Digital Logic & Boolean Algebra

Background Information

- Found in *Microcomputer Structures by Henry D'Angelo, 1981, pg 7-20*

Background Information - Charge & Electrons

- A charge is a physical quantity
  - Can be positive (a proton)
  - Can be negative (an electron)
- The unit of measure for a charge is a coulomb
- “Like” charges repel, “unlike” charges attract
  - Think poles on a magnet

Conductors vs. Insulators

Conductors:

- Allow electricity to flow more freely
- Better conductor better the flow of electricity
- Copper wiring is used because it very conductive
    - It is not the most conductive, that's silver

Insulators:

- Disallow (or resist) the flow of electricity
- Rubber or glass are good insulators

Voltage (V)

- Difference in electrical potential
- Work necessary to move a charged unit from point A to point B
- Difference in electrical pressure between two points
- Measured in Volts (V) or Joules/Coulomb

Current (I)

- Rate of charged motion
- Describes the flow of electrons
- Measured as Coulomb/Second (or Ampere)

Resistance (R), Resistors

- Opposition to the flow of electrons
- A way to reduce electrical flow
  - Similar concept to mechanical friction
- Resistors are the method through which resistance is added to electrical circuits

Circuits

- A closed path formed by interconnecting various electronic components
- Electric current can flow through the closed path
- See http://en.wikipedia.org/wiki/Electronic_circuit

Ohm’s Law

- Current through a conductor between two points defined as:
  - Directly proportional to the potential difference (Voltage)
  - Inversely proportional to the resistance between the points
- Mathematically given as:
  - `I = V / R` **or**
  - `V = I * R`
  - I is current, V is Voltage, R is Resistance
- See http://en.wikipedia.org/wiki/Ohm%27s_law

Short Circuit

- A different path *for current* than the intended path through a circuit
- Abbreviated to **short** or **s/c**
- See http://en.wikipedia.org/wiki/Short_circuit

Voltage Drop Across Resistors and Switches

- energy has to go somewhere
- limits the amount of energy that is dissipated
- a resistor converts energy into heat
- an LED is a diode
  - there is a constant drop of voltage (0.7V) across a diode
  - need to deal with the remaining 4.3V
    - a resistor dissipates it.

2655 Refresher

- Computers store numbers and only numbers
- Number represented as fixed-length binary value
  - Different representations available

How is a bit represented?

- A voltage level is used to represent a bit:
  - +0V (“low”)
  - +5V (“high”)
- “Low” might mean 0 (or false)
- “High” might mean 1 (or true)

Analog Circuit

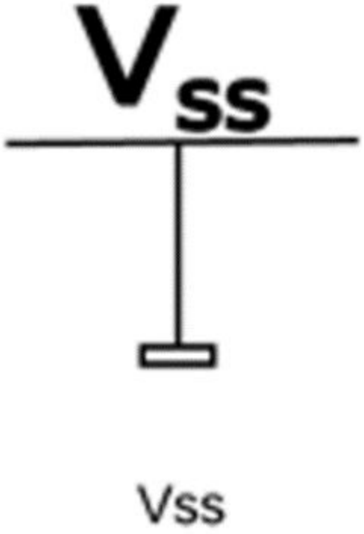
- Electronic circuit which processes *continuous* voltage levels
  - i.e. more than two distinct value
  - e.g. sound, light, temperature, pressure, position
  - Info translated from physical to electronic (e.g. microphone) **or**
  - Electronic to physical (e.g. speaker)

Digit Circuit

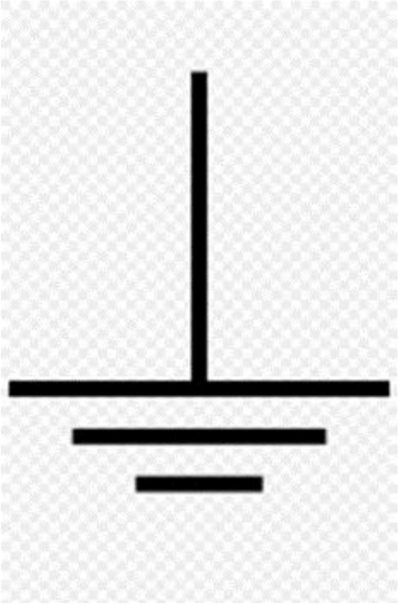
- Electronic circuit which processes *discrete* voltage levels
- The levels represent the values 0/1
- Built out of *logic gates*
  - and other components
  - small circuits which implement a logical operation

