Forms and simplifications of Boolean Expressions

Canonical forms of a boolean expression

Standard ways of uniquely representing boolean expressions

1. Sum of *minterms*
2. Product of *maxterms*

Terminology

Truth table for a 3-variable logic function F(X,Y,Z)

Row	X	Y	Z	F
0	0	0	0	F(0,0,0)
1	0	0	1	F(0,0,1)
2	0	1	0	F(0,1,0)
3	0	1	1	F(0,1,1)
4	1	0	0	F(1,0,0)
5	1	0	1	F(1,0,1)
6	1	1	0	F(1,1,0)
7	1	1	1	F(1,1,1)

Table 1.

Terminology (1)

Literal	A variable (X) or a complement of a variable (X’)
Product term	A single literal (X) or a logical product of two or more literals (X.Y.Z)
Sum-of-products expression	A logical sum of product terms: X + Y.Z + X’.Z
Sum term	A single literal or a logical sum of two or more literals: X + Y’ + Z
Product-of-sums expression	A logical product of sum terms: Z’.(X+Y).(Y’+Z)
Normal term	A product or sum term where no variable appears more than once: X.Y.Z’, X + Y’ + Z’ etc
Minterm	An n-variable minterm is a normal product term with n literals
Maxterm	An n-variable maxterm is a normal sum term with n literals

Minterms and maxterms

Row	X	Y	Z	F	Minterm		Maxterm	
0	0	0	0	F(0,0,0)	m_0	X'.Y'.Z'	M_0	X+Y+Z
1	0	0	1	F(0,0,1)	m_1	X'.Y'.Z	M_1	X+Y+Z'
2	0	1	0	F(0,1,0)	m_2	X'.Y.Z'	M_2	X+Y'+Z
3	0	1	1	F(0,1,1)	m_3	X'.Y.Z	M_3	X+Y'+Z'
4	1	0	0	F(1,0,0)	m_4	X.Y'.Z'	M_4	X'+Y+Z
5	1	0	1	F(1,0,1)	m_5	X.Y'.Z	M_5	X'+Y+Z'
6	1	1	0	F(1,1,0)	m_6	X.Y.Z'	M_6	X'+Y'+Z
7	1	1	1	F(1,1,1)	m_7	X.Y.Z	M_7	X'+Y'+Z'

Table 2.

Terminology (2)

Minterm number	An n-variable minterm can be represented by an n-bit integer
Minterm i	Minterm corresponding to the row i of the truth table
Maxterm i	Maxterm corresponding to the row i of the truth table
Canonical sum	Sum of the minterms corresponding to the truth table rows for which the output is 1. $\Sigma_{X,Y,Z}m(0,3,4,6,7) = X'.Y'.Z' + X'.Y.Z + X.Y'.Z' + X.Y.Z' + X.Y.Z$
Minterm list	$\Sigma_{X,Y,Z}m(0,3,4,6,7)$
on-set	Same as the minterm list
Canonical product	Product of maxterms corresponding to the truth table rows for which the output is 0. $\Pi_{X,Y,Z}M(1,2,5) = (X+Y+Z').(X+Y'+Z).(X'+Y+Z')$
Maxterm list	$\Pi_{X,Y,Z}M(1,2,5)$
off-set	Same as the maxterm list

Equivalent ways of representing a combinational logic function

- Truth table
- Canonical sum of product terms
  - Algebraic sum of minterms
- Minterm list
- Canonical product of sum terms
  - Algebraic product of maxterms
- Maxterm list

Standard Form

- This is a simplified canonical form
- Canonical form is a special case of the standard form

There are two types:

1. Sum of Products (SoP)
    a. Products do not necessarily have to be minterms
2. Product of Sums (PoS)
    a. Sums do not necessarily have to be maxterms

Standardization makes the evaluation, simplification, and implementation of Boolean expressions more systematic and easier.

