Introduction to Digital Logic

Lecture 7:

Design Goals (Metrics)

2-Level Logic

Negative Logic

Warmup

- Consider F(w,x,y,z). Show the algebraic form of m4 and M4?
- Use Boolean algebra to find the minimal SOP expression for the function, $F = \Sigma_{a,b,s}(3,4,6,7)$
- Use Boolean algebra to find the minimal POS expression for the function, $G = \Pi_{x,y,z}(1,3,5,7)$

Check Up

- The smaller the product term (i.e. fewer literals) the (**more / less**) minterms it covers
- (**True / False**) "Converting to SOP/POS" means convert to a canonical sum/product and add in missing variables
  - FALSE: SOP/POS is just a form…a canonical sum is the biggest SOP expression of a function while a canonical product is the biggest POS expression
    - SOP and POS expressions can be minimized using theorems, etc.
    - SOP and POS expressions can be expanded by adding in missing variables

Logic Exercise 1

- Tom and his three friends, Alice, Bob, and Charlie have been invited to a party. But Tom is an introvert and will only go if only a single one of his friends also goes. Express this as a logic function
  - T = A + B + C
  - T = A • B • C
  - T = AB'C' + A'BC' + A'B'C
  - T = B'C' + A'C' + A'B'

Logic Exercise 1

- Tom and his three friends, Alice, Bob, and Charlie have been invited to a party. But Tom is an introvert and will only go if only a single one of his friends also goes. Express this as a logic function
  - T = A + B + C (*at least one friend goes)*
  - T = A • B • C *(all friends go)*
  - T = AB'C' + A'BC' + A'B'C *(exactly one friend)*
  - T = B'C' + A'C' + A'B' *(0 or 1 friend goes)*

Logic Exercise 2

- Jill is also friends with Alice, Bob, and Charlie and has also been invited. Jill is shy and will only go if at least two of her other friends go. Express this as a logic function
  - J = AB + BC + AC
  - J = ABC' + AB'C + A'BC
  - J = A' + B' + C'
  - J = A' • B' • C'

Logic Exercise 2

- Jill is also friends with Alice, Bob, and Charlie and has also been invited. Jill is shy and will only go if at least two of her other friends go. Express this as a logic function
  - J = AB + BC + AC *(2 or 3 friends go)*
  - J = ABC' + AB'C + A'BC *(exactly two go)*
  - J = A' + B' + C' *(not everyone goes)*
  - J = A' • B' • C' *(no friends go)*

Covering Combinations 1

- An alien lands on earth and watches college students to try to learn their behavior, picking one student Bill as his subject. The alien watches Bill's class attendance behavior over several days and notices:
  - Day 1: Bill attends, a HW is due, it is sunny
  - Day 2: Bill attends, a HW is due, it is cloudy
  - Day 3: Bill doesn't attend, a HW is not due, it is sunny
  - Day 4: Bill doesn't attend, a HW is not due, it is cloudy
- What is a conclusion the alien can draw about Bill's attendance
- Bill attends if a HW is due
  - If the same result occurs for all the values of a variable, that variable must not affect the result

Covering Combinations 2

- The alien continues to watch and records Bill's attendance again based on a HW being due, whether it is sunny or not, and whether Bill sets his alarm the night before.
  - Help our alien friend derive the simplest expression for Bill's attendance
  - Find a pattern in a subset of input combinations such that the output is true wherever that pattern holds and the output is false wherever that pattern does not hold.
  - Make sure some pattern you found accounts for all locations in the truth table where the output is true

H	S	A	B
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Covering Combinations 2

- The alien continues to watch and records Bill's attendance again based on a HW being due, whether it is sunny or not, and whether Bill sets his alarm the night before.
  - Help our alien friend derive the simplest expression for Bill's attendance
  - Find a pattern in a subset of input combinations such that the output is true wherever that pattern holds and the output is false wherever that pattern does not hold.
  - Make sure some pattern you found accounts for all locations in the truth table where the output is true
- B = H + SA

H	S	A	B
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Covering Combinations 2

- The alien continues to watch and records Bill's attendance again based on a HW being due, whether it is sunny or not, and whether Bill sets his alarm the night before.
  - Help our alien friend derive the simplest expression for Bill's attendance
  - Find a pattern in a subset of input combinations such that the output is true wherever that pattern holds and the output is false wherever that pattern does not hold.
  - Make sure some pattern you found accounts for all locations in the truth table where the output is true
- B = H + SA
- Each term 'covers' combinations of variables not present in the term

H	S	A	B
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Speed, area, and power

DIGITAL DESIGN GOALS

Digital Design Goals

- When designing a circuit, we want to optimize for the following three things:
  - Area or Circuit Size (minimize)
  - Speed (maximize) / Delay (minimize)
  - Power (minimize)
- Can usually only optimize 2 of the 3
  - There is a huge trade space! This is what engineering is all about!

