CS 201

Digital Logic

Introduction to Logic

Related Resources:
Mano, Chapter 1 and 2

Binary Algebra

- All variables have one of two values: 0 1
- Strings of variables represent data
  - Numbers, letters, colours, sounds etc.
  - Only our interpretation provides meaning.
- As numbers:
  - Base 2, each place value means "$+a \times 2^n$"

$e.g.: 101_2$

Some examples

$1_2 = 2^0 = 1_{10}$

$10_2 = 2^1 = 2_{10}$

$100_2 = \boxed{}$

$1000_2 = \boxed{}$

$1111_2 = \boxed{}$

$10000_2 = \boxed{}$

$1\ 0000\ 0000_2 = \boxed{}$

Know these

$0_{10} = 0000_2$	$8_{10} = 1000_2$
$1_{10} = 0001_2$	$9_{10} = 1001_2$
$2_{10} = 0010_2$	$10_{10} = 1010_2$
$3_{10} = 0011_2$	$11_{10} = 1011_2$
$4_{10} = 0100_2$	$12_{10} = 1100_2$
$5_{10} = 0101_2$	$13_{10} = 1101_2$
$6_{10} = 0110_2$	$14_{10} = 1110_2$
$7_{10} = 0111_2$	$15_{10} = 1111_2$

Others to know

2^{10} = 1024 = 1 Kilo- ≅ 10^3 =1 k

2^{20} = 1,048,576 = 1 Mega- ≅ 10^6

2^{30} = 1,073,741,824 = 1 Giga- ≅ 10^9

2^{40} = 1,099,511,627,776 = 1 Tera- ≅ 10^{12}

2^{50} = 1,125,899,906,842,624 = 1 Peta- ≅ 10^{15}

these are for bytes.

Use exactly 10^3, 10^6, 10^9, 10^{12}, 10^{15}

for Hz, flops etc

In general

- $2^{10n} \cong 10^{3n}$

  Exa-: $10^{18} \cong 2^{60} = 1.1529 \times 10^{18}$

  Zetta-: $10^{21} \cong 2^{70} = 1.1806 \times 10^{21}$

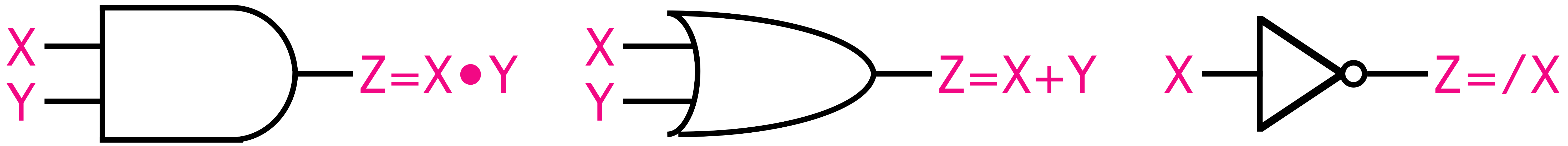
  Yotta-: $10^{24} \cong 2^{80} = 1.2089 \times 10^{24}$
- Works OK till $n = 30$

  $2^{300} = 2.037035976334486 \times 10^{90}$

  $2^{299} = 1.018517988167243 \times 10^{90}$
- Googol $= 10^{100} \cong 2^{333}$

Back to binary algebra

- an ***Algebra*** is a system with ***symbols*** operated upon by ***actions***.
- Binary algebra: two symbols are {0,1}
  - also called Boolean Algebra: George Boole (1815-1864)
  - Could use any two symbols:
    - {a,b} {true,false} {⇑,⇓} {♦,♦} {1,0}
- And the actions?
  - 3 basic primitives
  - AND (•, &, ∩, XY)   OR (+, |, ∪)   NOT (/,¬,$\overline{X}$, X')



What do they do?

- AND is 1 *iff* all inputs are 1
- OR is 0 *iff* all inputs are 0 (not like english)
- NOT is opposite:
  - 1 if the input is 0, and 0 if the input is 1
- Truth tables:

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

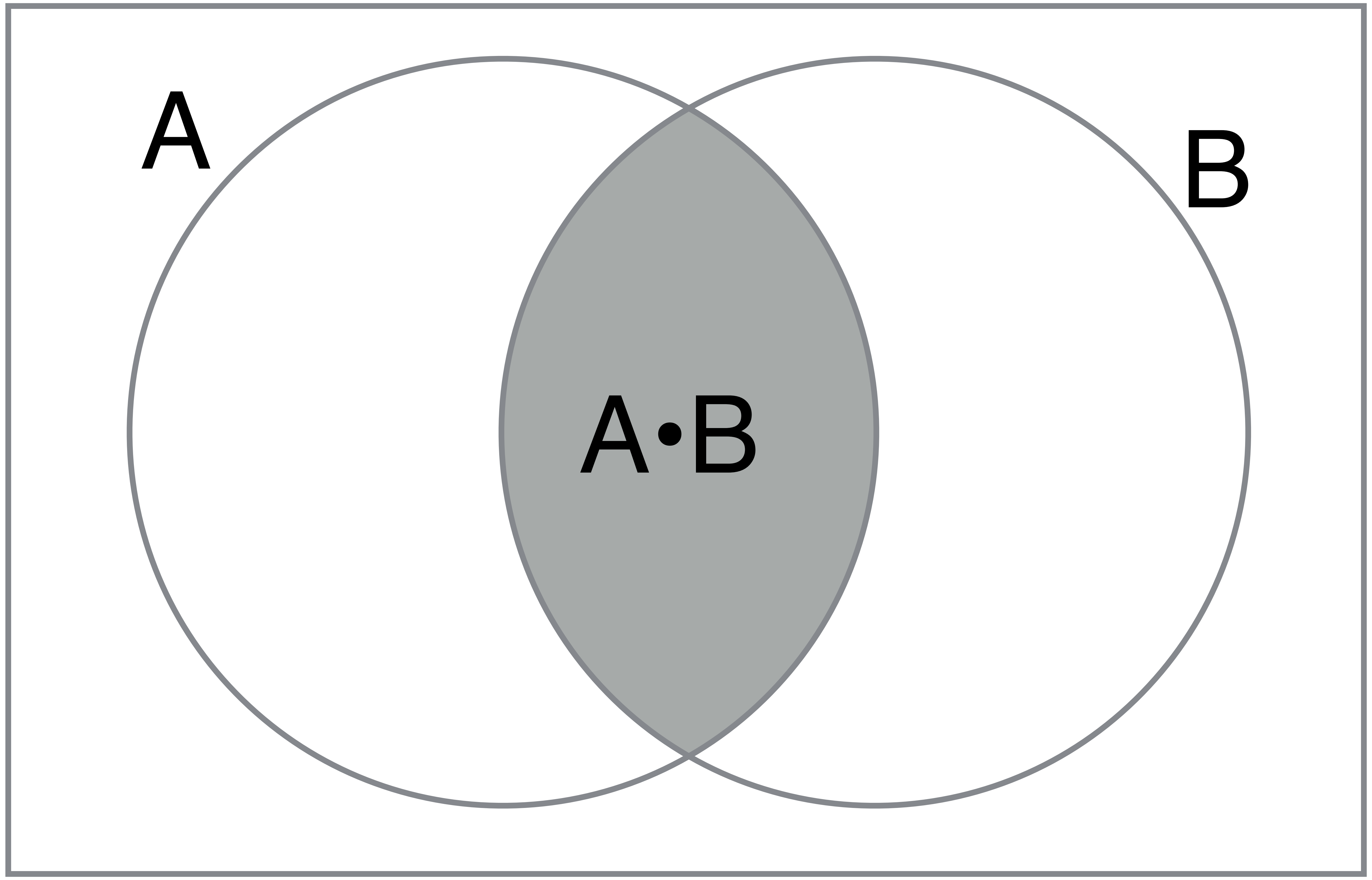
x	$\bar{x}$
0	1
1	0

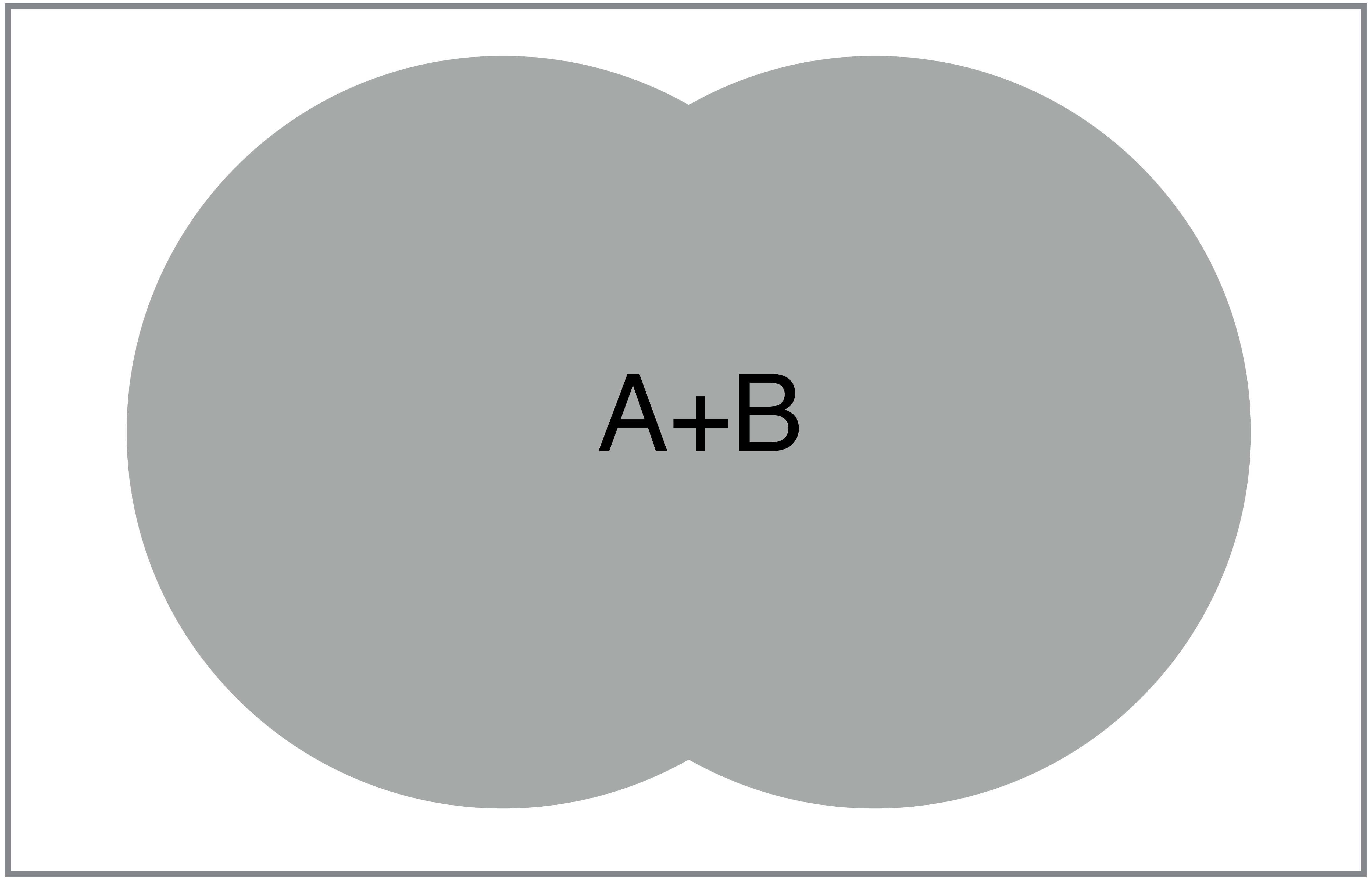
Functions of two variables

- A function of 2 variables has 4 possible output cases
- Therefore there are only 16 possible functions of 2 variables (some with names and common usage)

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
Name		Zeros	AND	$A\overline{B}$	A	$\overline{A}B$	B	XOR	OR	NOR	XNOR	$\overline{B}$	$A+\overline{B}$	$\overline{A}$	$\overline{A}+B$	NAND	Ones

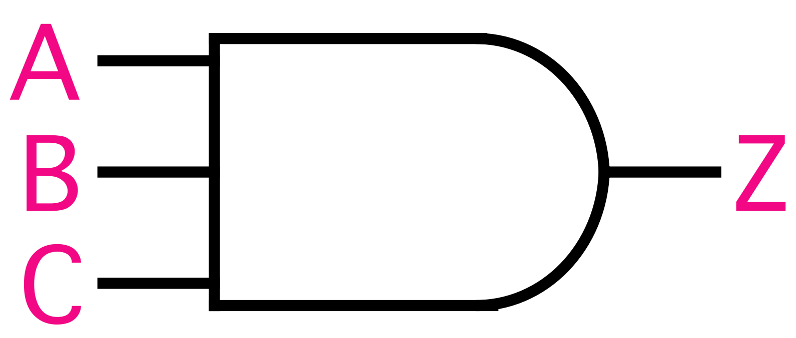
Venn Diagrams



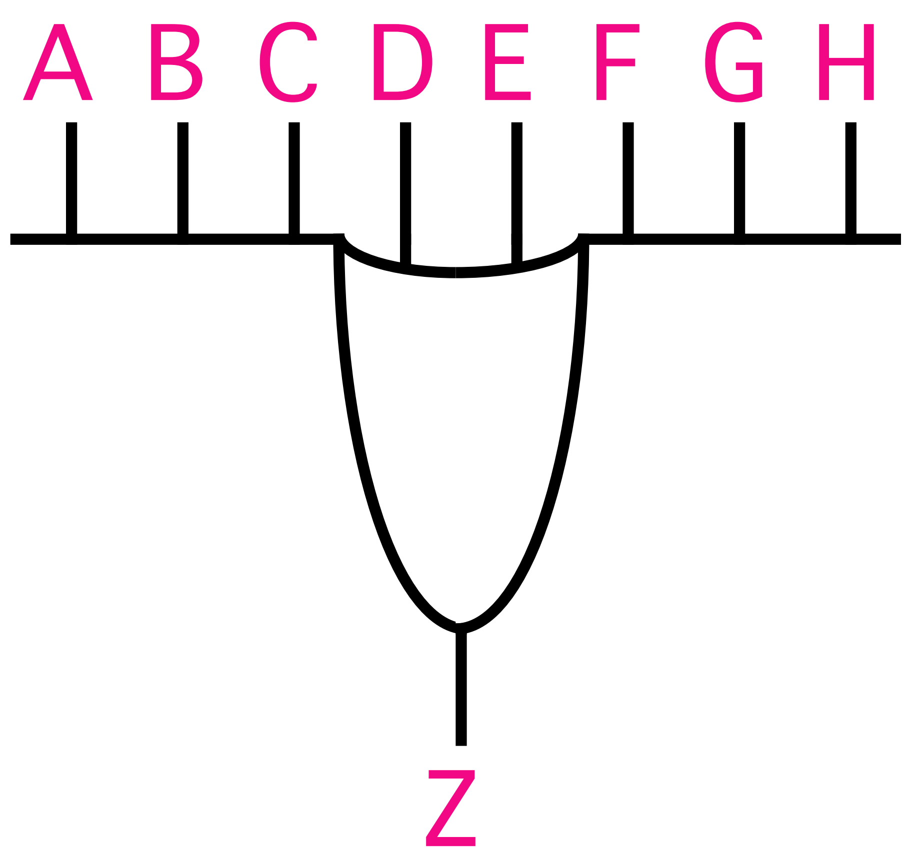


More gates

3-input AND



8-input OR



Formulas?

Z =

Z =

Why these symbols?

- AND is much like multiplication

  $0 \bullet n = 0$ ✅

  $1 \bullet n = n$ ✅

x	n	xn
0	0	0
0	1	0
1	0	0
1	1	1

- OR is sort of addition, but not really

  $0 + n = n$ ✅

  $1 + n = 1$ ❌

x	n	x+n
0	0	0
0	1	1
1	0	1
1	1	1

- Best to think of it as entirely new functions

Boolean Functions (*e.g.* $F = A + B$)

- Functions consist of
  - Binary variables
  - Binary constants {1, 0}
  - Logic operation symbols
  - Parentheses
  - Equal sign
- Each variable represents a binary value
  - For a given set of input variable values, $A, B \in \{0,1\}$, $F \in \{0,1\}$

Representations: Same Info

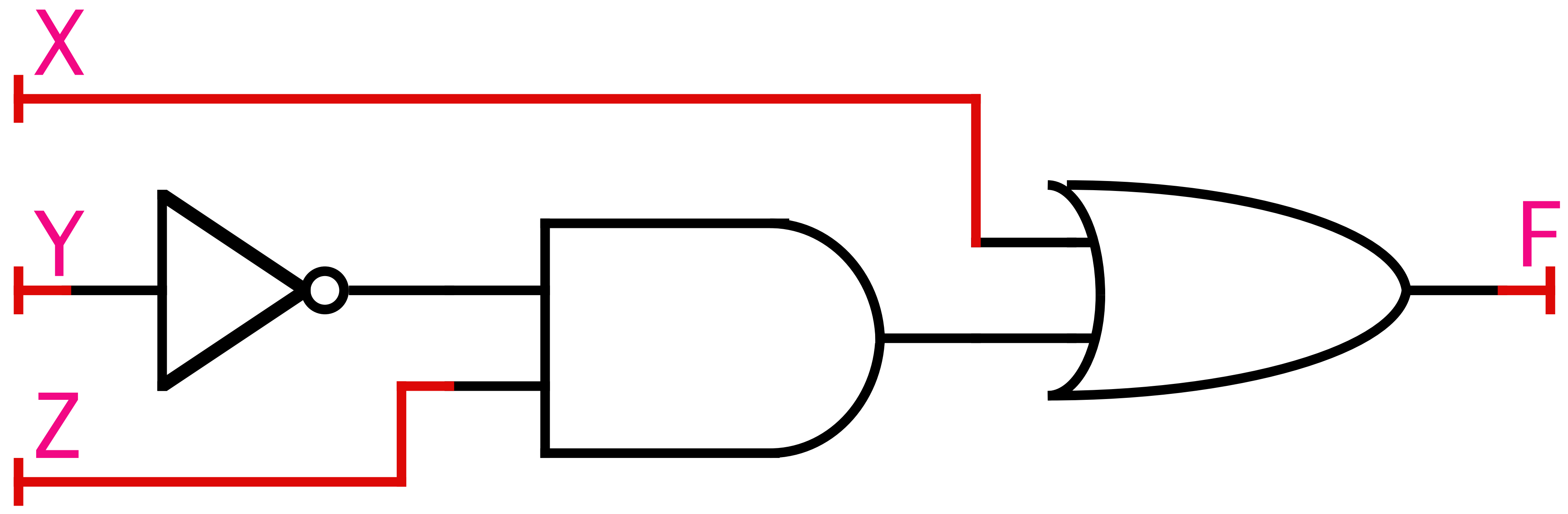
Function

$$F = X + \overline{Y}Z$$

Truth Table

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	1
1	1	1	1

Circuit Diagram



Boolean Identities (Mano and Kime table 2-3)

1. $x + 0 = x$	2. $x \bullet 1 = x$	Identity
3. $x + 1 = 1$	4. $x \bullet 0 = 0$	One/Zero
5. $x + x = x$	6. $x\,x = x$	Idempotent
7. $x + \overline{x} = 1$	8. $x\,\overline{x} = 0$	Inverse
9. $\overline{\overline{x}}=x$		Double Neg
10. $x + y = y + x$	11. $x\,y = y\,x$	Commutative
12. $x+(y+z) = (x+y)+z$	13. $x(yz) = (xy)z$	Associative
14. $x(y+z) = xy+xz$	15. $x+yz = (x+y)(x+z)$	Distributive
16. $\neg(x+y) = (\overline{x})(\overline{y})$	17. $(xy)' = \overline{x} + \overline{y}$	DeMorgan

Proofs

- All of these identities (and all valid boolean functions) can be proved exhaustively using truth tables.
  - Shows that in all cases, for all inputs, the output is the same
  - e.g. demorgan

x	y	x'	y'	x'+y'	xy	(xy)'
0	0	1	1	1	0	1
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	1	0	0	0	1	0

Simplification

- Given a boolean function,
- Use identities to simplify:
  - Reduce number of ***literals***
    - A literal is a single instance of a variable or input.
  - Reduce number of ***logic levels***
    - A logic level is a gate feeding into another gate.
  - Reduce number of ***operations***
    - An operation is a gate performing a computation.
- Sometimes can't reduce all of these at the same time
  - Tradeoffs, e.g. more levels for fewer literals

Simplification Examples(1)

- $F = X'YZ+X'YZ'+XZ$
  - 8 literals, 3 levels, 6 operations.