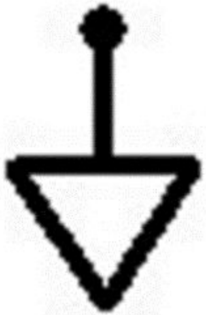
Schematic Diagram

Power



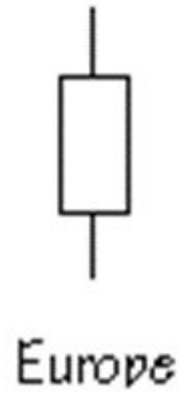
Ground





Schematic Diagram

Resistor - Europe

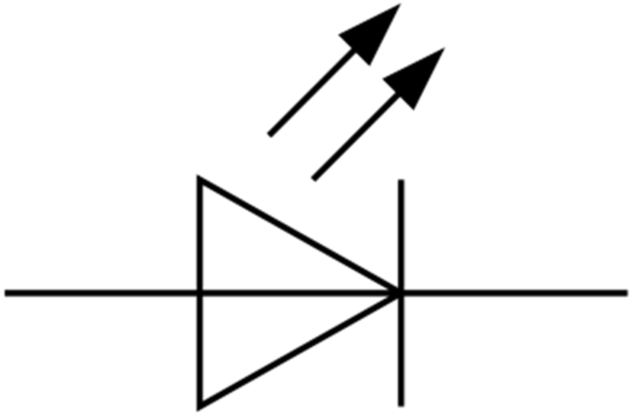


Resistor - USA, Japan

Schematic Diagram

Switch

LED



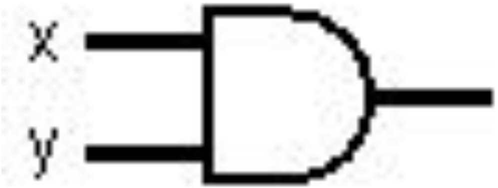
Logic Gates - Not

X	X’
0	1
1	0

Logic Gates - Buffer

X	X
0	0
1	1

Logic Gates - And

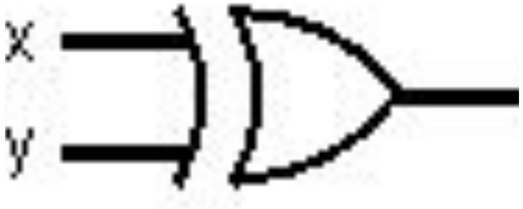


X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates - OR

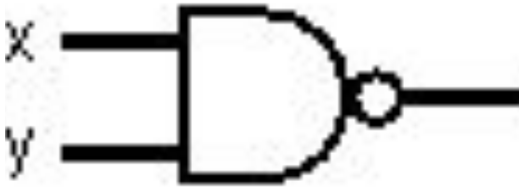
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates - XOR



X	Y	(X+Y)(XY)’
0	0	0
0	1	1
1	0	1
1	1	0

Logic Gates - NAND



X	Y	(XY)’
0	0	1
0	1	1
1	0	1
1	1	0

Logic Gates - NOR

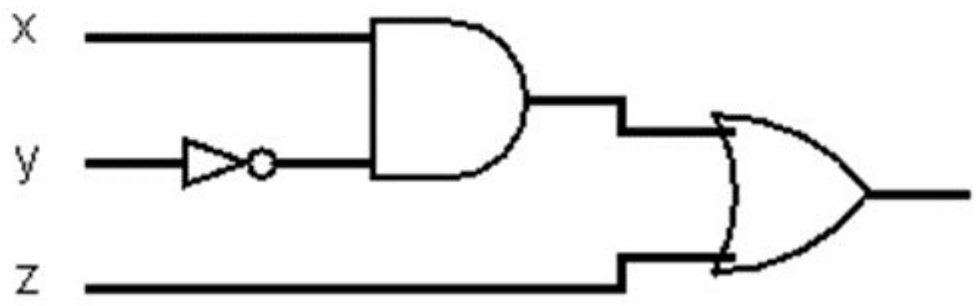
X	Y	(X+Y)’
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gates - XNOR

X	Y	X'Y' + XY
0	0	1
0	1	0
1	0	0
1	1	1

Digital Circuit ⟶ *"x and not y, or z"*

Note: Punctuation is important!!



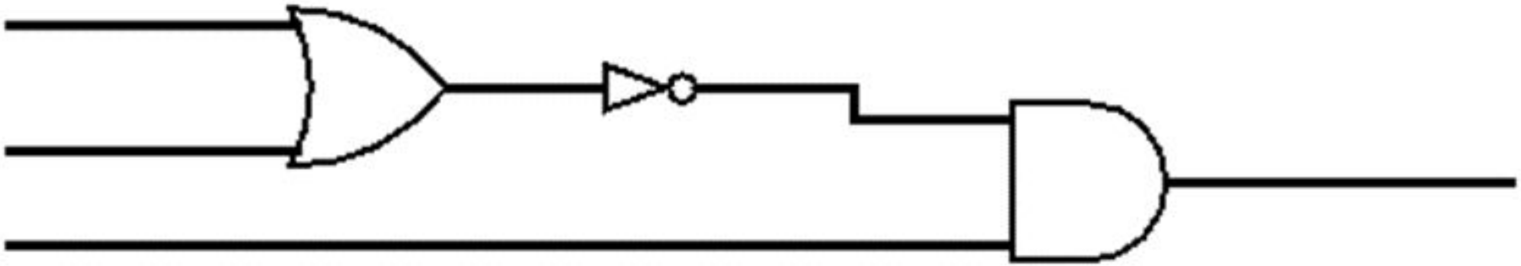
Digital Circuit - Exercise

Draw the logic diagrams for digital circuits that compute:

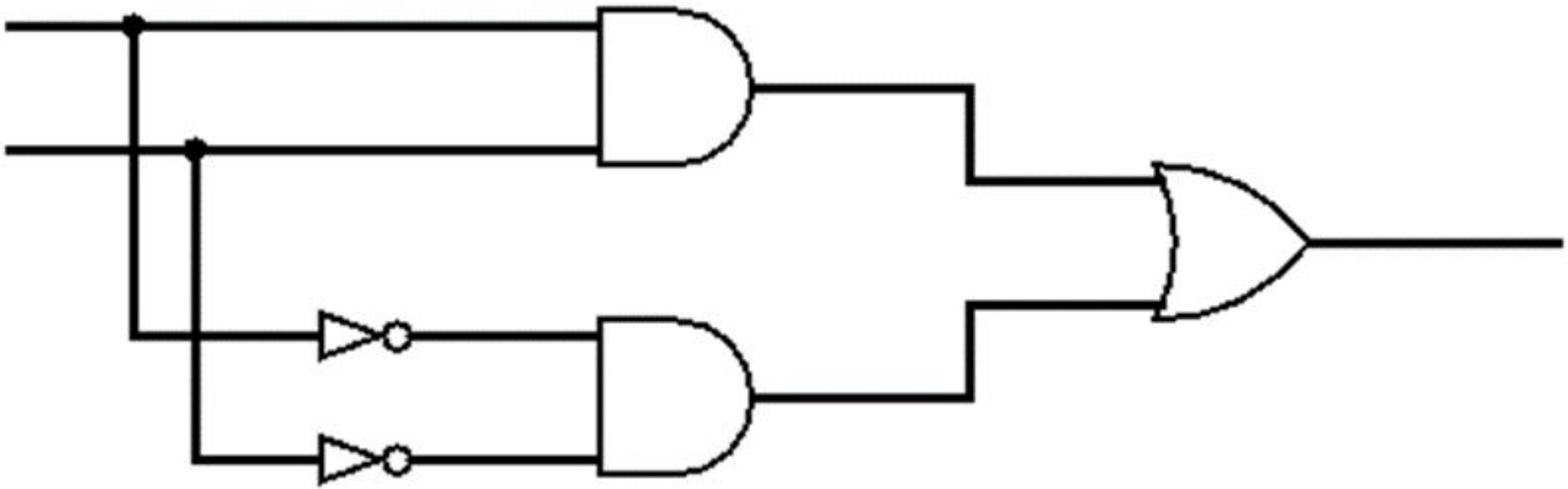
1. neither x nor y, and z
2. x and y are equal

Digital Circuit - neither x nor y, and z

Can be rewritten as: NOT (EITHER x OR y) AND z



Digital Circuit - x and y are equal



Modern Computers and Logic Gates

- Logic gates are built using transistors
- A transistor is a semiconductor component
  - Acts as an electrically controlled electrical switch
  - No moving parts
- Example: using CMOS technology
  - an inverter is built using two transistors
  - a 2-input NAND gate is using four transistors

Integrated Circuit (IC)

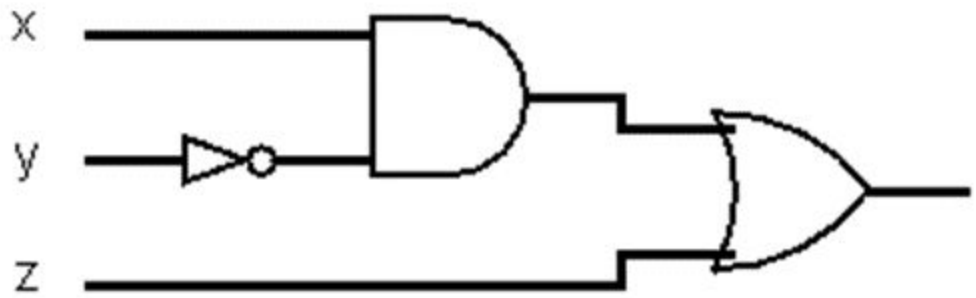
- Miniaturized electronic circuit
- Manufactured in thin layer of semiconductor material
  - Typically silicone
- A modern microprocessor may contain 100’s of millions of transistors
- Each chip (CPU or otherwise) has electrical contacts around the edges or the bottom, the chip is placed inside a plastic/ceramic package
  - Connects the contacts with the packages pins
- A chip can be used as a component in a larger digital circuit
  - i.e. soldered onto a circuit board

Boolean Algebra

- Captures the essential properties of the logic operations:
    - AND
    - OR
    - NOT
- Are able to write out complex circuits using the boolean algebra notation
- In CS boolean algebra is used to describe and reason about digital circuits