Sum of Products

X + X.Y’ + X’.Y.Z

- Each term can contain a single literal or a logical product of multiple literals
- Can only contain complements of single variables

Canonical sum of products (sum of minterms)

Each term in SoP should contain all the variables

- ABC + A’BC’ + ABC’

Converting a SoP into a canonical sum of minterms

SoP (non-canonical): AB + ABC + CB

Canonical sum of minterms:

AB.(C+C’) + ABC + (A+A’)CB

= ABC + ABC’ + ABC + ACB + A’CB

Since X + X = X

<u>ABC</u> + ABC’ + <u>ABC</u> + <u>ACB</u> + A’CB

ABC + ABC’ + A’CB

Converting a SoP into a canonical sum of minterms

SoP (non-canonical): AB + B’

Canonical sum of minterms:

= AB + (A+A’)B’

= AB + AB’ + A’B’

$\mathbf{= m_3 + m_2 + m_0}$

$\mathbf{= \sum (m_3, m_2, m_0)}$

$\mathbf{= \sum m(0,2,3)}$

A	B	Minterm		Maxterm	
0	0	m_0	A’B’	M_0	A+B
0	1	m_1	A’B	M_1	A+B’
1	0	m_2	AB’	M_2	A’+B
1	1	m_3	AB	M_3	A’+B’

Product of Sums

PoS: (A+B).(A’+B+C’).B

- Product of sum terms
- Each sum term can contain a single literal or any sum of those

Canonical product of sums (sum of maxterms)

- Each sum term should contain each of the variables or their complement
  - (A+B+C’)(A’+B’+C’)

Converting a PoS into a canonical product of maxterms

PoS: (A+B).A

Canonical PoS:

Use A = A+0 and B.B’=0

(A+B). (A+0)

= (A+B).(A+B.B’)

= (A+B).(A+B).(A+B’)

=(A+B).(A+B’)

= $M_0.M_1$

= Π (M_0,M_1)

= Π M(0,1)

A	B	Minterm		Maxterm	
0	0	m_0	A’B’	M_0	A+B
0	1	m_1	A’B	M_1	A+B’
1	0	m_2	AB’	M_2	A’+B
1	1	m_3	AB	M_3	A’+B’

Duality of canonical forms

- The following are equivalent:
  - $\Sigma_{X,Y,Z} m(0,3,4,6,7) = X'.Y'.Z' + X'.Y.Z + X.Y'.Z' + X.Y.Z' + X.Y.Z$
  - $\Pi_{X,Y,Z} M(1,2,5) = (X+Y+Z').(X+Y'+Z).(X'+Y+Z')$
  - A function expressed as an SoP can also be expressed as a PoS
- $\Sigma_{X,Y,Z} m(0,3,4,6,7) = \Pi_{X,Y,Z} M(1,2,5)$
- In general, $\Sigma m(\{a\}) = \Pi M(\{b\})$ where,
  - $\{a\} \cup \{b\} = \{0,1,2,...,2^n-1\}$
  - $\{a\} \cap \{b\} = \{\phi\}$

Canonical forms and De Morgan's law

A sum of minterms equals the inverse of the product of the corresponding maxterms

$$\sum m(\{a\}) = \overline{\prod M(\{a\})}$$

$$\begin{aligned}\overline{\prod M(\{a\})} &= \overline{\left(M_{a_1} \cdot M_{a_2} \cdot \cdots \cdot M_{a_k}\right)} \\ &= \overline{M_{a_1}} + \overline{M_{a_2}} + \cdots + \overline{M_{a_k}} \\ &= m_{a_1} + m_{a_2} + \cdots + m_{a_k} = \sum m(\{a\})\end{aligned}$$

E.g. $M_0' = (A+B+C)' = A'B'C' = m_0$

Why canonical SoP and PoS?

Canonical SOP and POS are unique for each boolean function

Provides a concise and unique way of representing a truth table

We can use a specific set of rules that guarantee the simplest function for any logic expression

Elements of Combinational Logic Design

Outline

1. Implementing boolean logic circuits
    a. Standard form
    b. Canonical form
2. Simplifying logic circuits
3. Universal gates
    a. NAND/NOR implementations

Implementing boolean logic circuits

Typical architecture to implement a standard form

Level 1 - to complement some of the variables (this might be skipped in some diagrams)

Level 2 -

- AND gates for SoP
- OR gates for PoS

Level 3 -

- One n-input OR gate for SoP for getting the sum of everything
- One n-input AND gate for PoS for getting the product of everything