$= X'Y(Z+Z')+XZ$ Distributive

$= X'Y(1)+XZ$ Inverse

$= X'Y+XZ$ AND with 1

  - 4 literals, 3 levels, 4 operations.

Simplification proof by observing truth table

$X'YZ+X'YZ'+XZ$

$X'Y+XZ$

X	Y	Z	F
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

$X'YZ'$

$X'YZ$

XZ

$X'Y$

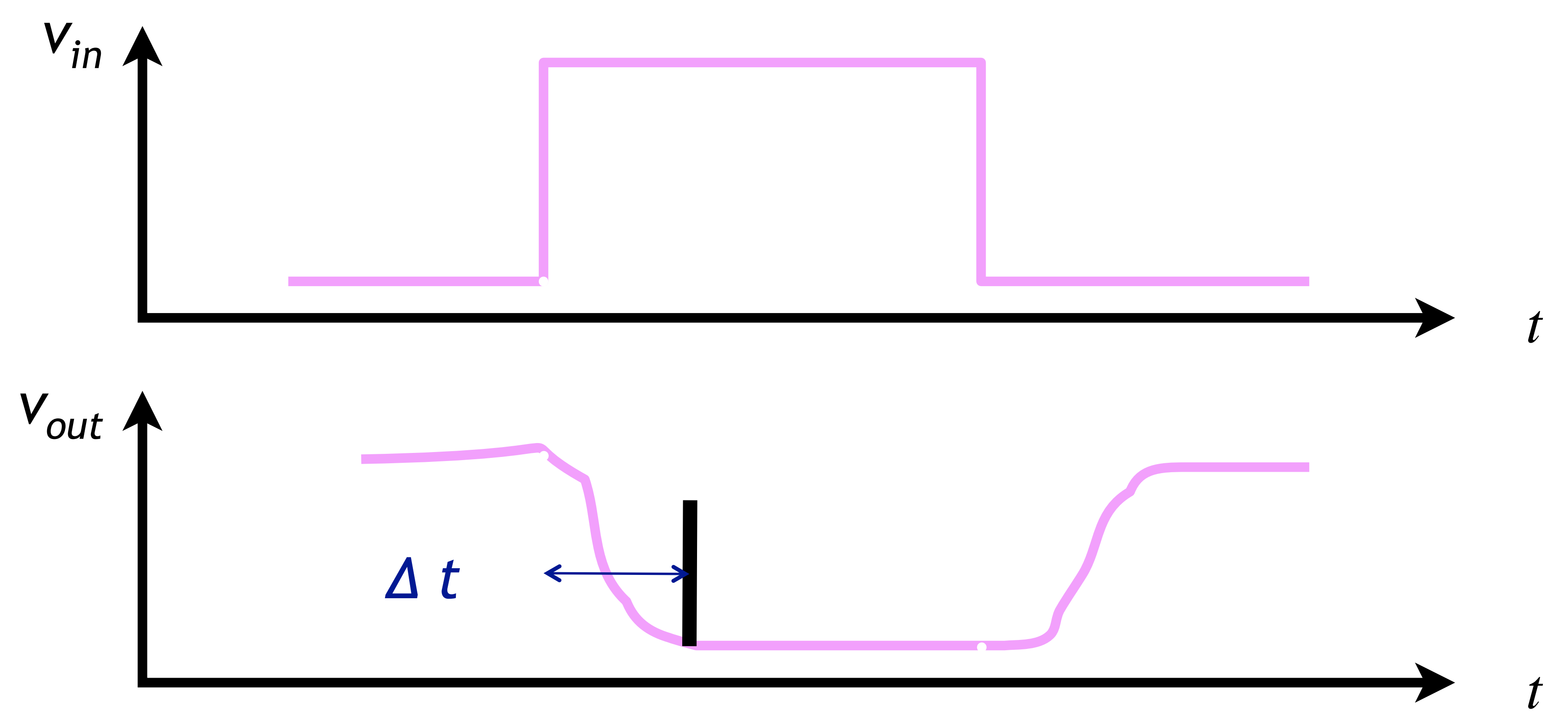
XZ

Simplification Examples(2)

- $F = XY + X'Z + YZ$

Why simplify?

- Circuit cost
  - gates cost money to build
  - gates take time to operate
- Gate Delay
  - time from change at input and stable output
  - NOT gate: 1-5 *ns*, AND/OR gate: 5-10 *ns*
  - Can be in the *ps* range depending on implementation technology
  - Nanoseconds matter. Light travels a full foot in a nanosecond



Find the complement

- $F = X(Y'Z'+YZ)$
- $F' = (X(Y'Z'+YZ))'$ complement both sides

$= X' + (Y'Z'+YZ)'$ demorgan

$= X' + (Y'Z')'(YZ)'$ demorgan

$= X' + (Y+Z)(Y'+Z')$ demorgan

$= X' + YY'+ZY'+YZ'+ZZ'$ distributive

$= X' + ZY' + YZ'$ inverse

Duality

- **The *Duality* principle**: the dual *F** of a function *F* is formed by swapping all AND⇔OR and all 1⇔0
  - If $F=G$, then $F^*=G^*$
  - You can find the complement of a function by finding the dual, and complementing each literal
    - essentially the same as demorgan
  - Calculate truth table, switch 1⇔0

Some Theory

- **Minterm**
  - A **product** term (i.e. AND term) with exactly one literal for each variable in the function
  - e.g. ABC, $\overline{A}BC$, $\overline{A}\overline{B}\overline{C}$…
- **Maxterm**
  - A **sum** term (i.e. OR term) with exactly one literal for each variable in the function
  - e.g. $A+B+C$, $\overline{A}+B+C$, $\overline{A}+\overline{B}+\overline{C}$…
- Note: product = AND, sum = OR but…
  - different from add, multiply, as we've seen

All 2-variable minterms and maxterms

A	B	Minterm	Symbol	Maxterm	Symbol
0	0	$\bar{A}\bar{B}$	m0	A+B	M0
0	1	$\bar{A}B$	m1	$A+\bar{B}$	M1
1	0	$A\bar{B}$	m2	$\bar{A}+B$	M2
1	1	AB	m3	$\bar{A}+\bar{B}$	M3

- Note: $\bar{A}\bar{B}$ = $(A+B)'$ by demorgan
  - in general, $m_i = M_i'$
- Note: binary encoding of variable values give minterm names
  - 00 = 0; 11 = 3

3-variable minterms and maxterms

A	B	C	Minterm	Symbol	Maxterm	Symbol
0	0	0	$\bar{A}\bar{B}\bar{C}$	m_0	$A+B+C$	M_0
0	0	1	$\bar{A}\bar{B}C$	m_1	$A+B+\bar{C}$	M_1
0	1	0	$\bar{A}B\bar{C}$	m_2	$A+\bar{B}+C$	M_2
0	1	1	$\bar{A}BC$	m_3	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$A\bar{B}\bar{C}$	m_4	$\bar{A}+B+C$	M_4
1	0	1	$A\bar{B}C$	m_5	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$AB\bar{C}$	m_6	$\bar{A}+\bar{B}+C$	M_6
1	1	1	ABC	m_7	$\bar{A}+\bar{B}+\bar{C}$	M_7

- Note: minterm names (*e.g.* m_0) are ambiguous unless you know how many variables you have

Take care when naming maxterms

- Minterms are easy to name.
  - Each literal that is positive counts as a 1
  - Each literal that is negated counts as 0
  - Calculate the binary encoding for the minterm name
    - e.g. $\overline{A}B\overline{C}$ = 010 = m2
- Maxterms are opposite
  - Each literal that is positive counts as a 0
  - Each literal that is negated counts as 1
  - eg $(\overline{A}+B+\overline{C})$ = 101 = M6

Canonical Forms

- Any boolean function can be expressed as a **sum of minterms** or a **product of maxterms**
- e.g. $F(X,Y,Z) = X'Y'+Z$

$$= X'Y'(Z+Z') + (X+X')(Y+Y')Z$$

$$= X'Y'Z+X'Y'Z' \quad + \quad XYZ+XY'Z+X'YZ+X'Y'Z$$

$$= X'Y'Z'+X'Y'Z+X'YZ+XY'Z+XYZ$$

$$= \ m0 + m1 + m3 + m5 + m7$$

$$= \sum(m0,m1,m3,m5,m7)$$

$$= \sum m(0,1,3,5,7) \qquad \text{(Sum of minterms 0, 1, 3, 5, 7)}$$

Canonical Forms: product-of-maxterms

- $F(X,Y,Z) = X'Y'+Z$

  $= X'Y'+Z$

  $= (X'+Z)(Y'+Z)$

  $= (X'+YY'+Z)(XX'+Y'+Z)$

  $= (X'+Y+Z)(X'+Y'+Z)(X+Y'+Z)(X'+Y'+Z)$

  $=(X+Y'+Z)(X'+Y+Z)(X'+Y'+Z)$

  $= M_2\, M_4\, M_6$

  $= \prod(M_2, M_4, M_6)$

  $= \prod M(2,\ 4,\ 6)$

  product of maxterms 2,4,6

Duality of Canonical Forms

- Note: $\sum m(0, 1, 3, 5, 7) = \prod M(2, 4, 6)$
- in general,
  - $\Sigma m(\{a\}) = \Pi M(\{b\})$, where
    - $\{a\} \cup \{b\} = \{0, 1, .. 2n-1\}$, and
    - $\{a\} \cap \{b\} = \{\varnothing\}$
    - and n is the number of variables