Minimizing Circuit Area

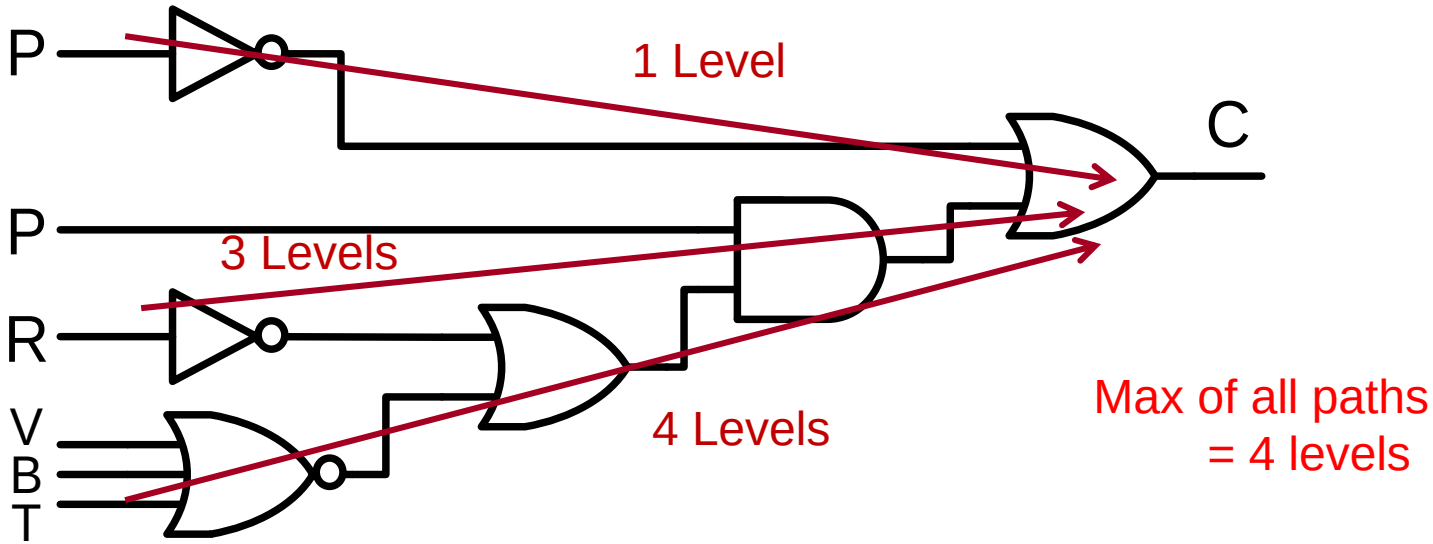
- Approaches:
  - Reduce the number of gates used to implement a circuit
  - Reduce the number of inputs to each gate
    - In general a gate with n inputs requires 2n transistors to implement
- Simplify logic expressions (usually by factoring and then canceling terms) to reduce the number of gates

Maximizing Speed

- Speed is affected by:
  - Levels of logic (path length)
  - Gate type
  - Number of inputs (fan-in) to the gate
  - Number of outputs a gate connects to (fan-out)
  - Feature size and implementation technology

Levels of Logic

- Definition: Maximum number of gates [not including inverters] on any path from an input to the output

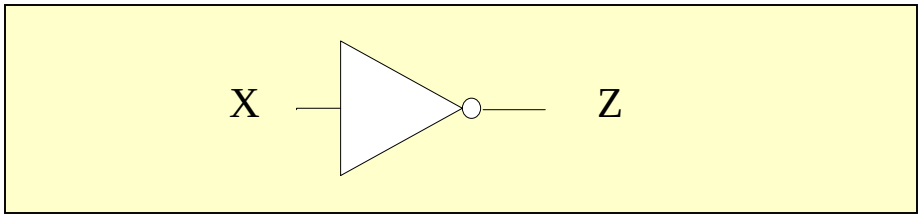


$$C = \overline{P} + P(\overline{(V+B+T)}+\overline{R})$$

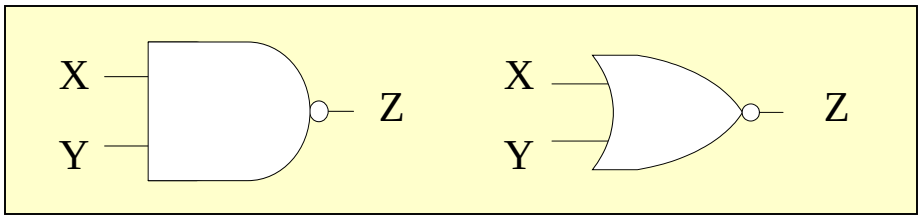
Gate Delays

- Order the gate types in terms of fastest to slowest?
- Typical gate delay for a 2-input NAND or NOR is under a 100 ps.

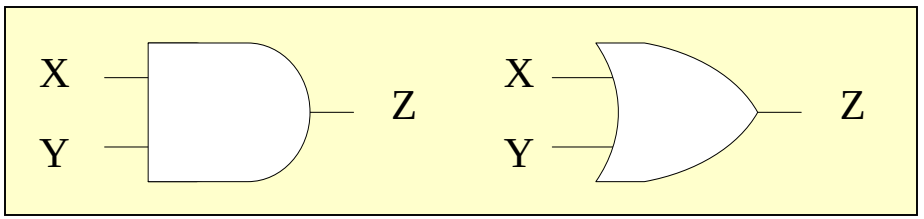
1



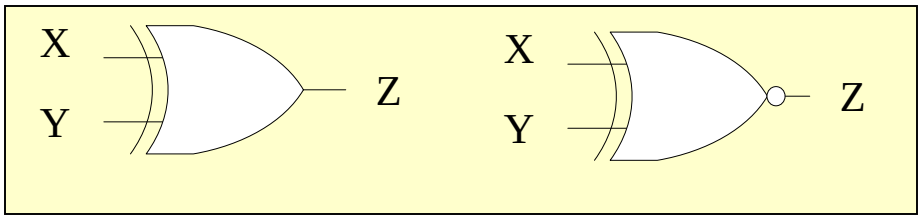
2



3



4



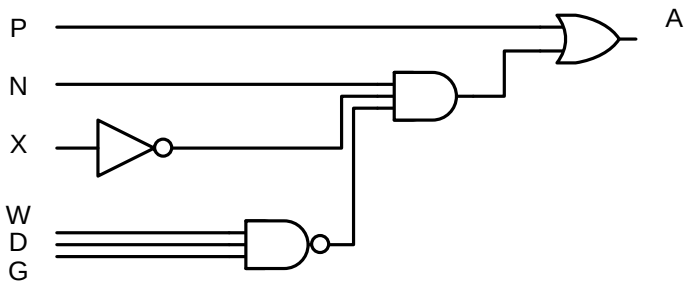
Digital Design Goals

- When designing a circuit, we want to optimize for the following three things:
  - Area (minimize)
    - Use fewer number of gates
    - Use gates w/ fewer inputs
  - Speed (maximize) / Delay (minimize)
    - Fewer levels of logic
      - Levels of logic = max. # of gates on a path from ANY input to output
    - Relative speed of gates: INV, NAND/NOR, AND/OR, XOR/XNOR
  - Power (minimize)
    - How much energy the circuit consumes when switching between 0 and 1
- Can usually only optimize 2 of the 3