Boolean Algebra

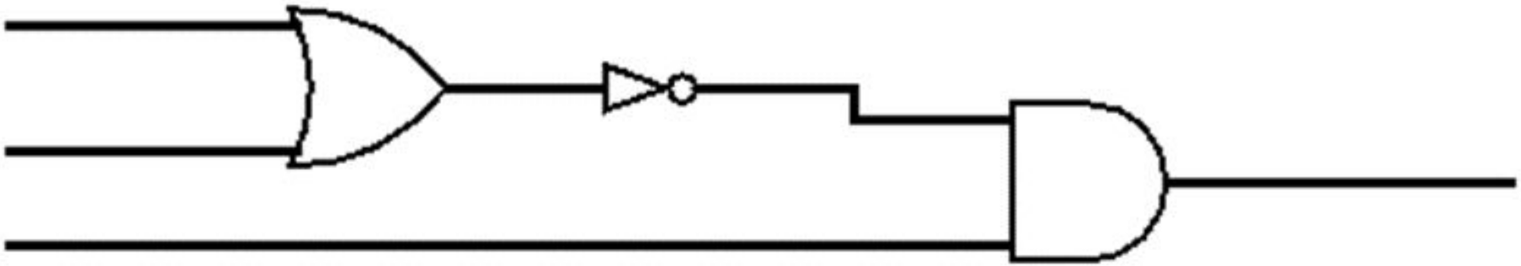
- The formula corresponding to the logical diagram below is:



- (x • (y’)) + z

Exercise 1 - Boolean Functions

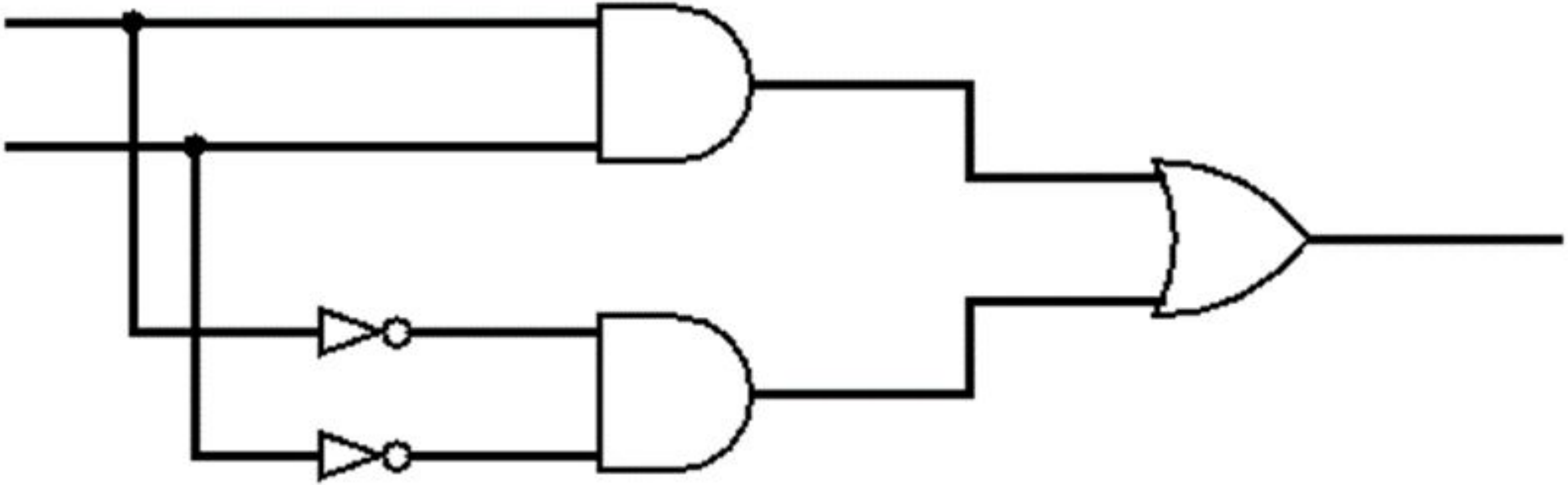
- Using the symbols above, write the formulas corresponding to:



- $(x + y)' \cdot z$

Exercise 1 - Boolean Functions

- Using the symbols above, write the formulas corresponding to:



- (x · y) + (x' · y')

Boolean Algebra

Consists of:

- a set A
- constants and operations:
  - `0: A`
  - `1: A`
  - `(•): A x A → A`
  - `(+): A x A → A`
  - `(‘): A → A`

Boolean Algebra

In a boolean algebra the following axioms must hold:

x • 1 = x	x + 0 = x	identities
x • y = y • x	x + y = y + x	commutativity
x•(y+z) = (x•y)+(x•z)	x+(y•z) = (x+y)•(x+z)	distributivity
x•x’ = 0	x+x’ = 1	complentativity

- Operator precedence, implied (·)
- In Distribution both symbol **and** operation are distributed

Example

- What does the following expression say:

  `xy' + z`

- It says the following:

x	y	z	y'	xy'	xy' + z
0	0	0	**1**	**0**	**0**
0	0	1	**1**	**0**	**1**
0	1	0	0	0	0
0	1	1	0	0	**1**
1	0	0	**1**	**1**	**1**
1	0	1	**1**	**1**	**1**
1	1	0	0	0	0
1	1	1	0	0	**1**

Boolean Algebra

- According to the definition, there can be many Boolean algebras!
- In this course, we are concerned with the **2-value Boolean algebra** {0, 1} with AND (·), OR (+) and NOT (‘)
    - as defined by the previous truth tables (on the logic gates handout).
- An alternative Boolean Algebra: given a set A its powerset and the operations union (AND), intersection (OR) and complement (NOT) is a Boolean algebra.