Canonical Forms and Demorgan

- Also: $\sum m(\{a\}) = (\prod M(\{a\}))'$

  $(\prod M(\{a\}))' = (M_{a1} \bullet M_{a2} \bullet \ldots \bullet M_{ak})'$

  $= M_{a1}' + M_{a2}' + \ldots + M_{ak}'$

  $= m_{a1} + m_{a2} + \ldots + m_{ak}$ because $m_i = M_i'$

  $= \sum m(\{a\})$

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

Standard Forms

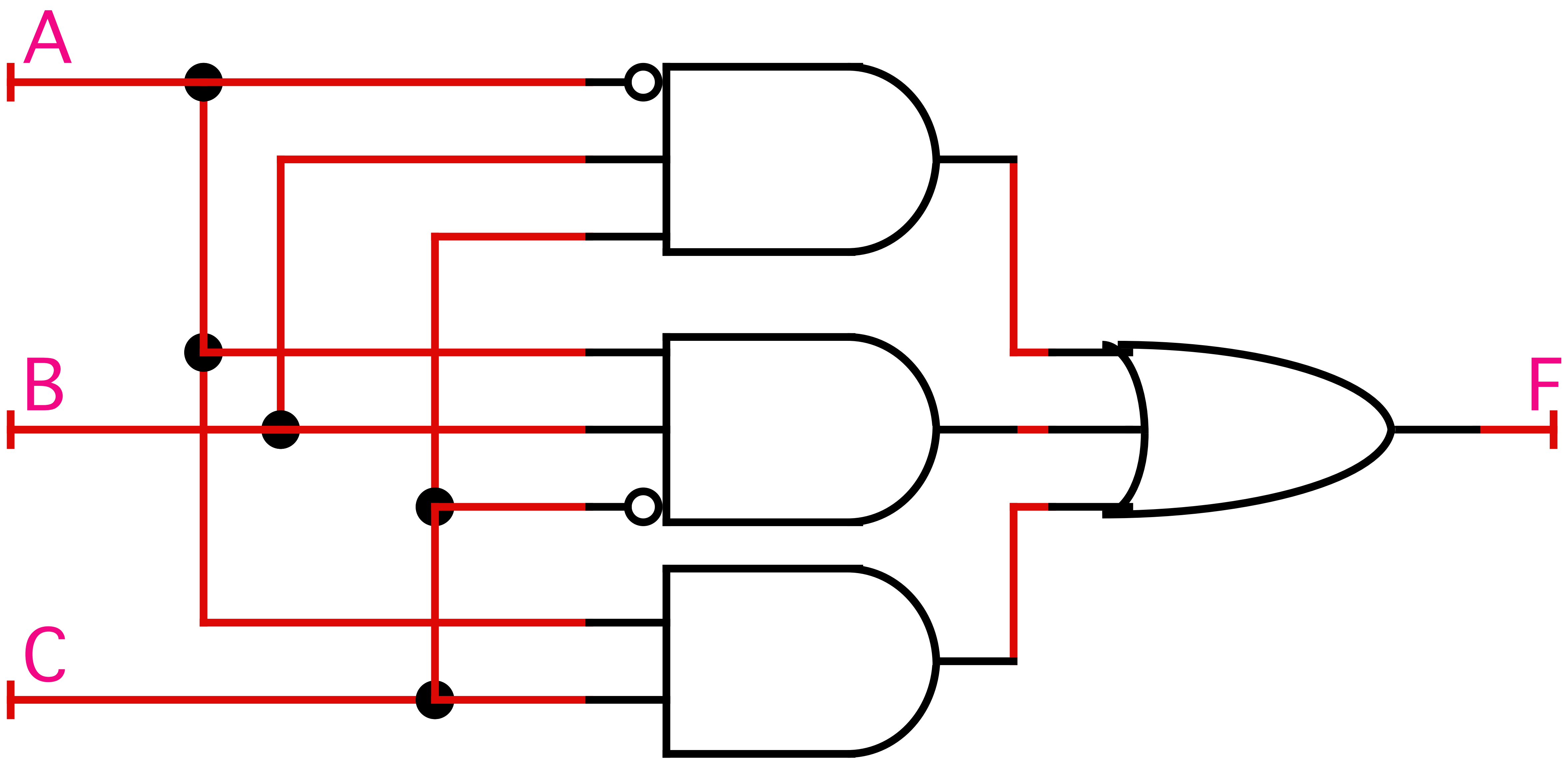
- Sum of Products form (SOP)
  - *e.g.* $F = A(B+C) = AB+AC$
  - can be simpler than canonical (sum of minterms)
  - sum of minterms is an example of SOP
- Product of Sums form (POS)
  - *e.g.* $F = A+BC = (A+B)(A+C)$
  - product of maxterms is an example of POS

Implementation of Standard Forms with gates

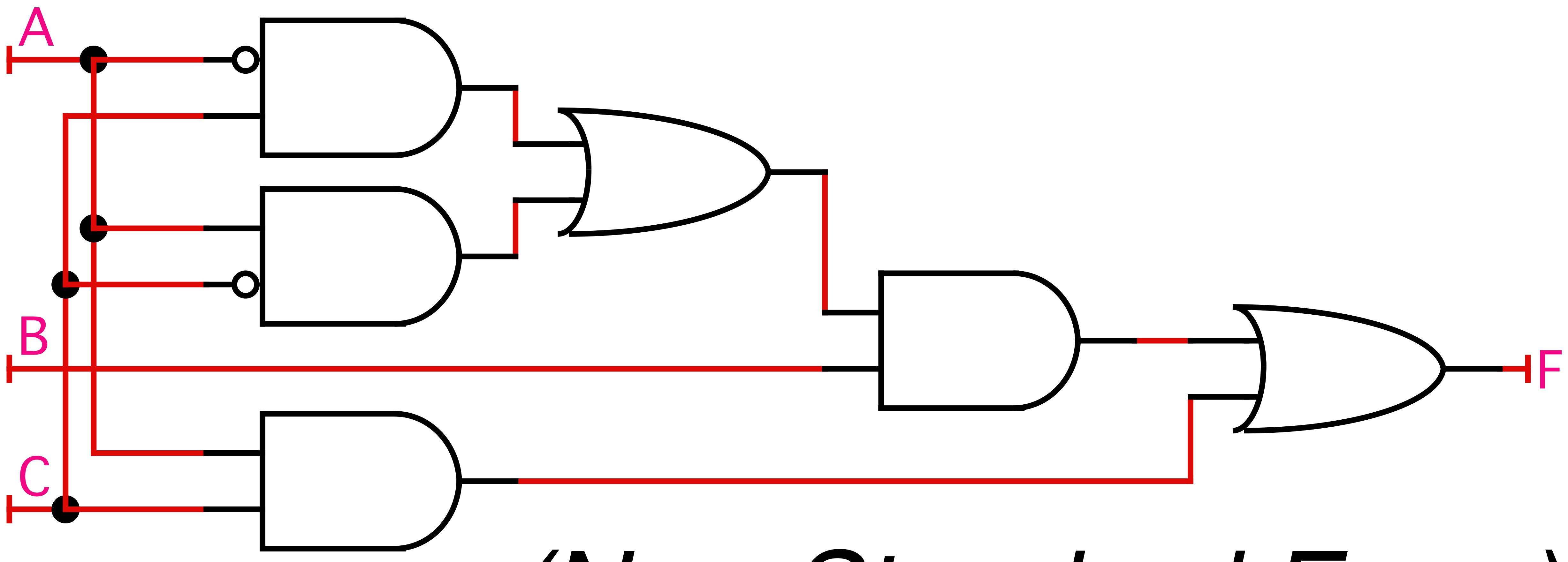
- OR-AND implementation
  - POS can be implemented in two levels
    - sum terms become OR gates (with inverter inputs as necessary
    - one *n*-input AND gate
- AND-OR implementation
  - SOP can be implemented in two levels
    - product terms become AND gates
    - one *n*-input OR gate

Example of AND-OR standard form

- $F = \overline{A}BC + AB\overline{C} + AC$



- Note: this could be simplified to $F = B(\overline{A}C + A\overline{C}) + AC$ but this is not standard form

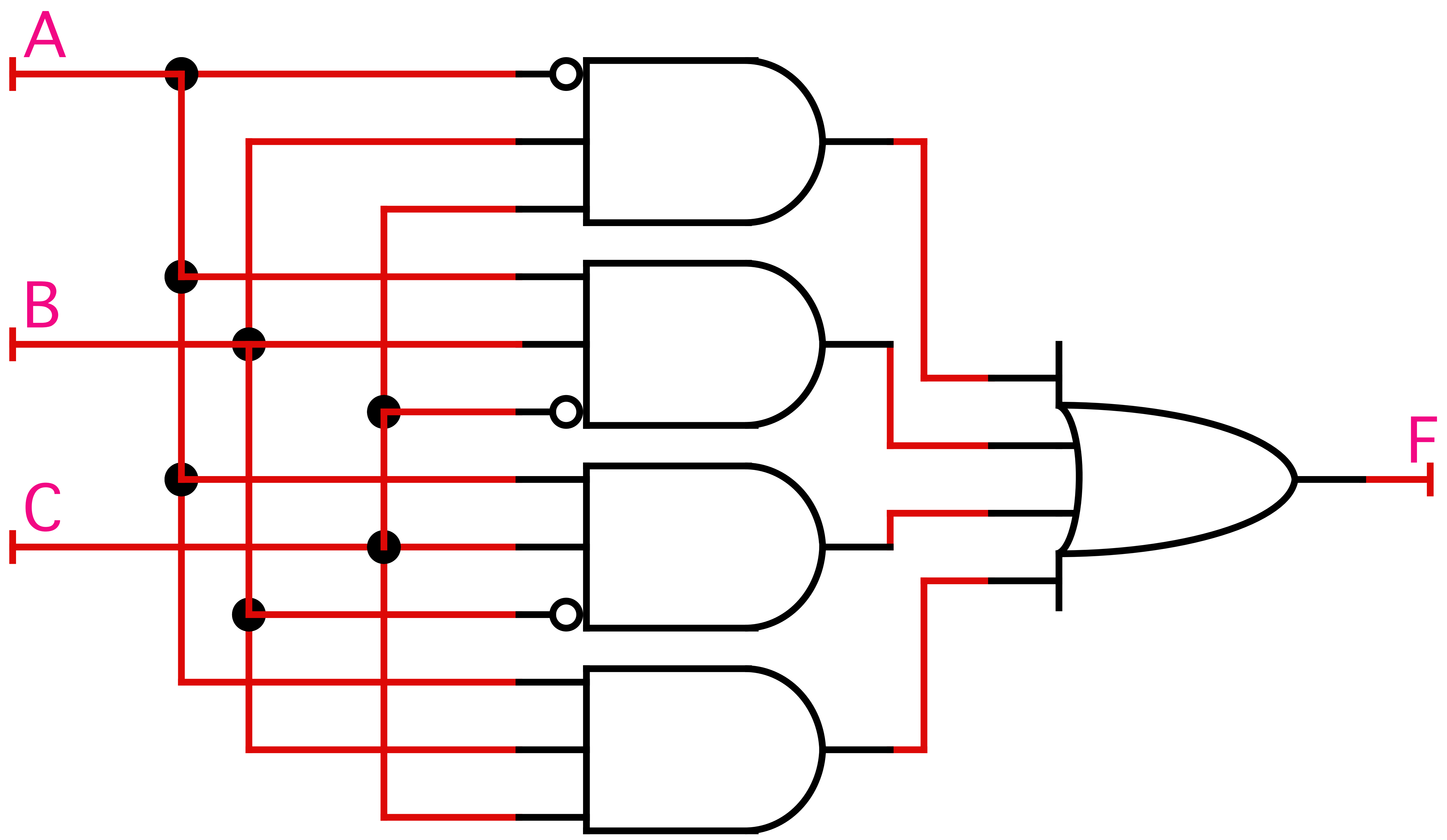


(Non-Standard Form)

