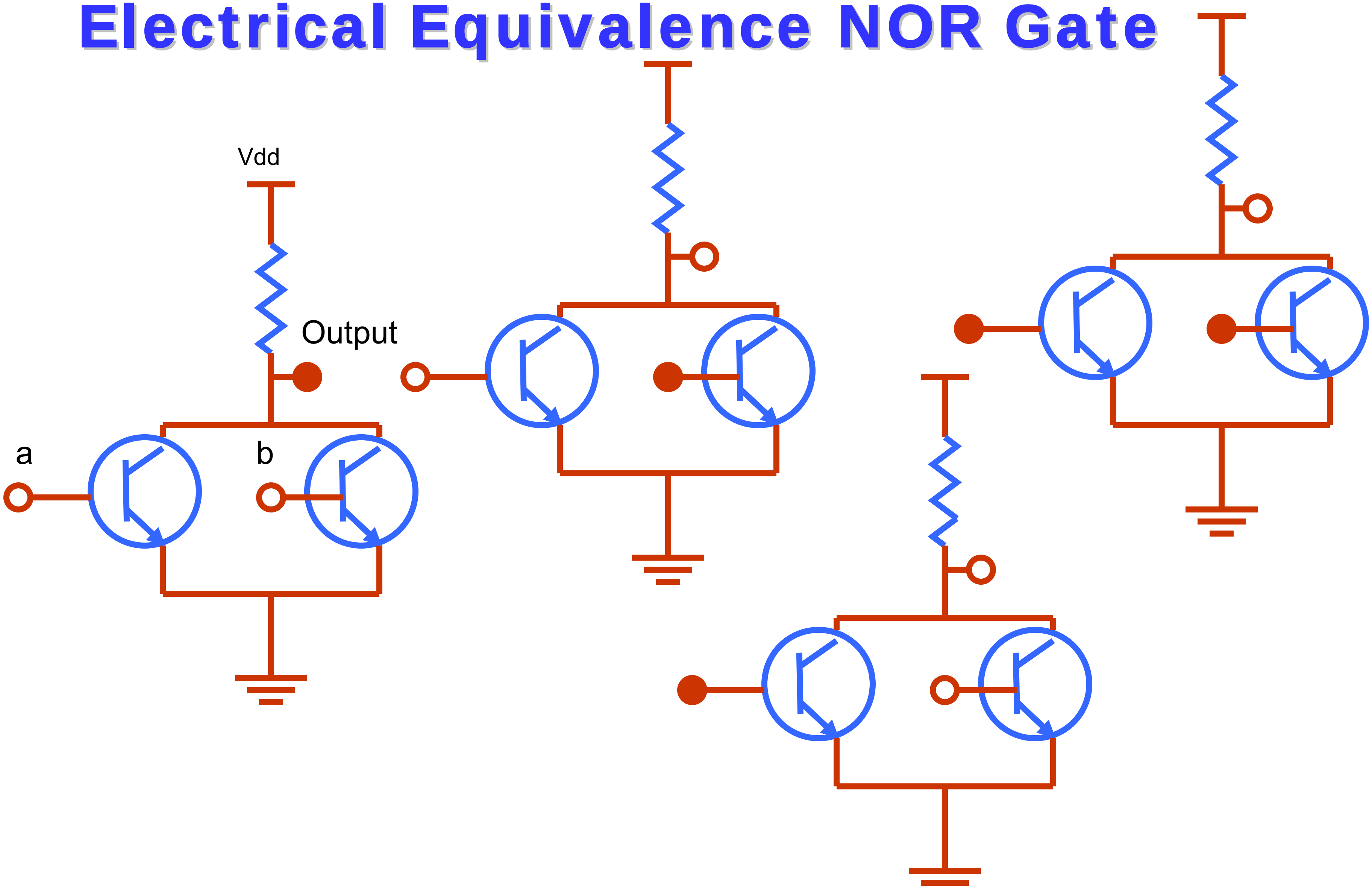
Electrical Equivalence NOT Gate (Inverter)