2-LEVEL IMPLEMENTATIONS

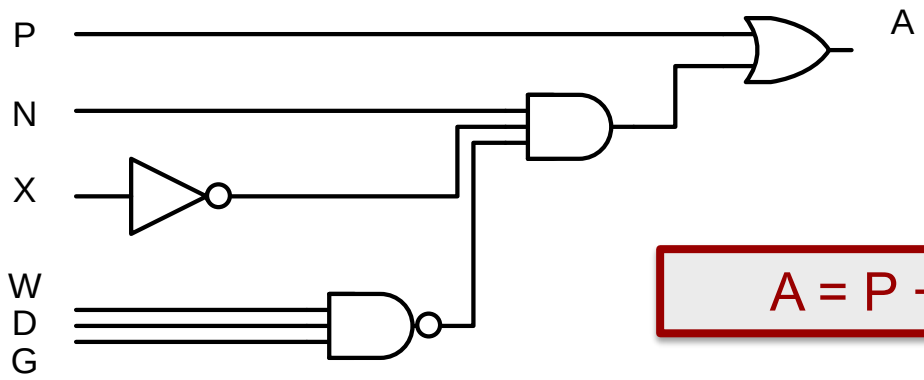
How Many Levels?

- Consider our the alarm circuit with an additional panic input:
  - The alarm should sound if the panic button is true or if the system is enabled, you are not exiting, and the house is not secure. The house is considered secure if the windows, door, and garage sensors are all true



- How many levels of logic is this circuit?
  - 3
- Can we reduce to 2?
  - Yes…Any circuit can be converted to a 2-level implementation
  - How? By converting to SOP or POS.

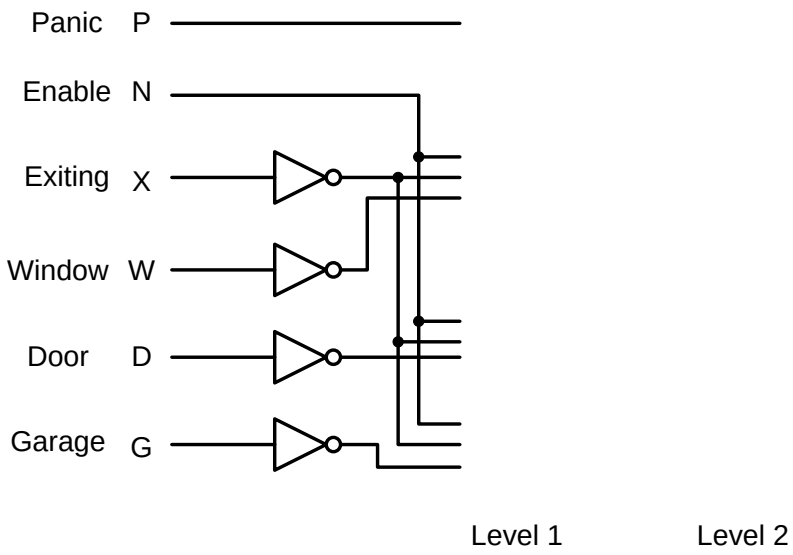
Converting to 2-Levels



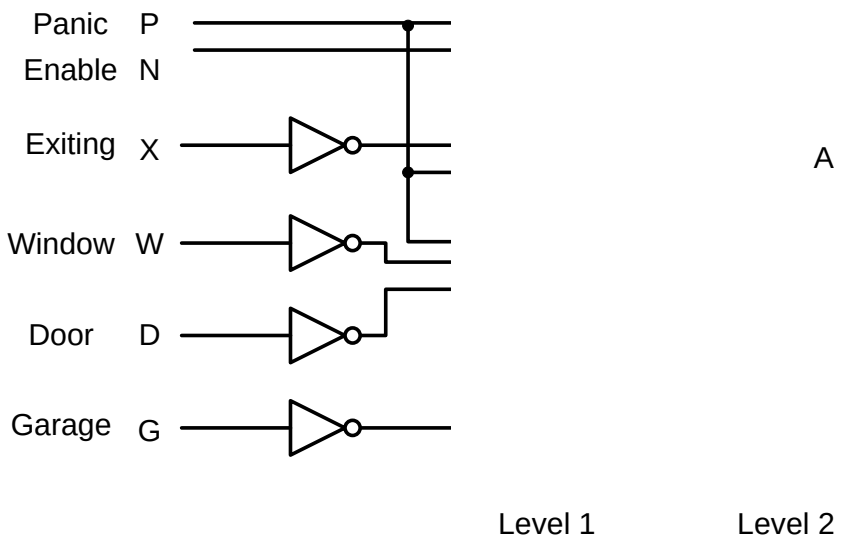
A = P + NX'(WDG)'

- To SOP:
- To POS:

- SOP => AND-OR implementation

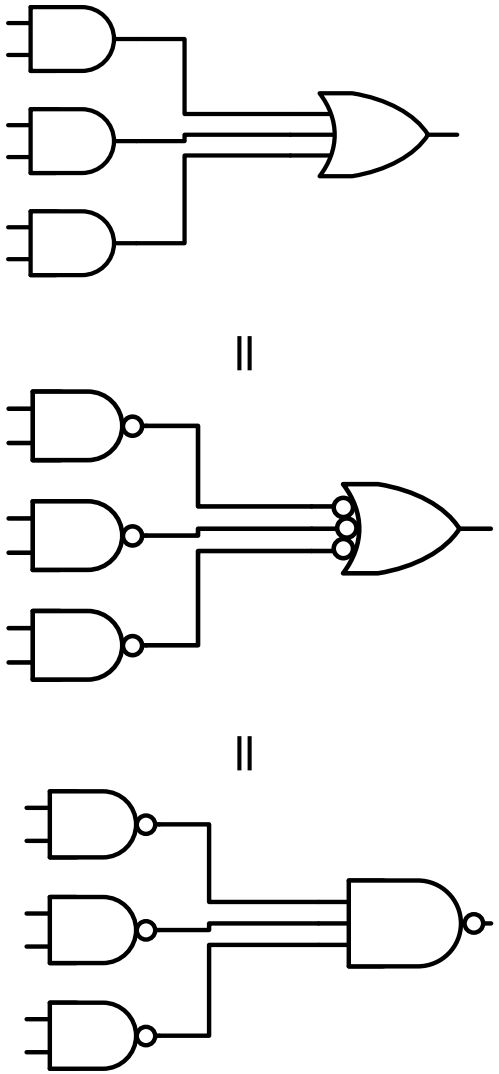


- POS => OR-AND implementation



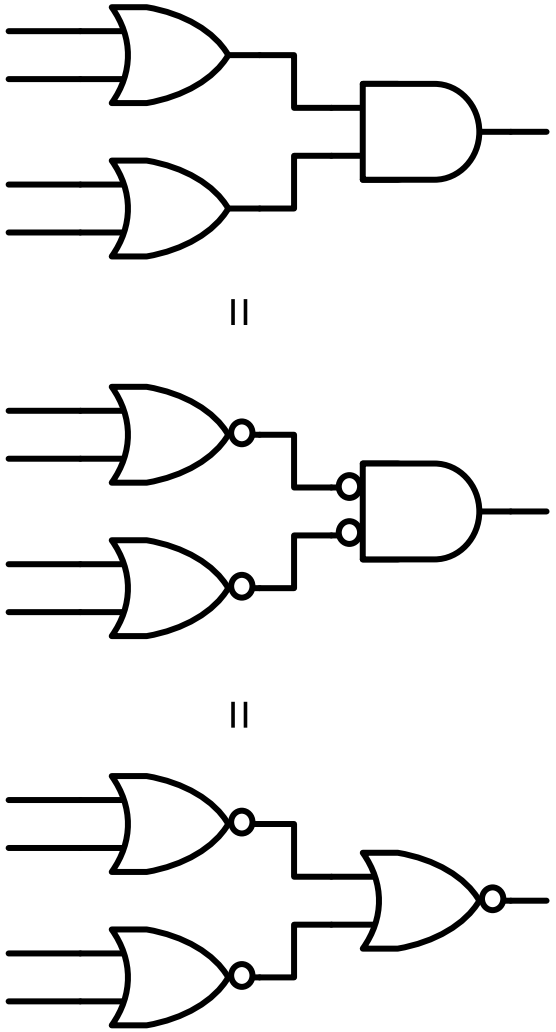
AND-OR / NAND-NAND

- SOP =>
  - AND-OR Implementation
  - NAND-NAND Implementation



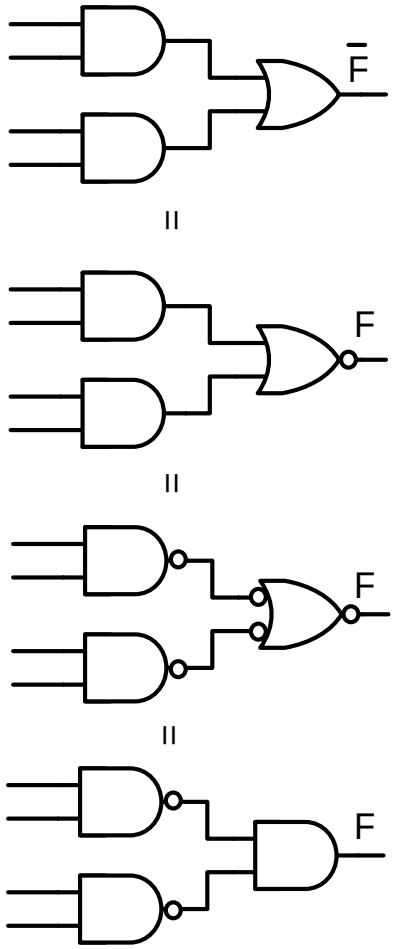
OR-AND / NOR-NOR

- POS =>
  - OR-AND Implementation
  - NOR-NOR Implementation



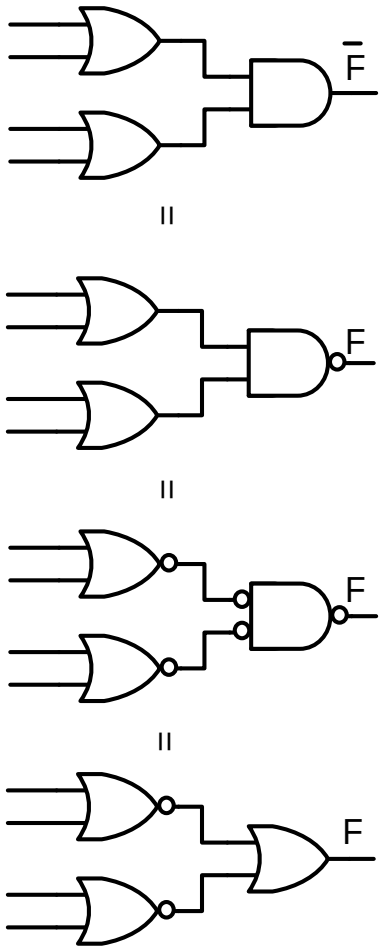
AND-NOR / NAND-AND

- Suppose we wanted to implement a function with AND-NOR
  - Implement F' via SOP, AND-OR
  - Change last OR gate to NOR and you will have F.
  - AND-NOR can then be converted by NAND-AND



OR-NAND / NOR-OR

- Suppose we wanted to implement a function with OR-NAND
  - Implement F' via POS, OR-AND
  - Change last AND gate to NAND and you will have F.
  - OR-NAND can then be converted by NOR-OR