Same example in canonical form

- All terms must be minterms

$$F = \overline{A}BC + AB\overline{C} + AC$$
$$= \overline{A}BC + AB\overline{C} + AC(B + \overline{B})$$
$$= \overline{A}BC + AB\overline{C} + ABC + A\overline{B}C$$



Simplifying Standard Forms

- Two-level minimum cost design
  - POS or SOP with minimum number of terms
  - Each term has minimum number of literals
- "best" design (depending on criteria)
- How do we find this?
  - Repeated boolean simplification
    - we must somehow make sure it is the simplest.
- Systematic method to find simplest:
    - ***Karnaugh Maps***

Karnaugh Maps

- Also called k-maps
- Graphical representation of all minterms
  - Map depends on number of variables
- Allows systematic simplification of boolean functions
- Variables are enumerated as both positive (A=1) and negative (A=0),
  - Half are listed vertically, the other half horizontally, making a grid

K-maps: 2 variables

A \ B	0	1
0	m0	m1
1	m2	m3

e.g. $F = \boxed{AB'} + \boxed{A'B}$

A \ B	0	1
0	0	$\boxed{1}$
1	$\boxed{1}$	0

K-maps: 3 variables

- 3-variable map
- Note minterm ordering
- Adjacent cells differ by one variable

A \ BC	00	01	11	10
0	000	001	011	010
1	100	101	111	110

A \ BC	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

A \ BC	00	01	11	10
0	A’B’C’	A’B’C	A’BC	A’BC’
1	AB’C’	AB’C	ABC	ABC’

Adjacent cells differ by one variable value
Alternate forms are also acceptable

B=1

A \ BC	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

C=1

AB \ C	0	1
00	m0	m1
01	m2	m3
11	m6	m7
10	m4	m5

B=1

A=1

Minterms and Maxterms of 4 variables (16 combinations)

A	B	C	D	Minterm	Symbol	Maxterm	Symbol
0	0	0	0	$A'B'C'D'$	m_0	$A+B+C+D$	M_0
0	0	0	1	$A'B'C'D$	m_1	$A+B+C+D'$	M_1
0	0	1	0	$A'B'CD'$	m_2	$A+B+C'+D$	M_2
0	0	1	1	$A'B'CD$	m_3	$A+B+C'+D'$	M_3
0	1	0	0	$A'BC'D'$	m_4	$A+B'+C+D$	M_4
0	1	0	1	$A'BC'D$	m_5	$A+B'+C+D'$	M_5
0	1	1	0	$A'BCD'$	m_6	$A+B'+C'+D$	M_6
0	1	1	1	$A'BCD$	m_7	$A+B'+C'+D'$	M_7
1	0	0	0	$AB'C'D'$	m_8	$A'+B+C+D$	M_8
1	0	0	1	$AB'C'D$	m_9	$A'+B+C+D'$	M_9
1	0	1	0	$AB'CD'$	m_{10}	$A'+B+C'+D$	M_{10}
1	0	1	1	$AB'CD'$	m_{11}	$A'+B+C'+D'$	M_{11}
1	1	0	0	$ABC'D'$	m_{12}	$A'+B'+C+D$	M_{12}
1	1	0	1	$ABC'D$	m_{13}	$A'+B'+C+D'$	M_{13}
1	1	1	0	$ABCD'$	m_{14}	$A'+B'+C'+D$	M_{14}
1	1	1	1	$ABCD$	m_{15}	$A'+B'+C'+D'$	M_{15}