Electrical Equivalence NAND Gate

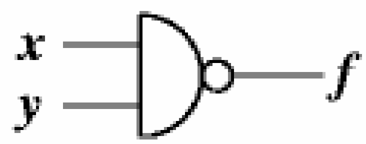


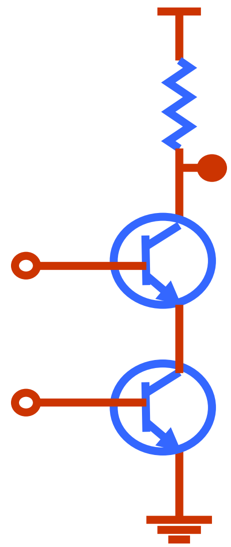
Electrical Equivalence NOR Gate



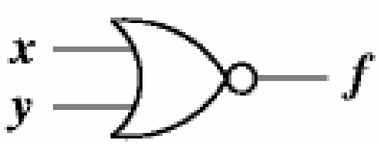
Equivalence

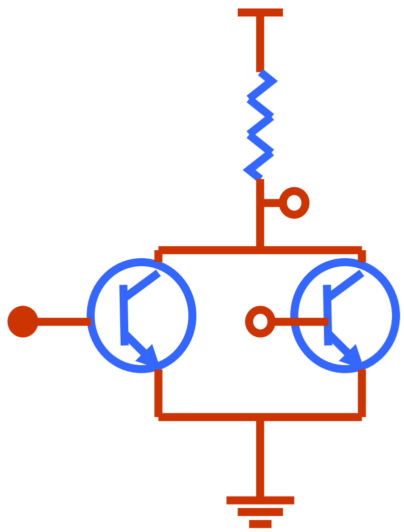
x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0