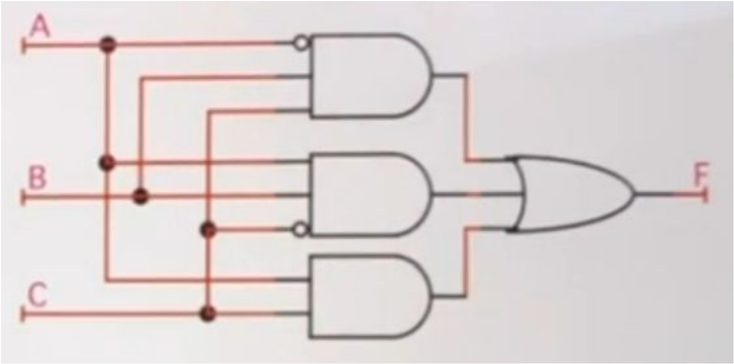
SoP: AND-OR implementation

PoS: OR-AND implementation

Example of AND-OR Standard Form

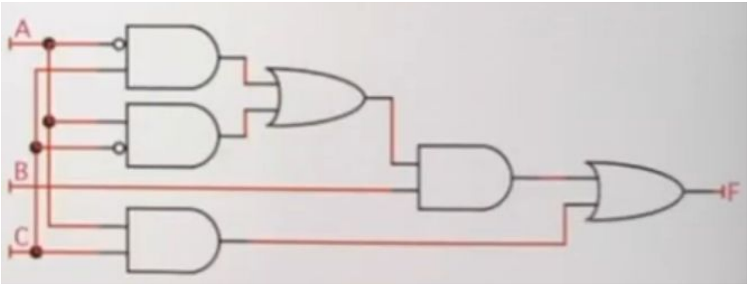
F = A’BC + ABC’ + AC

Standard form but not canonical



F = B(A’C + AC’) + AC

(Non-standard form)



- Fewer inputs but more gate levels
- Comparison & implementation is more difficult than with the standard form

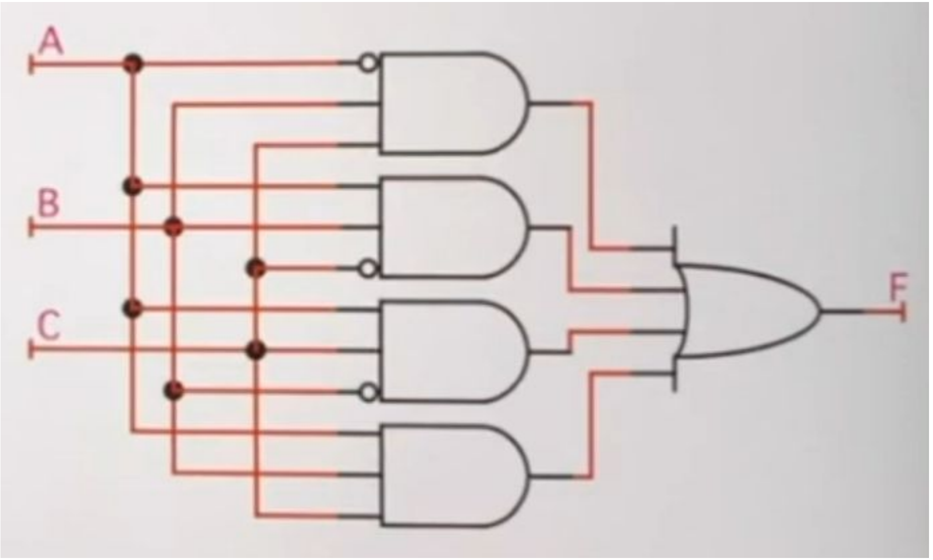
Same example in canonical form

Sum of minterms

F = A’BC + ABC’ + AC

= A’BC + ABC’ + AC(B+B’)

= A’BC + ABC’ + ACB+ ACB’