K-maps: 4 variables

C=1

AB \ CD	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

B=1

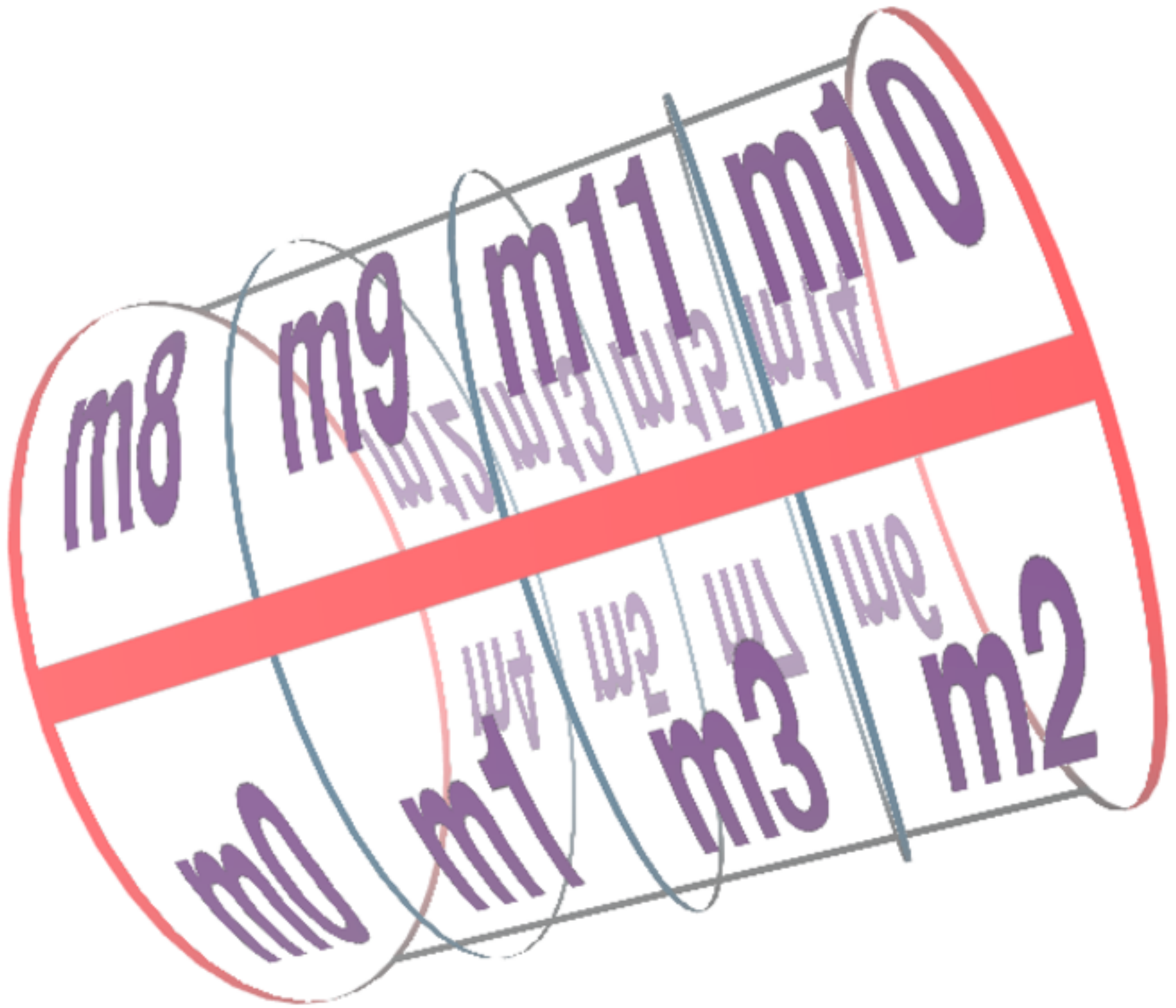
A=1

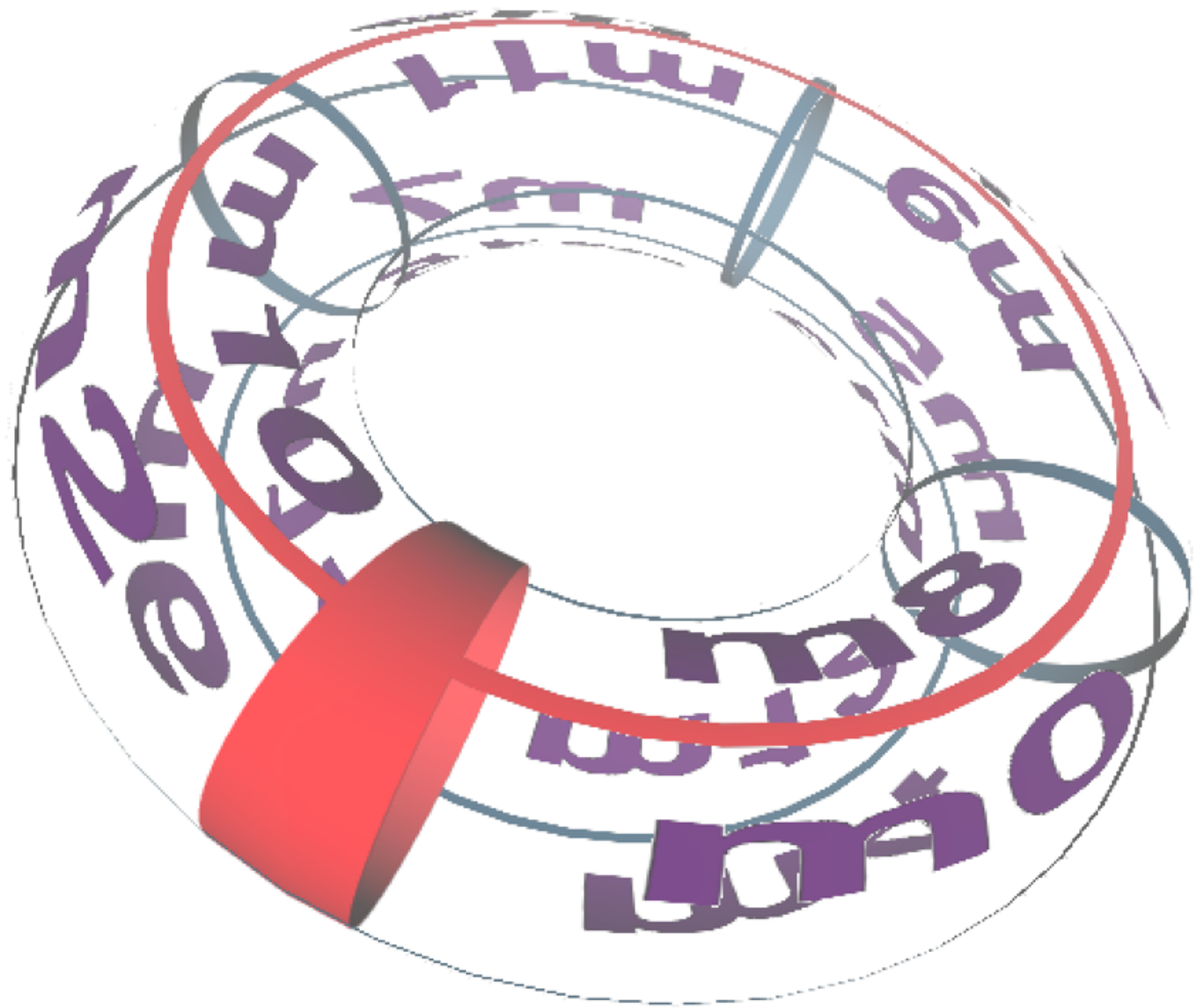
D=1

A k-map is a torus

m0	m1	m3	m2
m4	m5	m7	m6
m12	m13	m15	m14
m8	m9	m11	m10

- m0 is adjacent to m1, m2, m4, and m8
- m15 is adjacent to m7, m13, m14, and m11





K-map simplification

- Adjacent cells differ by one variable
  - Up, down, left, right
  - Wrap around
  - Not diagonally

A \ BC	00	01	11	10
0	1	0	0	1
1	1	1	0	0

- Simplify: grouping cells
  - groups of cells = groups of related minterms
  - simplify minterms in a standard way.

K-map simplification

- Example: $F(A,B,C)=\sum m(0,1)$

  $= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C$

  $= \overline{A}\,\overline{B}(\overline{C}+C)$

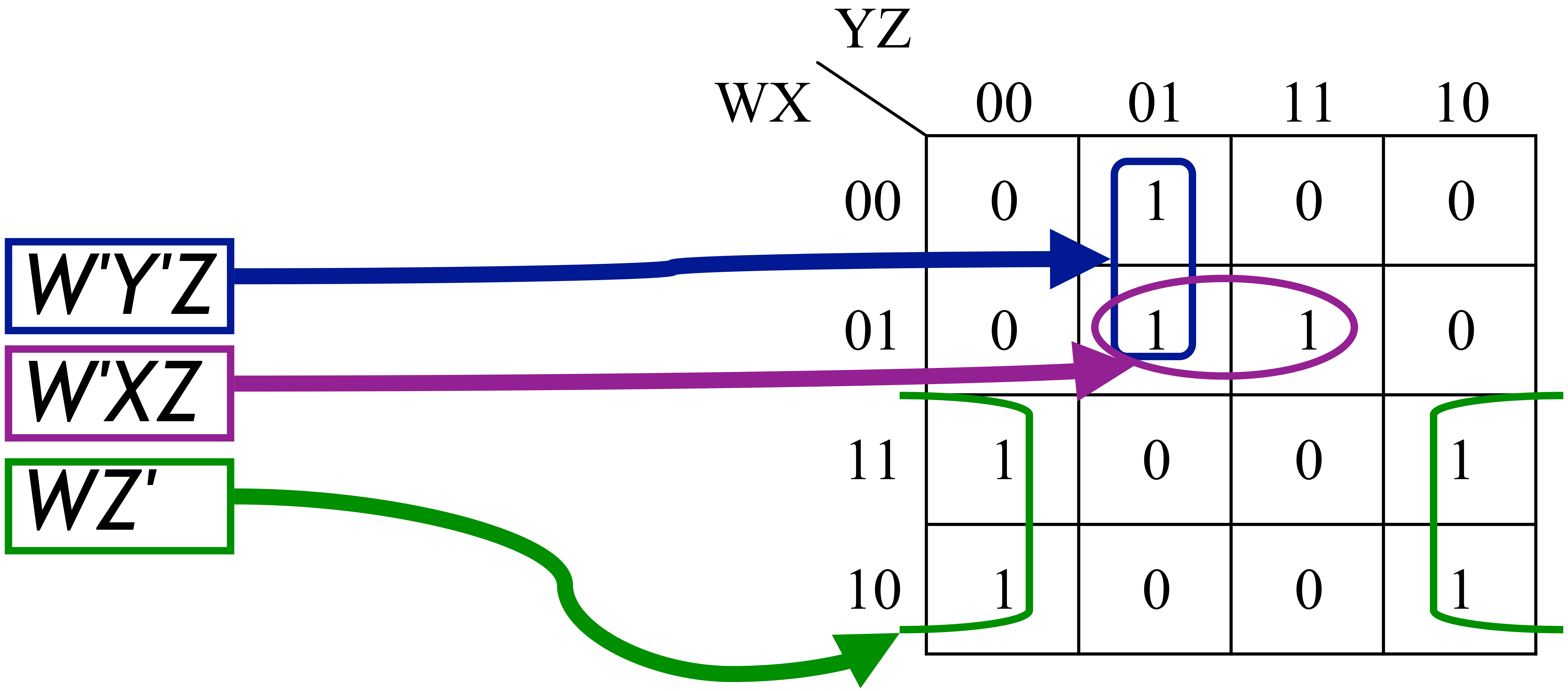
  $= \overline{A}\,\overline{B}$

A \ BC	00	01	11	10
0	1	1	0	0
1	0	0	0	0

- Note variable values
  - for the group:
    - only *C* is different between these two minterms,
    - so *C* disappears from the product term.
    - Uses distributive rule for each group.

K-map simplification

- $F(W,X,Y,Z)=\sum m(1,5,7,8,10,12,14)$



$= W'Y'Z+W'XZ+WZ'$

Essential Prime Implicants

- an *Implicant* is another name for a group of 1s on a map
  - must be a power of 2 (i.e. 1, 2, 4, 8, or 16 terms), and in a rectangular shape
- a ***Prime Implicant*** is an implicant that is as large as possible
- an ***Essential Prime Implicant*** is a prime implicant that contains at least one term not covered by another Prime Implicant

Example (1)

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	0	1	1	1
10	0	1	1	0

Essential Prime Implicants

Non-Essential Prime Implicant

Non-prime Implicant

Example (2)

indicate all essential prime implicants

AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	1	1
10	0	0	1	1

k-map procedure

- draw the map
- write "1" for each minterm in the function
- fill the rest with "0"
- find essential prime implicants
- choose prime implicant(s) to cover remaining "1"s
- write corresponding terms in SOP form

AB \ CD	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	0	0	1	1
10	0	1	0	1

$F = BC + A'C'D + B'C'D + ACD'$

also valid:

$F = BC + A'BD + B'C'D + ACD'$

Example

- $F(A,B,C) = \Sigma m(1, 2, 3, 4, 5)$

Two more examples

$F(X,Y,Z) = \Sigma m(3,4,6)$ $G(X,Y,Z)=\Sigma m(0,2,4,5,6)$

New Feature: “*Don’t Care*” Conditions

- A “don’t care” is a minterm for which the function is ***undefined***, or when we don’t care what the result is
- Often used when that input combination is impossible
  - so the output can be unspecified.
- e.g. 4-bit binary code for a decimal digit
  - Minterms 0-9 would valid; 10-15 would be invalid
- Notation: F = ∑m(…) + d(…)
  - ‘x’ indicates a don’t care in a k-map or truth table
- In a k-map, can be in a group or not.
  - Use selectively to make biggest groups

Don’t Care Conditions: example

$F(W,X,Y,Z) = \sum m(1,3,7,11,15) + d(0,2,5)$

WX \ YZ	00	01	11	10
00	x	1	1	x
01	0	x	1	0
11	0	0	1	0
10	0	0	1	0

$F = \overline{W}\,\overline{X} + YZ$

$\overline{W}Z$ is an Alternative to $\overline{W}\,\overline{X}$

Simplifying in POS form

- All k-maps so far have been in SOP form
- k-maps can be used to find POS form as well
- Find the form for the ***complement*** of the map
  - Recall $\sum m(\{a\}) = (\prod M(\{a\}))'$
  - Group the ***zeros*** instead of the ones, then complement the result