Exercise

- Prove the initially defined 2-value Boolean Algebra is a Boolean algebra
- This is done by showing that ALL of the axioms hold for the algebra
- Which is done using truth tables where you show that both sides of the axiom equations are equivalent
- Show that $x \cdot 1 = x$

	$\cdot$	0	1
x	0	0	0
	1	0	1

Boolean Algebra

Many theorems follow from those axioms:

$x \cdot x = x$	$x + x = x$	idempotency
$x \cdot 0 = 0$	$x + 1 = 1$	boundedness
$x \cdot (x+y) = x$	$x + xy = x$	absorption
$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	$x+(y+z) = (x+y)+z$	associativity
$(x \cdot y)' = x' + y'$	$(x+y)' = x' \cdot y'$	De Morgan's Law
$0' = 1$	$1' = 0$	0,1 are comp's
$(x')' = x$		involution

Boolean Algebra

- idempotency – property of operations that yield the same result after the operation is applied numerous times.
- boundedness - a distinct and knowable upper and lower bound

Exercise: Prove Idempotency

- Show that x•x = x

- $$\begin{aligned} x \cdot x &= x \cdot x + 0 && \text{(identity)} \\ &= x \cdot x + x \cdot x' && \text{(complements)} \\ &= x \cdot (x + x') && \text{(distributivity)} \\ &= x \cdot 1 && \text{(complements)} \\ &= x && \text{(identity)} \end{aligned}$$

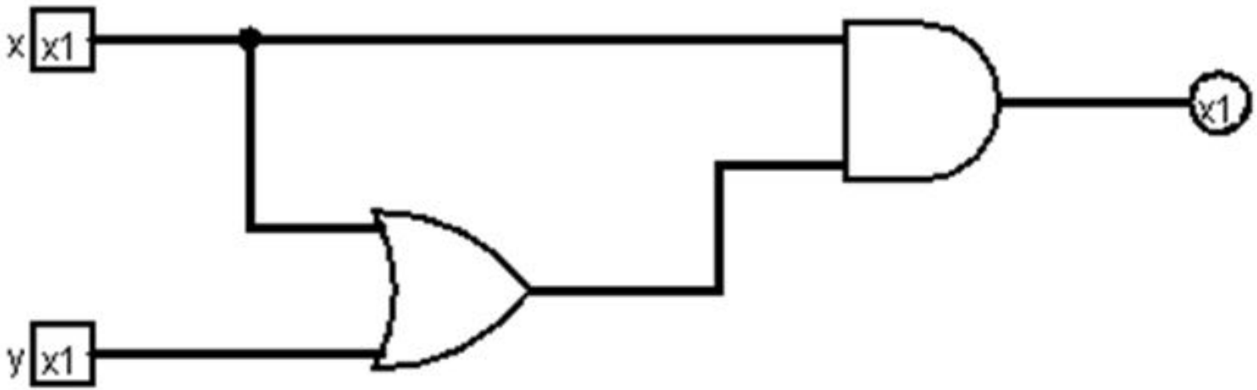
- This proves half of the idempotency theorem
  - The second proof is analogous

Boolean Algebra

- Note the duality of the axioms and theorems.
  - Obtained by swapping (·) and (+) as well as 0 and 1
- A theorem is true if and only if its dual is true (this is the duality meta-theorem)
- How can we use associativity to further avoid explicit parenthesization?
  - Since the result is identical regardless of the way the terms are grouped the parenthesis can be eliminated

Exercise

- choose any appropriate axioms or theorems, and show how they can be used to simplify the following logic diagrams



- Which is f = x(x+y)
- By absorption is becomes f=x

Boolean Algebra

- The theorems can be generalized to include more than two inputs
- Example:
  - The first idempotency theorem (xx=x) can generalized to (xx...x)=x
  - How would you prove this?
- It is useful to have generalization of other theorems/axioms
  - Distributivity
  - De Morgan's Law

Proof of Generalized Idempotency Theorem

Use Induction

- Given xx=x is true
- Assume that by applying the idempotency theorem in reverse (x=xx) grows xx to xx…x (with n x’s)
- Take any of the x’s in the sequence and apply the idempotency theorem to it (x=xx)
    - This replaces the one with x with two, making the sequence n+1 x’s in length

Note: You can apply induction similarly to “shrink” a sequence of x’s as well

Boolean Algebra Formulas

- A Boolean expression can be considered a function, where the variables represent the inputs
  - These functions are assigned names
  - Example: `f = xy' + z`

Forms of Boolean Algebra Formulas

- There are two ways in which an expression can be listed
  - SOP - Sum of Products
  - POS - Product of Sums
- These two methods are considered **standard** forms
- The function on the previous slide is in SOP
- All functions in SOP can also be written in POS

Forms of Boolean Algebra Formulas

Consider the following table:

- minterms are the product (AND) of the specified values
- maxterms are the dual of the minterms
  - and visa versa

x	y	z	minterms	maxterms
0	0	0	x'y'z'	x+y+z
0	0	1	x'y'z	x+y+z'
0	1	0	x'yz'	x+y'+z
0	1	1	x'yz	x+y'+z'
1	0	0	xy'z'	x'+y+z
1	0	1	xy'z	x'+y+z'
1	1	0	xyz'	x'+y'+z
1	1	1	xyz	x'+y'+z'