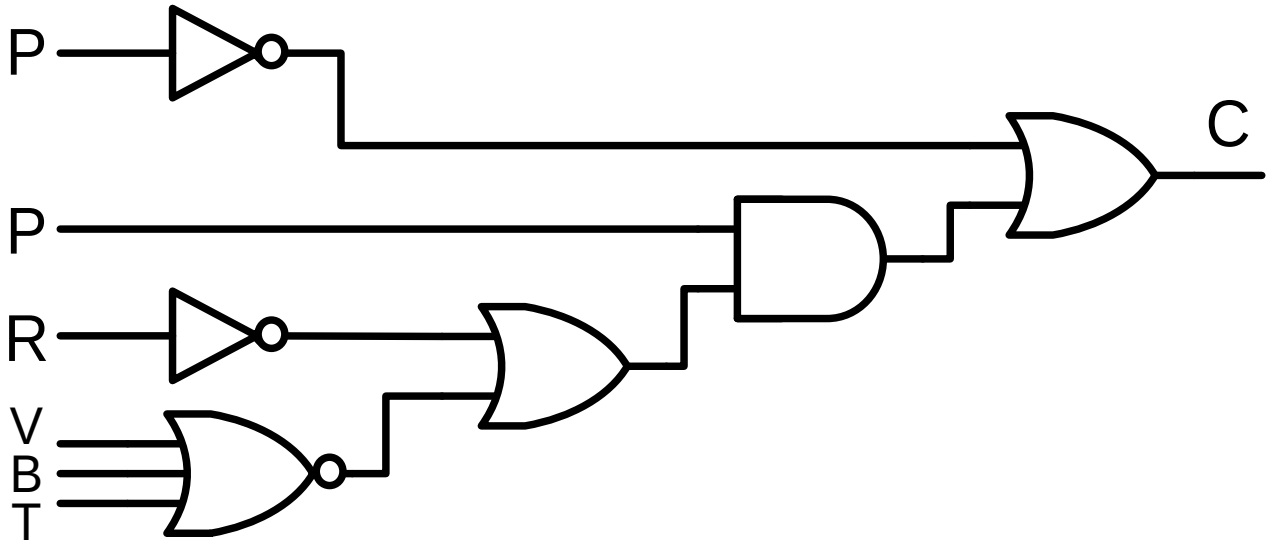


Circuit Design

- Take a problem description and create a circuit
  - "A car stereo display should show the **c**lock time (as opposed to other information) when the **p**ower is off or when the **p**ower is on and the **s**ound settings or **r**adio station are not being changed. The sound settings are considered as the **v**olume, **b**alance and **t**reble/bass."

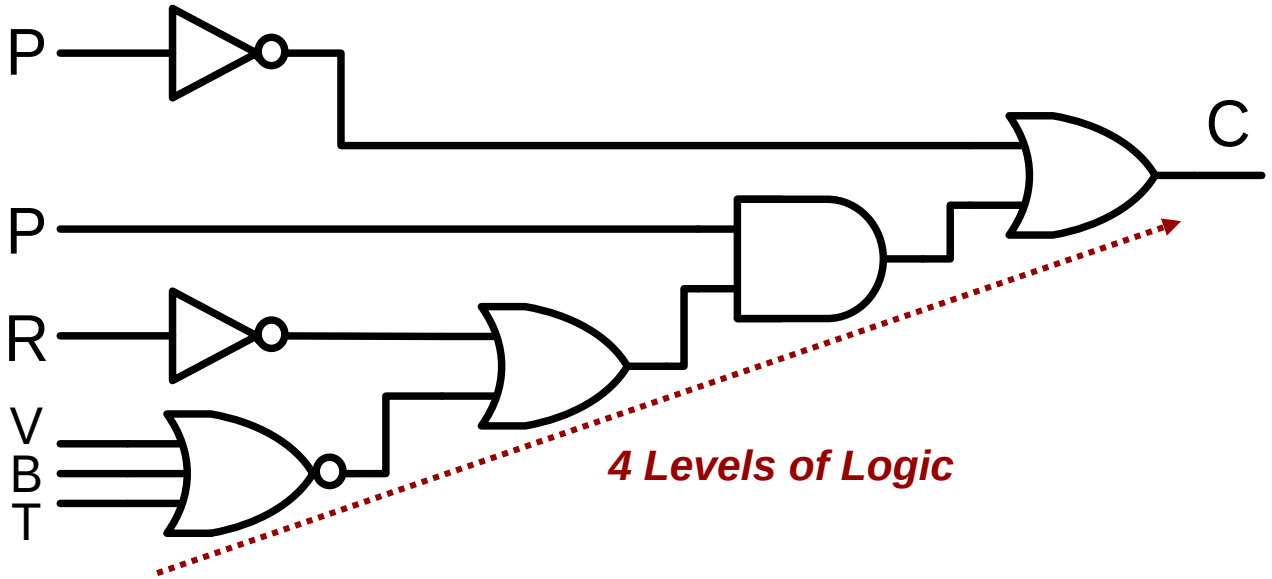
$$C = \overline{P} + P(\overline{S}+\overline{R})$$
$$= \overline{P} + P(\overline{(V+B+T)}+\overline{R})$$