X \ YZ	00	01	11	10
0	1	0	0	1
1	1	1	0	1

$$G' = X'Z + YZ$$

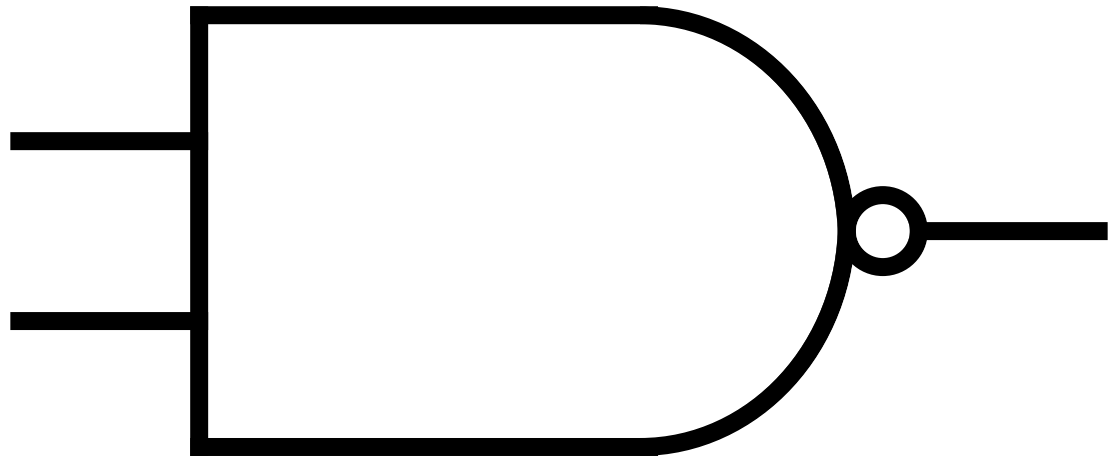
$$G = (X'Z+YZ)'$$

$$= (X+Z')(Y'+Z') \rightarrow POS$$

More gates

NAND

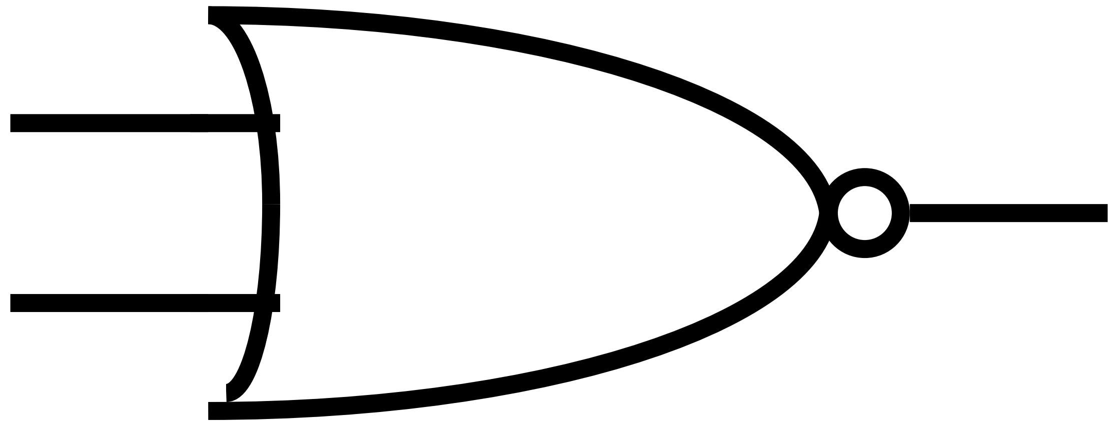
$F = (XY)'$



X	Y	(XY)'
0	0	1
0	1	1
1	0	1
1	1	0

NOR

$F = (X+Y)'$



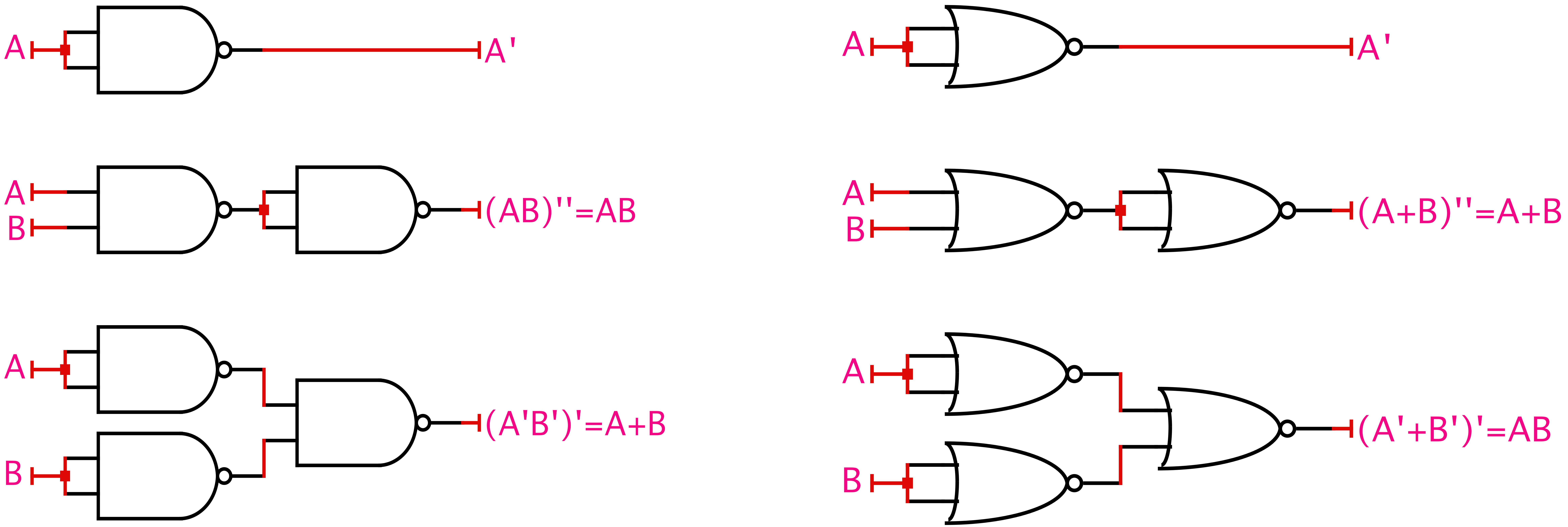
X	Y	(X+Y)'
0	0	1
0	1	0
1	0	0
1	1	0

Universal Gate

- A universal gate is a gate which (in combination with copies of itself) can implement AND, OR, or NOT
- All circuits can be represented in standard form (SOP)
  - so all circuits can be build with •, +, $\overline{}$
  - so a universal gate can implement any circuit

NAND and NOR are both universal

If all you have is a bucket of NAND gates (or a bucket of NOR gates), you can build any circuit