Truth Tables and Boolean Algebras

- A truth table for a function can be used to generate its Boolean function
    - sum all the minterms when the function is 1 (gives SOP form)
    - multiply all the maxterms where the function is 0 (gives POS form)
- These are called the "canonical" forms. It may be possible to simplify them (e.g. algebraically)
- In "canonical" form all of the terms in SOP/POS use **ALL** variables

Digital Circuit Design With Boolean Algebra

- Design a digital circuit which implements the following Boolean function, as specified by the truth table
- This by summing the minterms, this yields:
  - `f = x'y'z + xy'z' + xy'z`

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Digital Circuit Design With Boolean Algebra

- `f = x’y’z + xy’z’ + xy’z`
- This can be directly implemented using 5 inverters (really only 3 are needed), 3 3-input AND gates and 1 3-input OR gate.
- Can it be simplified so it uses a smaller number of gates? **YES**
- `f = x’y’z +` **`xy’`**`z’ +` **`xy’`**`zLook for duplicate parts`
- `f = x’y’z + xy’(z’ + z)      (z’ + z) = 1 and 1 x = x`
- `f = x’y’z + xy’             =` **`(x’z + x)y’`**
- Now down to 2 invertors, 1 3-input AND, 1 2-input AND and 1 2-input OR can also change to 2 invertors, 2 2-input AND and 1 2-input OR

Boolean Algebra

- It may be possible to algebraically simplify an expression
- If the expression remains completely as SOP and POS it is called in "standard" form.
- Otherwise it is simply an expression.

SOP vs POS - Which is better?

- Since they are duals they are equivalent!
- “Better” depends on what purpose the form is being used for.
- For purposes of expressing
    - whichever is simpler, i.e. has the fewer terms
- For purposes of implementing
    - again this depends on the gates being used
- With NAND gates → SOP is better to use
- With NOR gates → POS is better to use

Boolean Algebra - Completeness

- From this discussion it should be clear that any Boolean function can be implemented out of NOT, AND and OR gates (even if they are only 2-input AND and OR gates).
- For this reason, the set of these three operations is said to be complete.
- A set of operations/gates is said to be complete if any Boolean function/circuit can be implemented using only combinations of (or, “by composing”) the operations / gates in the set.

Boolean Operations

- how many 1- and 2-input Boolean operations are there?
- 4 1-input op   but we are only interested in NOT
  - identity (buffer),
  - not,
  - constant 0,
  - constant 1
- 16 2-input ops - only interested in AND, OR, XOR, NAND, NOR, XNOR
- see gre_bol1.pdf – output patterns

Boolean Algebra - Completeness

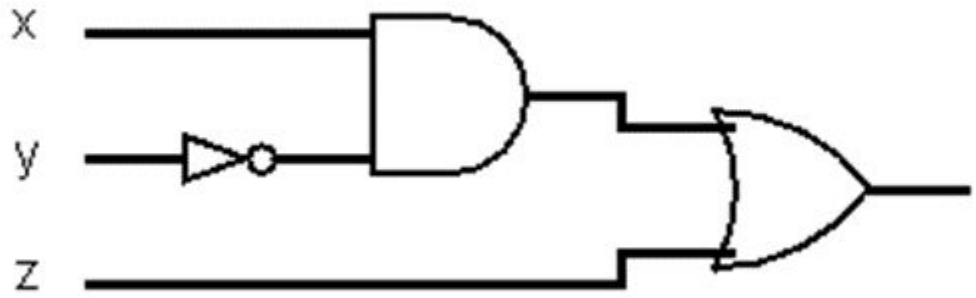
- Are there other complete sets of operations?
  - how about just NOT and AND?
  - how about just NOT and OR?
  - how about just NAND?
  - how about just NOR?
- Any Boolean function can be implemented just out of 2-input NAND gates!

Boolean Algebra - NAND Completeness

- The following can be proven via truth tables
  - NOT y = y NAND y
  - x AND y = (x NAND y) NAND (x NAND y)
    - 2 gates since bracketed terms the same
  - x OR y = (x NAND x) NAND (y NAND y) → x NAND y
- Since NAND gates are easy to build out of transistors, this is useful

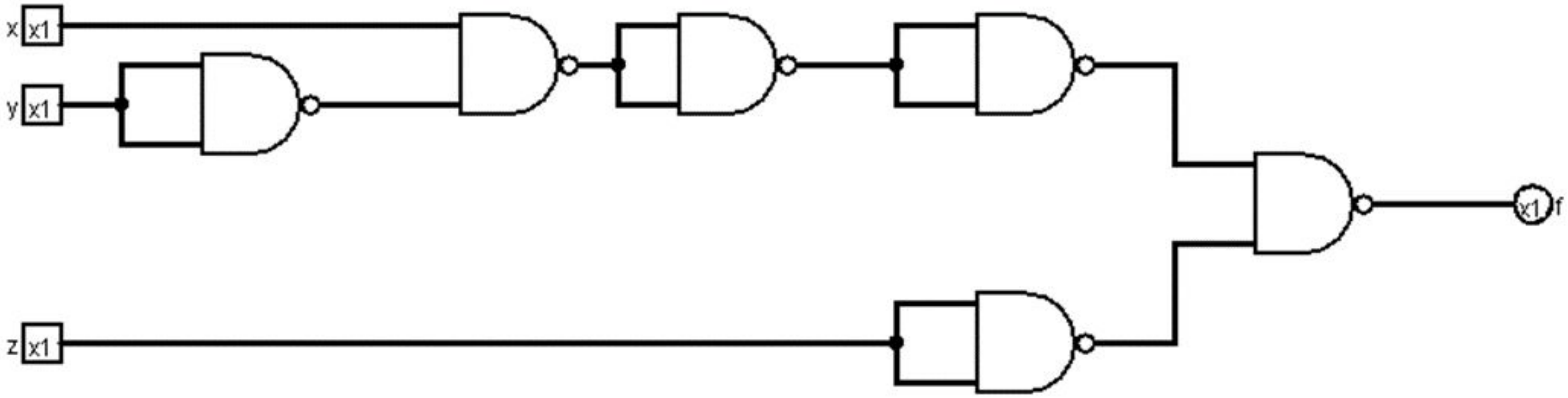
Exercise - Implement with NAND gates only