Car Stereo Example



$$C = \overline{P} + P(\overline{(V+B+T)}+\overline{R})$$

Car Stereo Example

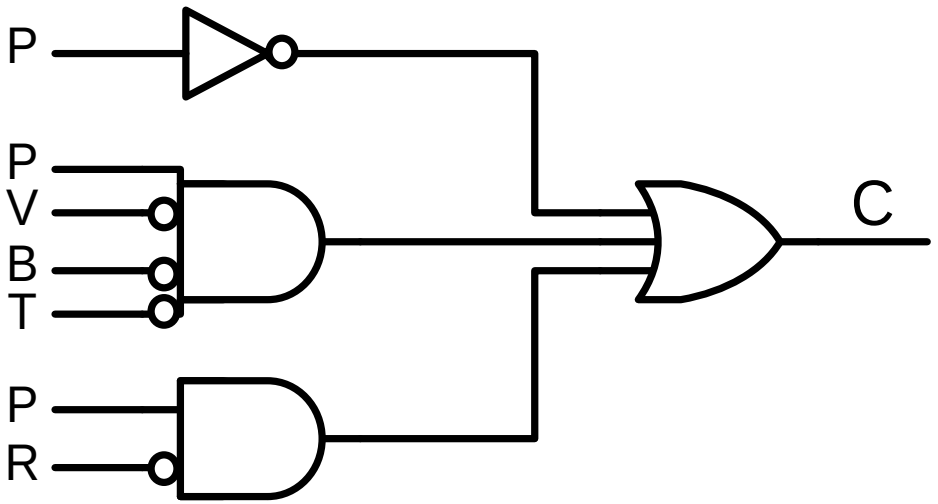


$$C = \overline{P} + P(\overline{(V+B+T)}+\overline{R})$$

2-Level Implementation

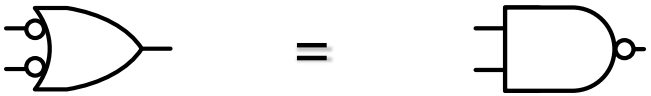
- Converting to SOP or POS always yields a 2-level implementation

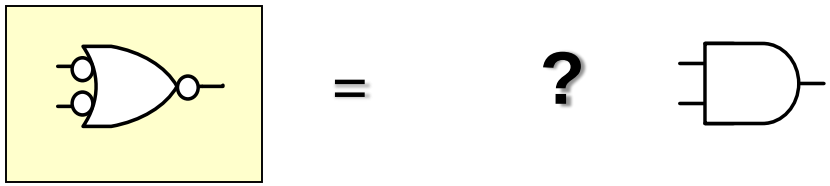
$$C = \overline{P} + P(\overline{(V+B+T)}+\overline{R})$$
$$= \overline{P} + P((\overline{V}\bullet\overline{B}\bullet\overline{T})+\overline{R})$$
$$= \overline{P} + P\bullet\overline{V}\bullet\overline{B}\bullet\overline{T} + P\overline{R}$$

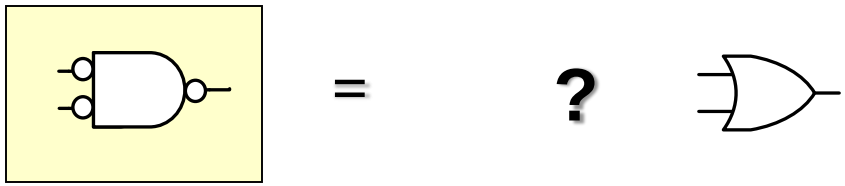


NEGATIVE (ACTIVE-LO) LOGIC

DeMorgan Equivalents

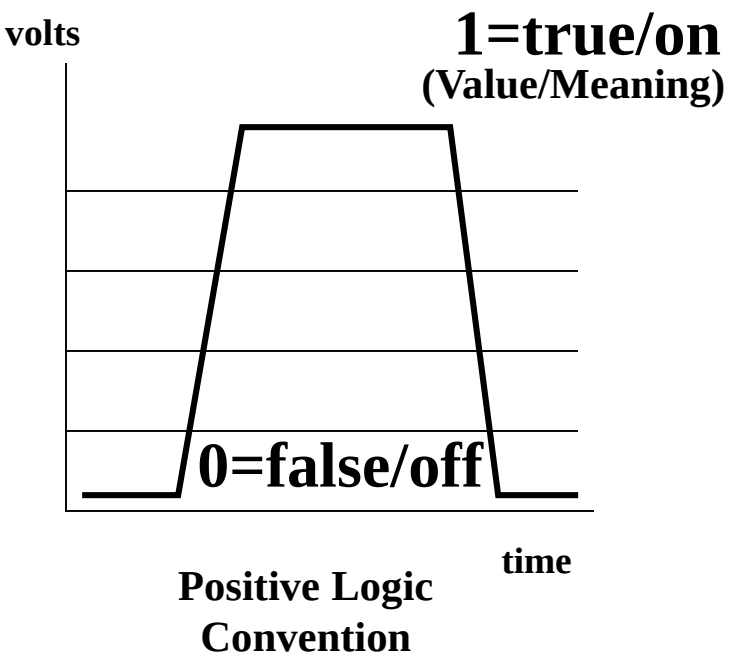


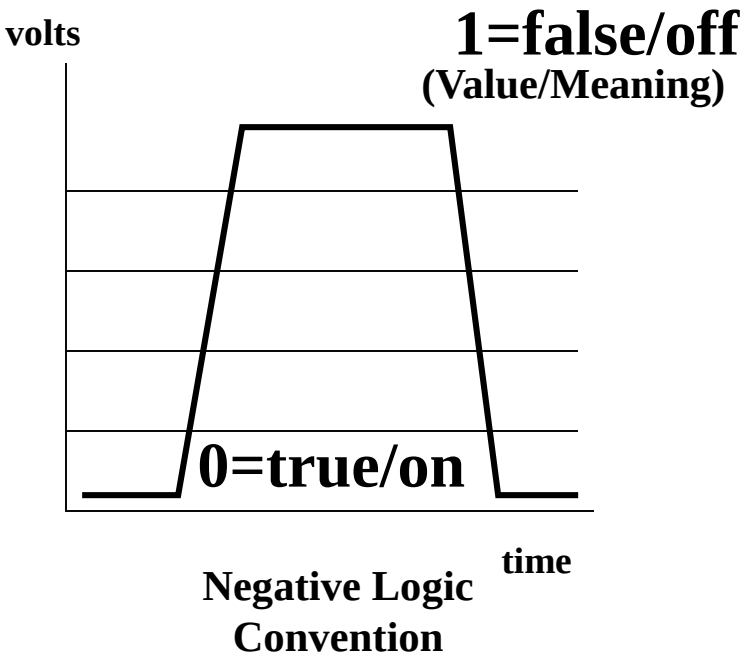




Negative Logic

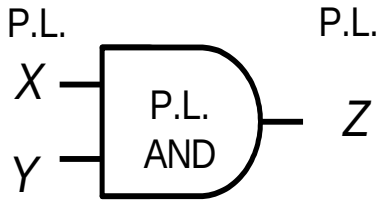
- Recall it is up to us humans to assign meaning to the two voltage levels
  - Thus, far we've used (unknowingly) the positive logic convention where 1 means true and 0 means false
  - In negative logic 0 means true and 1 means false