Simplifying standard forms

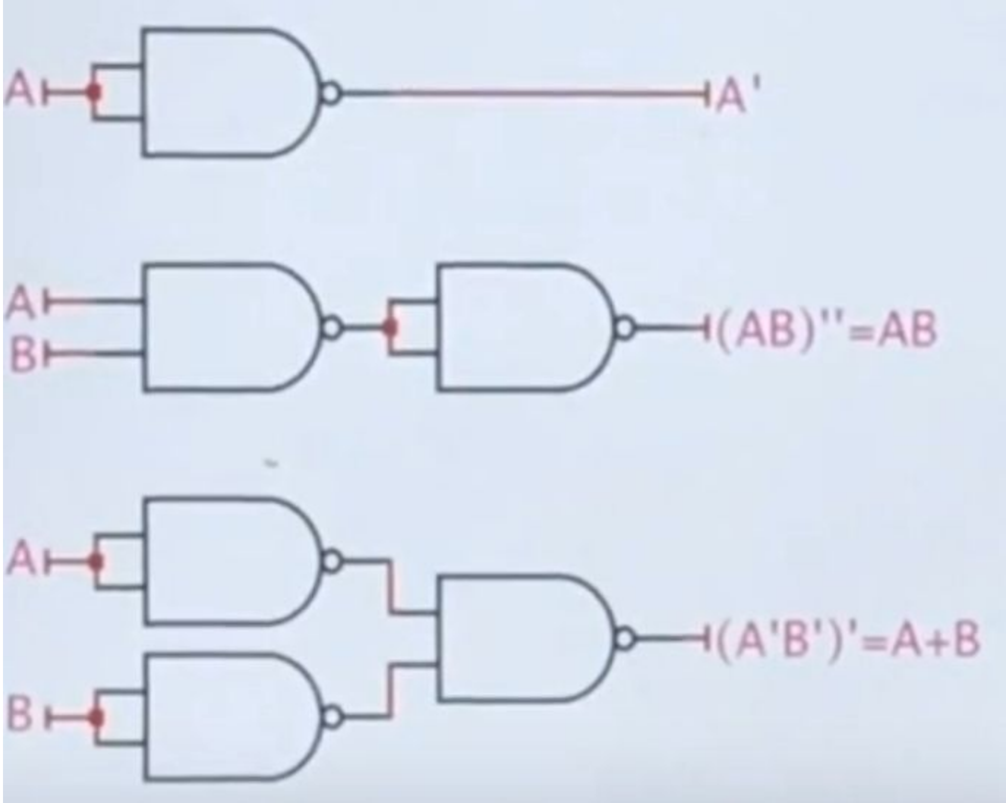
- Two-level minimum cost design
  - PoS or SoP with the minimum number of terms
  - Each term has the minimum number of literals
- How to get the simplest design systematically?
  - Karnaugh maps!

Universal Gates

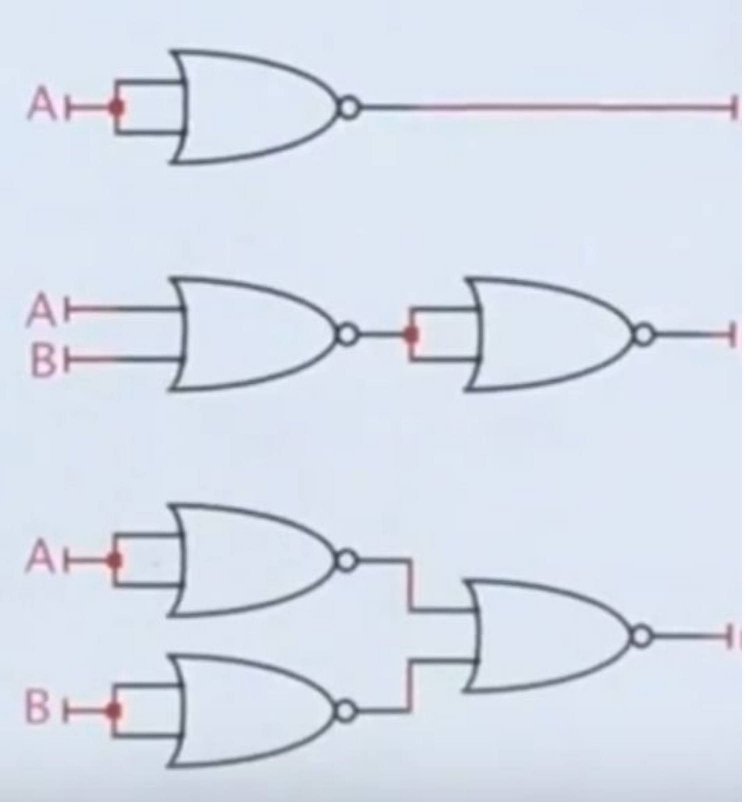
Universal gate

- A gate is a universal gate if a collection of that gates can be arranged to implement AND, OR & NOT gates
- All circuits can be represented in standard form
  - I.e. all circuits can be built with AND, OR, NOT
  - Therefore, a universal gate can be used to implement any circuit
- A universal gate is “functionally complete”

Is NAND universal?



Is NOR universal?