- Implement `f = xy' + z` out of just NAND gates
- Remember the basic circuit for the function is:



Method 1

- Each of the gates in the above circuit can be replaced with the corresponding NAND implementation
- But this requires remembering the AND, OR and NOT NAND implementations



Is there a simpler version

- While the above version a valid circuit it takes a lot of gates.
- Is there a simpler version? **YES**
- Truth table for the function is to the left

x	y	z	y’	xy’	xy’+z
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	1

Method 2

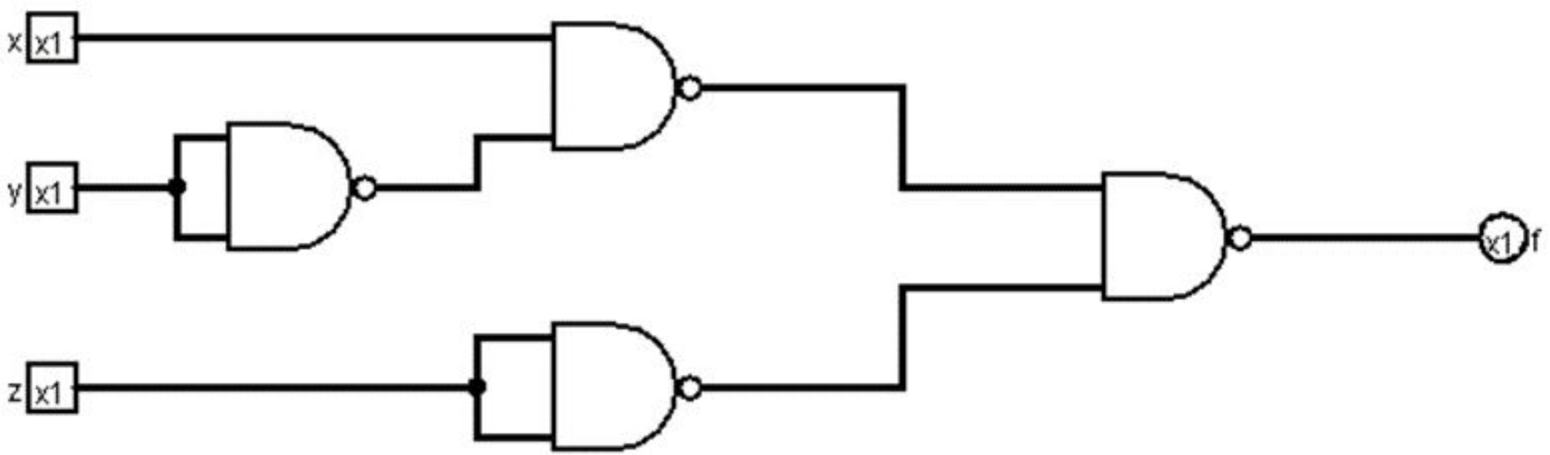
- Rewrite the functions as $f = A + B$
- Where $A = xy'$ and $B = z$
- Involution says we can apply NOT twice and get the same results
  - $f = ((A + B)')'$
- then by DeMorgan's
  - $(A+B)' = A'B'$
- So the original function becomes:
  - $f = (A'B')'$

Method 2

- Notice that this is only using an AND.
- This can be interpreted as an AND with the inputs negated and then the output negated.
- Thus, an OR can be converted to an AND with the inputs and outputs negated
- Now A is the AND of x and y'. Whether the NOT is done after this AND or before the above AND makes no difference.