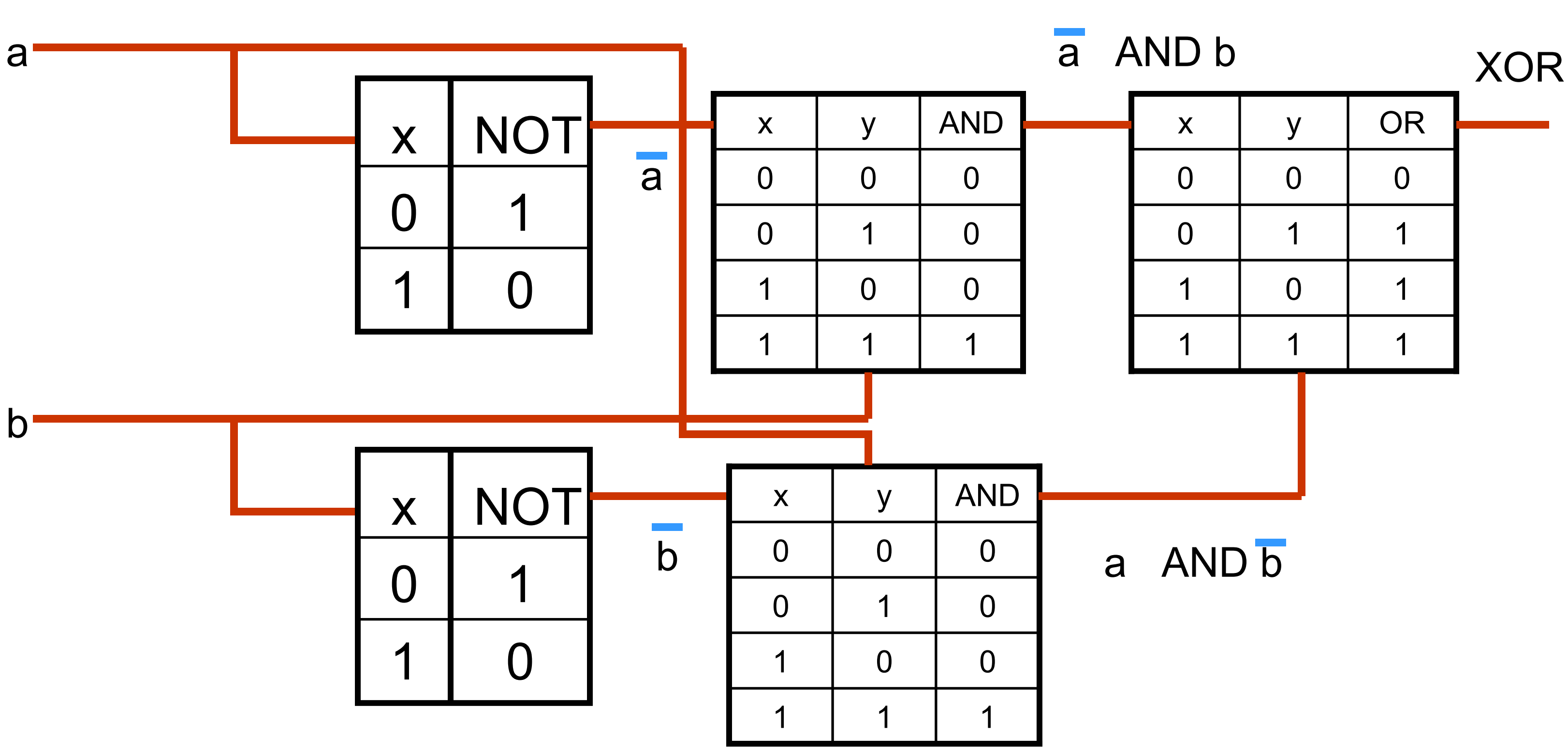
x	y	NOR
0	0	1
0	1	1
1	0	1
1	1	0



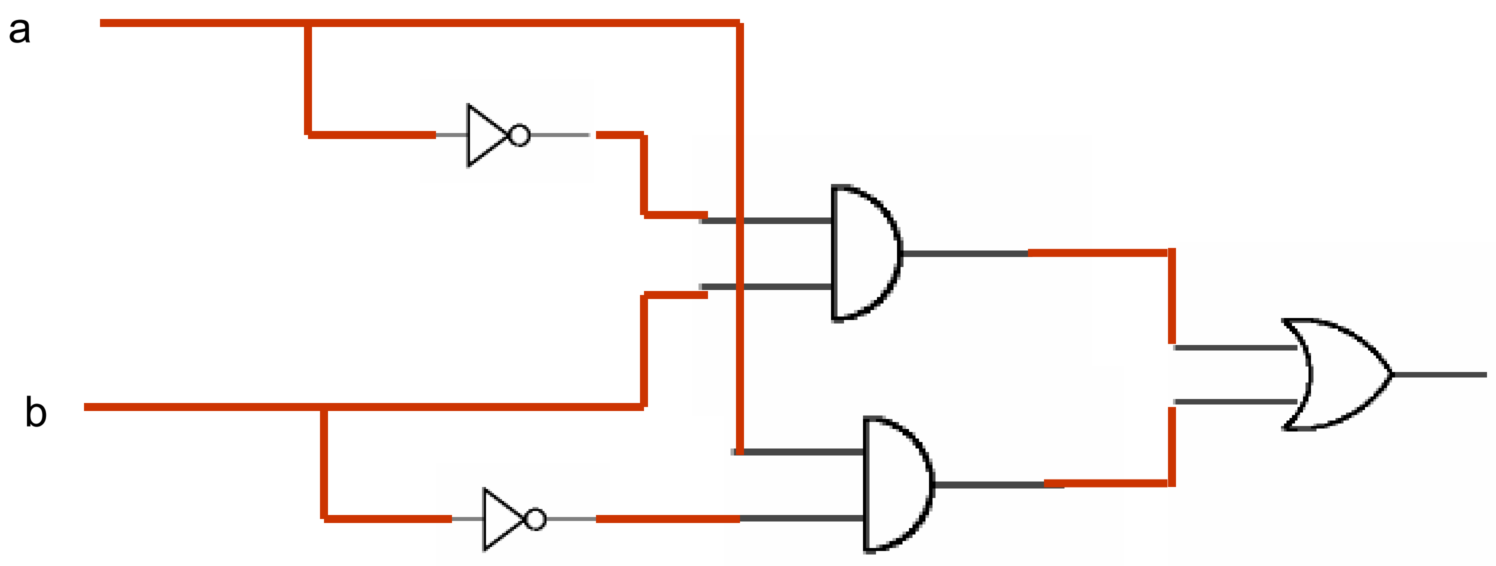


Let's Implement XOR

- XOR(a,b) = ($\bar{a}$ AND b) OR (a AND $\bar{b}$)



XOR Using Gates



Electrical Circuits

- Voltages, Currents
- Resistors, transistors etc.
- Well known device behaviors
  - Equations for current, voltage etc.
- Lots of details !!
  - Work out XOR with transistors
- Evaluation of a function
  - Simulate and see
  - Apply rules of the devices

Truth Tables

- Logical function
  - Abstraction
- Do not have to worry about implementation
  - All nasty transistor diagrams are gone
- Machine understandable
  - You can look up along the row and column for values
- Enumeration
  - Can get clumsy very easily

Gates

- Abstraction of Functions
  - Different from truth tables
- Gates and Truth tables are equivalent
  - One can be transformed to another
- The abstraction using gates is also useful
  - Easy to draw schematics than write truth tables
    - Or writing functions directly
  - A nice visual representation

To Ponder

- Why need different levels of abstraction?
- Hints:
  - Human vs Computer understandability
  - Scalability
    - Larger functions
    - More variables
  - Automation
  - Correctness of results
    - Can you verify?
    - How can you automate verifying?
- We will address some of the issues in the next class