A’

(A+B)’’ = A+B

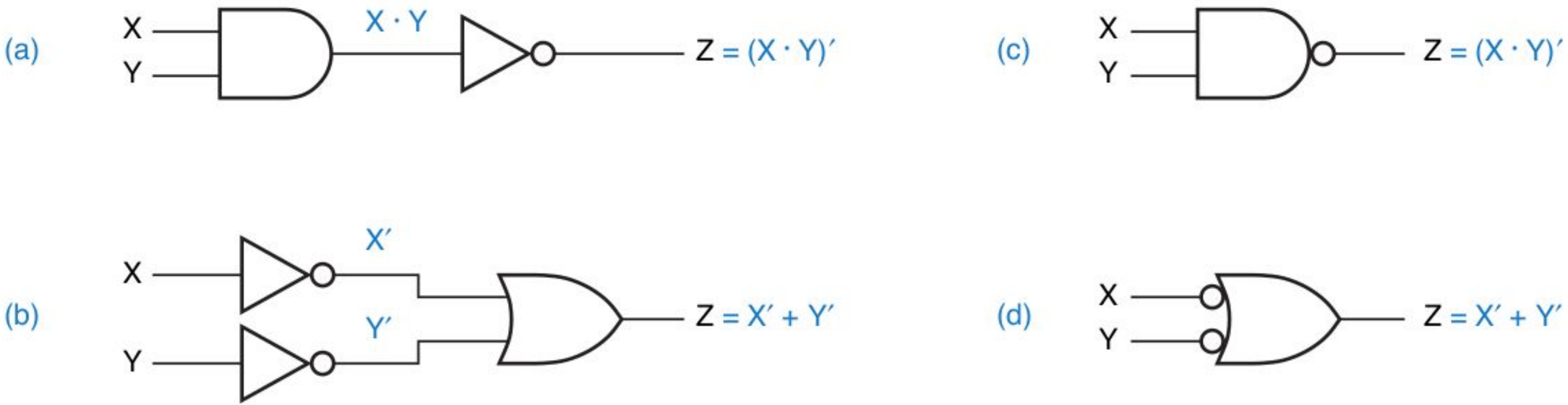
(A’+B’)’ = A.B

NAND and NOR implementation of circuits

- Function-based approach
  - Use De Morgan’s law to move from AND-OR to NAND-NOR
  - E.g. $AB+CD = (AB+CD)'' = ((AB)'(CD)')'$
- Circuit-based approach
  - Replace all gates with NAND gates
  - Undo any complements caused by the replacement

NAND & NOR gates are generally faster than AND & OR gates in most technologies

NAND equivalent circuits (De Morgan’s theorem)



Implementation of NOR

- Note the equivalence (De Morgan's law)
  - (A+B)' = A'.B'
- If you take the complement of both sides, we get another equivalence
  - A+B = (A'.B')' (i.e. NAND of inverted inputs)

NOR equivalent circuits (De Morgan’s theorem)

Try it out (somewhat parallel to slide 30)

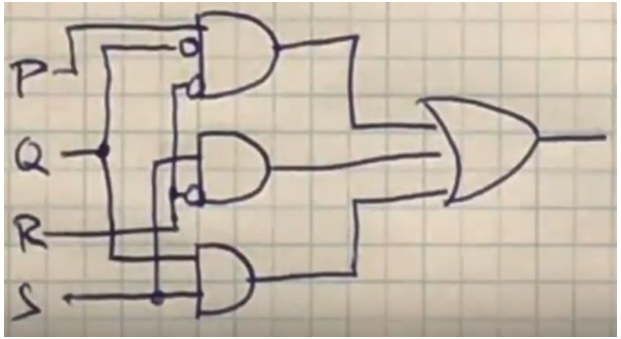
Example

F = PQS + P'R'S + PQ'R' + P'QRS

Step 1: Simplify this using a Karnaugh Map

PQ \ RS	00	01	11	10
00		1		
01		1	1	
11		1	1	
10	1	1		

QS + R'S + PQ'R'



Example

$F = QS + R'S + PQ'R'$

To implement this using NAND gates:

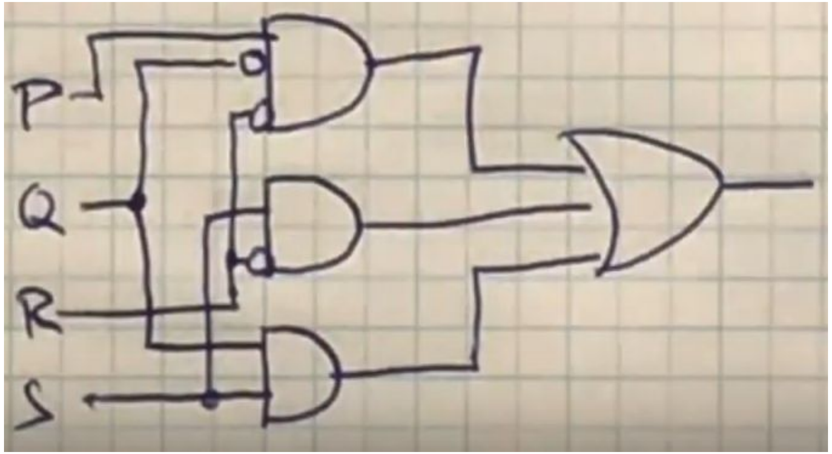
Step 2 - invert and invert again, to use De Morgan's law

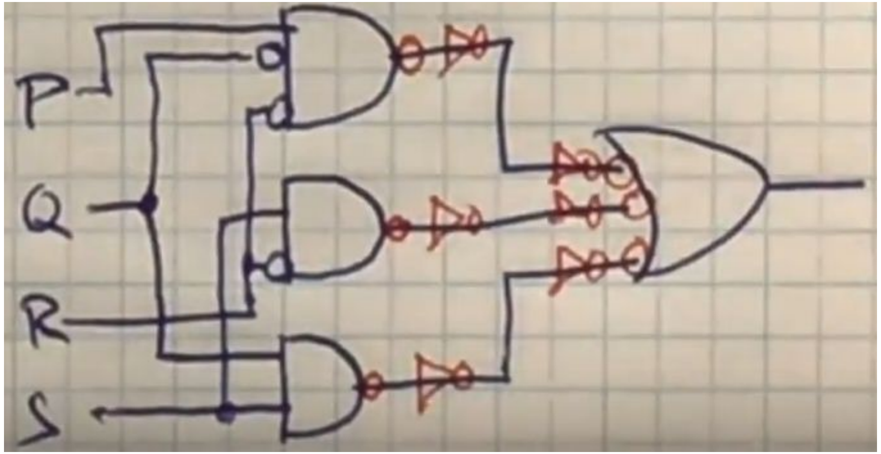
$F = F'' = (QS + R'S + PQ'R')''$

$\mathbf{= ((QS)'.(R'S)'.(PQ'R')')'}$

Homework: Draw the circuit

Example: Circuit based approach to convert to NAND

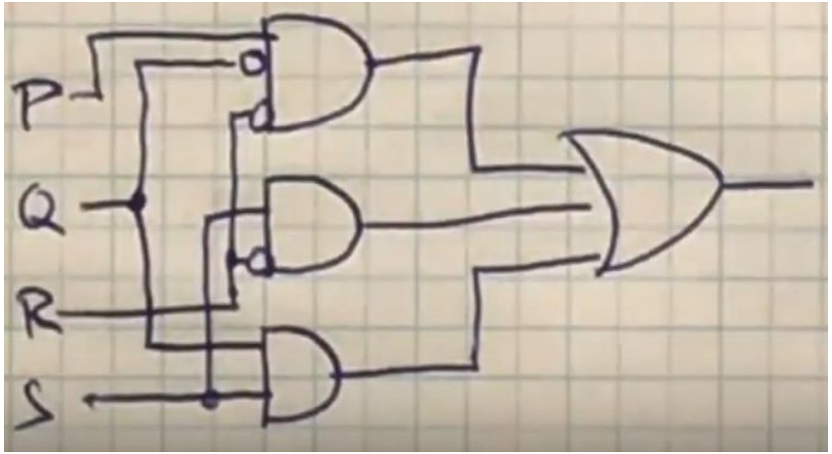




QS + R’S + PQ’R’

- Replace AND with NAND
  - Add NOT to fix unwanted inversions
- Convert OR into NAND by inverting the inputs (De Morgan, Slide 28)
  - Add NOTs to fix unwanted inversions

Example: Circuit based approach to convert to NAND



QS + R’S + PQ’R’

- Simplify the NOT gates that cancel each other out

Deriving / simplifying boolean functions: beyond K-maps

- K-maps are typically used for cases where there are 4 variables or less
- For more complex cases, particularly when multiple outputs are involved, there are other algorithms which are more suitable
    - Quine-McCluskey algorithms
        - This is a programmed minimization method
        - Typically, a high-level programming language implementation is used for running the algorithms