Method 2

- Thus, this circuit can be implemented as:

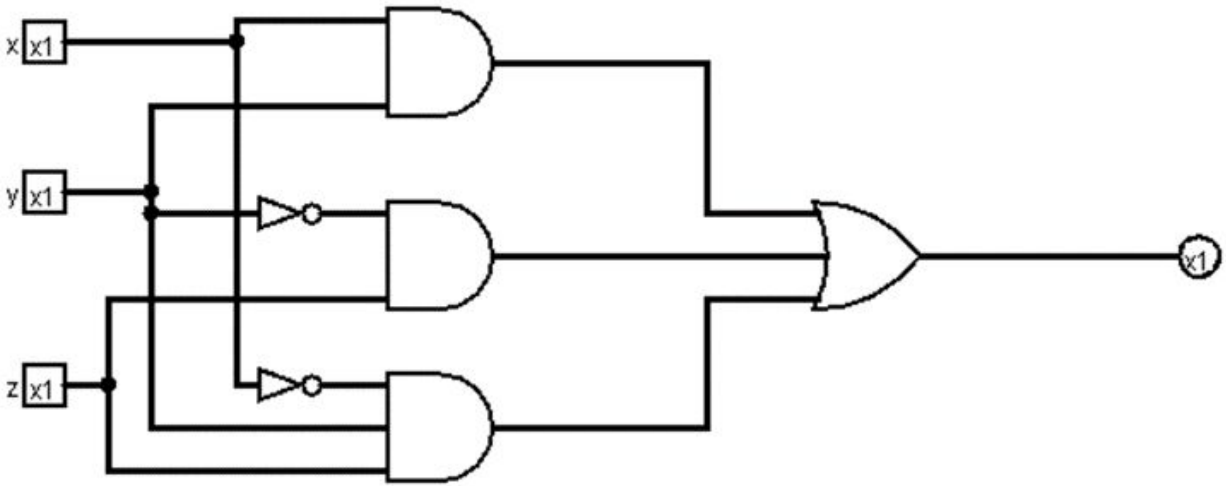


Boolean Algebra

- Thus, any expression in SOP form can be easily translated to an all-NAND circuit. Similarly, any expression in POS form can be easily translated to all-NOR
- Conversion from SOP to all NAND is done by:
- Applying involution to the function in SOP form
  - Then applying DeMorgan's to the inter expression
- An alternative way of viewing this is to negate both the output of AND gates and the inputs to the OR gate
  - DeMorgan's law shows that NOT OR is equivalent to NAND

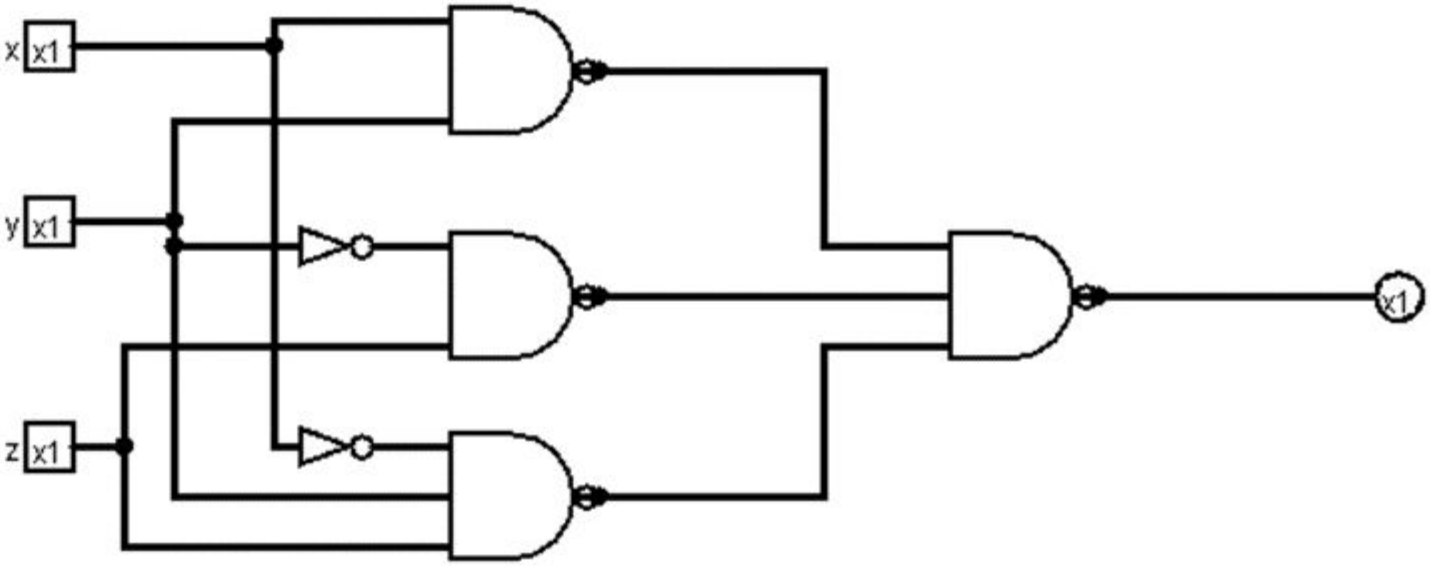
Example

- Create the all NAND circuit for
    - f = xy + yz + xyz
- This expression in normal SOP form



Example Solution

- Apply one of the above methods and the circuit becomes:



Grey Code

- Is a sequence of bit patterns that meet the following criteria
  - All the patterns are n-bits long for some value of n
  - Each pattern must differ from the previous by exactly one bit
  - All possible n-bit patterns must be included in the grey code
  - No bit pattern can be repeated in the code

```
2-bit Grey Code:00
                01
                11
                10
```

Summary

- Boolean algebra is used to reason about digital logic. These activities are common:
    - proof of equality
    - algebraic simplification