Negative Logic 'AND' Function

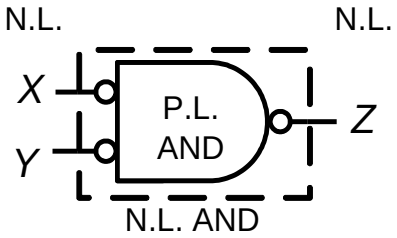
Traditional P.L. AND



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

Traditional AND gate functionality assumes positive logic convention

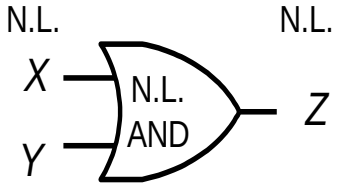
N.L. AND function



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

Given negative logic signals, we can invert to positive logic, perform the AND operation, then convert back to negative logic

N.L. AND = P.L. OR

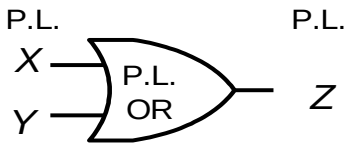


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

However, we then see that an OR gate implements the negative logic 'AND' function

Negative Logic 'OR' Function

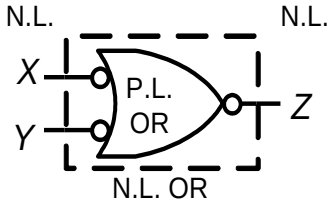
Traditional P.L. OR



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

Traditional OR gate functionality assumes positive logic convention

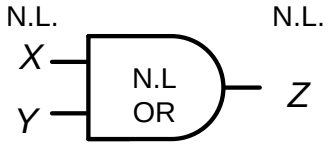
N.L. OR function



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

Given negative logic signals, we can invert to positive logic, perform the OR operation, then convert back to negative logic

N.L. OR = P.L. AND

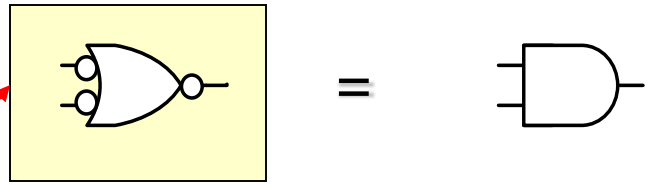


X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

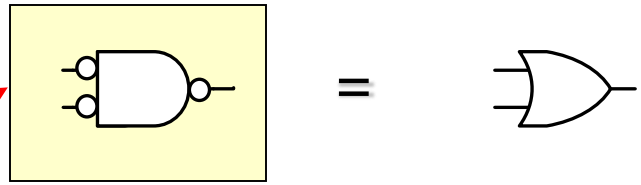
However, we then see that an AND gate implements the negative logic 'OR' function

Negative Logic

A negative logic OR function is equivalent to an AND gate



A negative logic AND function is equivalent to an OR gate



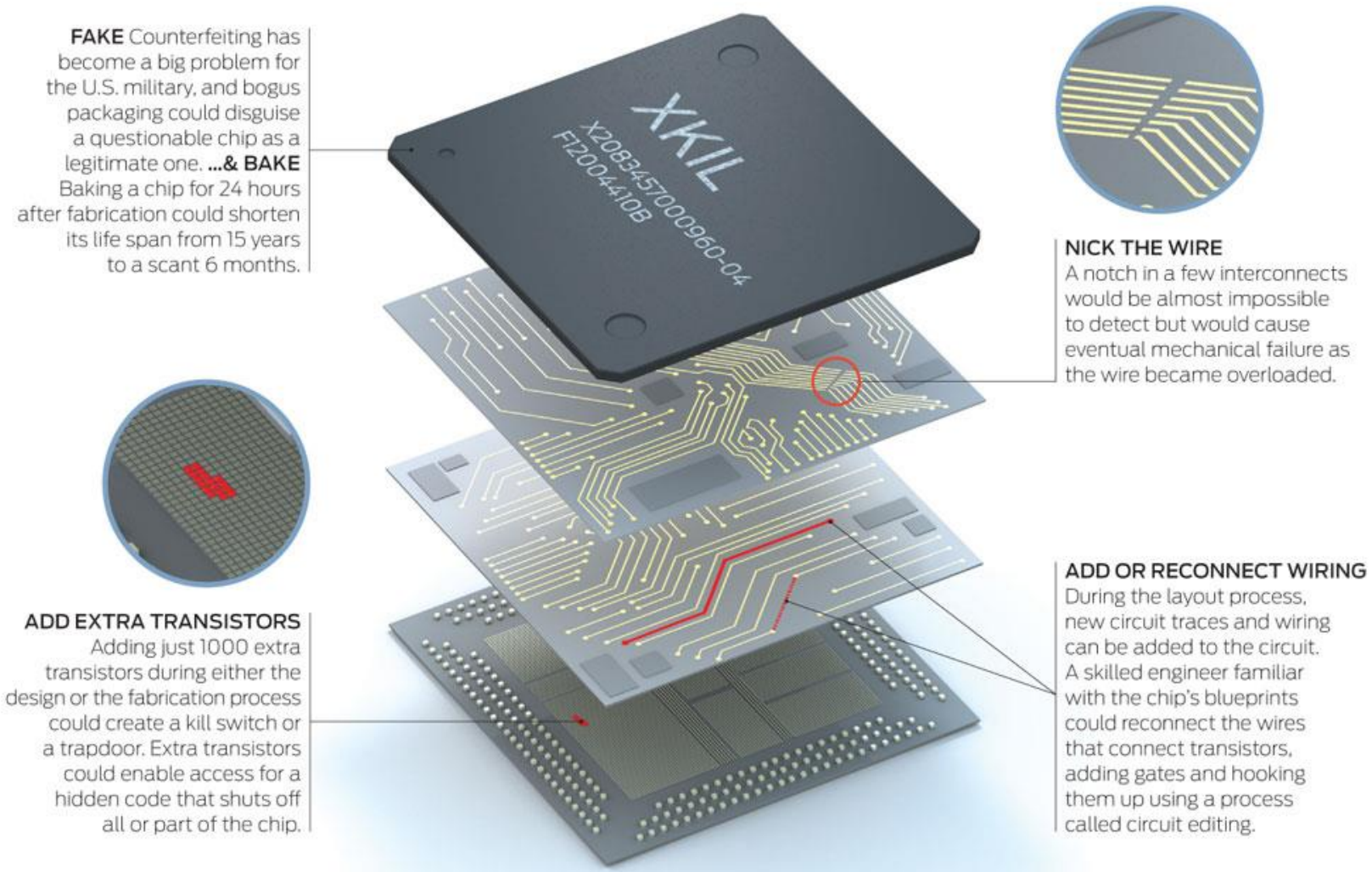
These are the preferred way of showing the N.L. functions because the inversion bubbles explicitly show where N.L. is being converted to P.L. and the basic gate schematics retain their meaning (when we see an AND gate we know we're doing some king of AND function with the bubbles indicating N.L.)

OLD

Application: TRUST Program

- The situation…
  - U.S. military consumes 1% of all chips worldwide
  - More and more chips are designed in the U.S. but fabricated in other countries (Taiwan, Korea, etc.)
  - Chip designs often use some IP (=Intellectual Property = pre-designed circuit components from another vendor)
- The problem: Sabotage in the form of…
  - Malicious changes to a design before fabrication
  - Introduce flaws during the fabrication process
  - Add additional logic that can be triggered at some later point in time
- Solution: DARPA's Trust in IC's program

Application: TRUST Program



Solutions

- Design Changes
  - Simulation
    - Describe the circuit to a computer, indicate what input combinations to plug in and let software CAD tools tell us what the output will be
    - Problem: 2^n combinations…too many
  - Formal Verification
    - Given two circuits, use theorems and other methods to formally prove that two circuits are equivalent or not
- Fabrication Errors
  - Reverse Engineer a circuit: shave off one layer of a chip at a time, taking images of each layer and let a computer reconstruct a model of the circuit; then compare it to original design
  - Other more advanced techniques (X-ray imaging, etc.)

Unique Representations

- Canonical => Same functions will have same representations
- Truth Tables along with Canonical Sums and Products specify a function *uniquely*
- Equations/circuit schematics are NOT inherently canonical

Truth Table

x	y	z	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Canonical Sum

$$P = \sum_{x,y,z}(2,3,5,7)$$

rows where P=1

Yields SOP equation, AND-OR circuit

Canonical Product

$$P = \prod_{x,y,z}(0,1,4,6)$$

rows where P=0

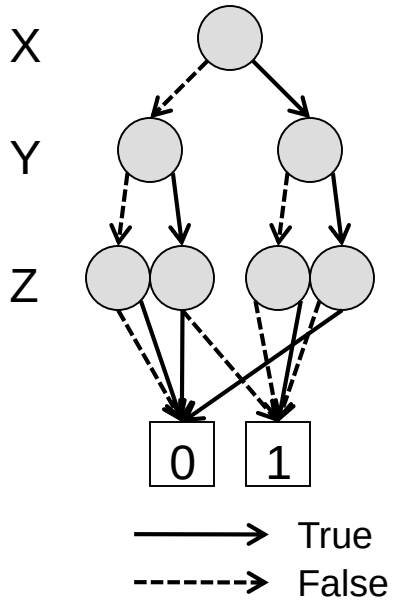
Yields POS equation, OR-AND circuit

Binary Decision Diagram

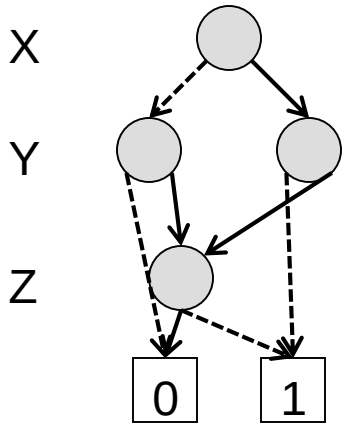
- Graph (binary tree) representation of logic function
- Vertex = Variable/Decision
- Edge = Variable value (T / F)

X	Y	Z	F
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

X	Y	Z	F
0	**0**	0	**0**
		1	**0**
	1	0	**1**
		1	**0**
1	**0**	0	**1**
		1	**1**
	1	0	**0**
		1	**1**



BDD for F



Canonical Representations

- As long as variable mapping between 2 functions is known then the following can be used as a unique/canonical representation [i.e. will be the same for the 2 functions]
- Truth Tables
  - show output for all possible combinations
  - n inputs => 2^n rows, only good for ≤ 5 var's.
- Canonical Sums
  - indicates where a function is 1 by listing the input combinations (rows) where it equals 1
- Canonical Products
  - indicates where a function is 0 by listing the input combinations (rows) where it equals 0
- Binary Decision Diagrams
  - Can be compact & are helpful to represent for computer tools

$$P = \sum_{x,y,z}(2,3,5,7)$$

? = ?

$$Q = \sum_{a,b,c}(2,3,5,7)$$

Yes, If x=a, y=b, z=c