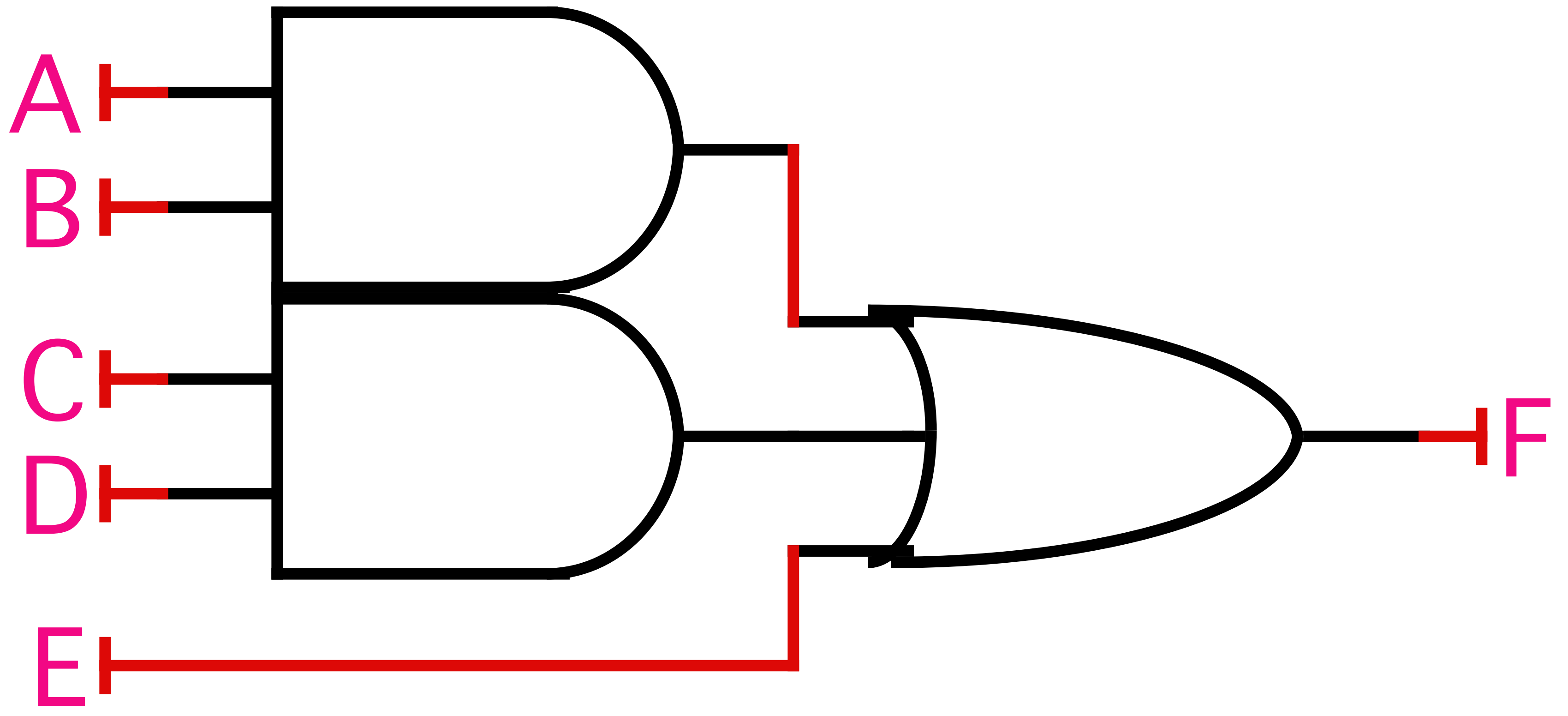


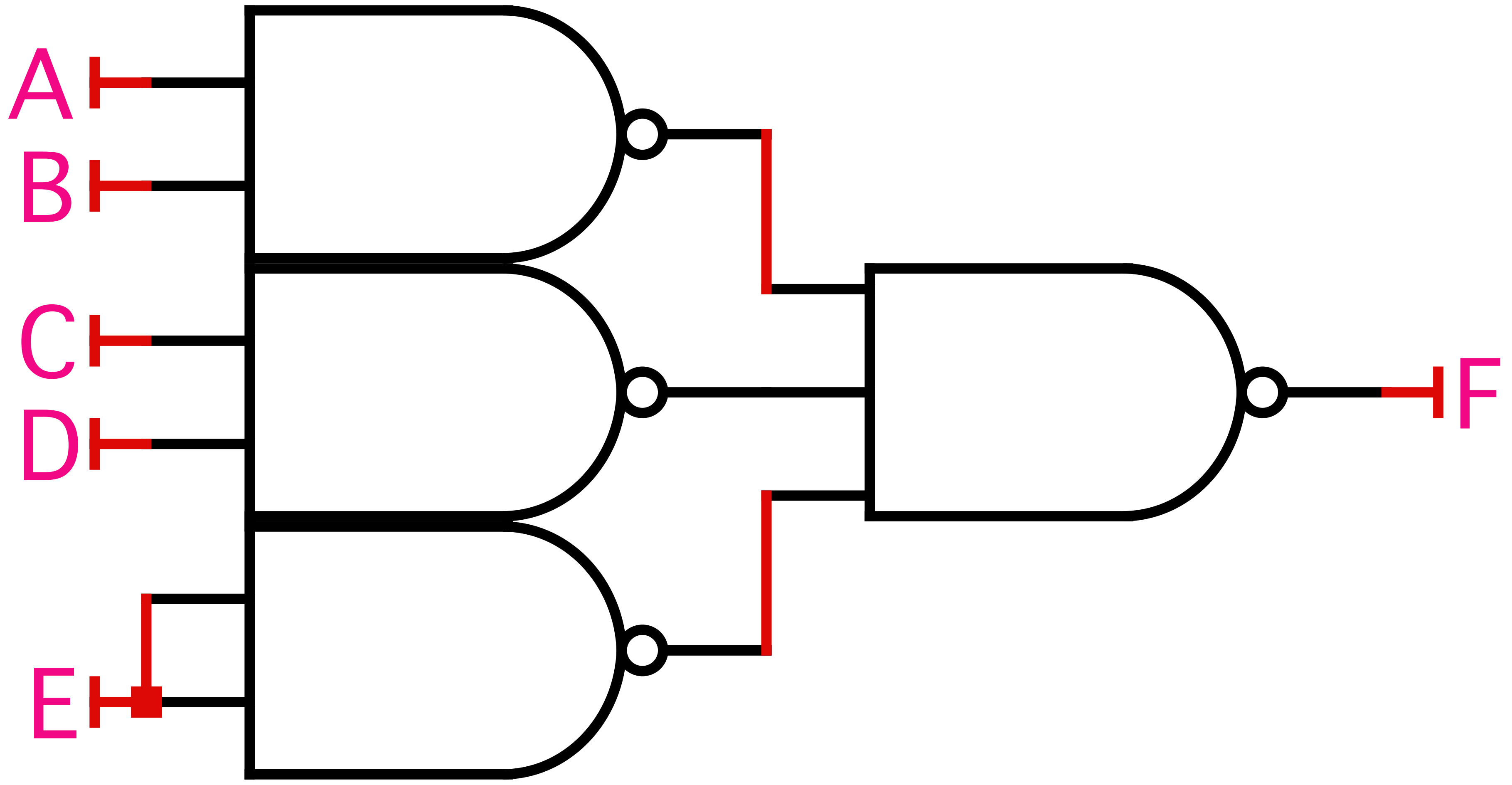
NAND and NOR implementations of circuits

- Implement a circuit with only NAND (or NOR)
  - Function-based:
    - use demorgan to move from AND-OR to NAND
    - e.g. AB+CD = (AB+CD)'' = ((AB)'(CD)')'
  - Circuit-based
    - replace all gates with NAND gates and undo any complements caused by the replacement

NAND implementation: Function-based

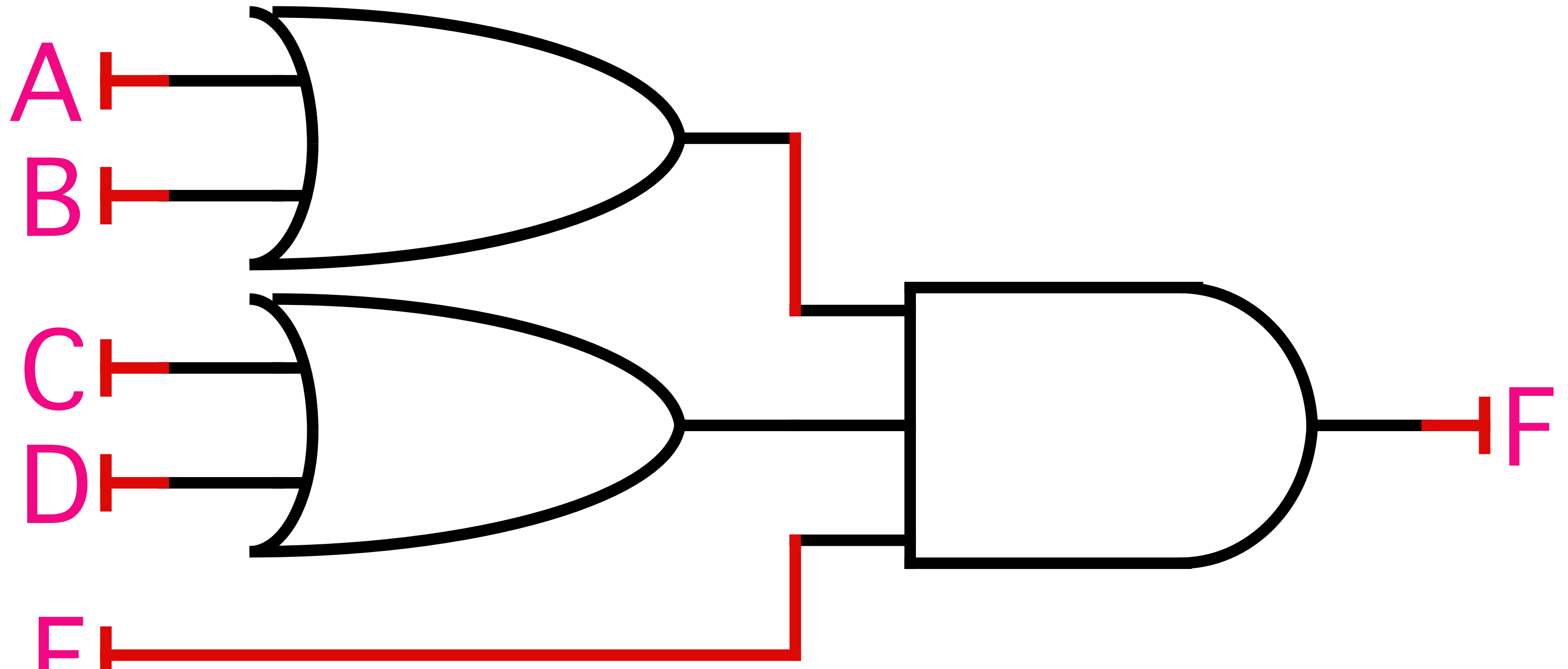
- e.g. F=AB+CD+E = $((AB)'\,(CD)'\,E'\,)'$

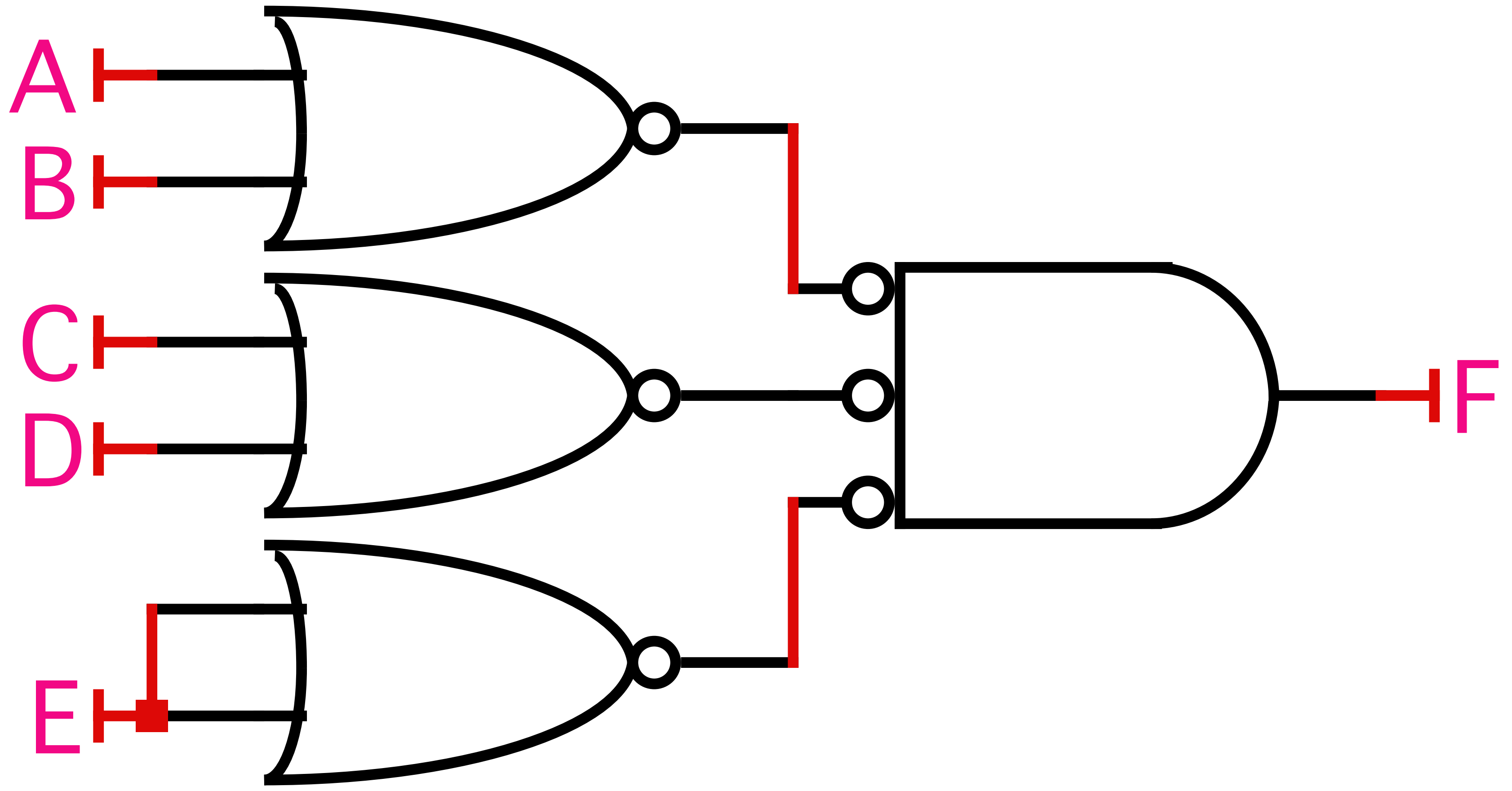




NOR implementation: Function-based

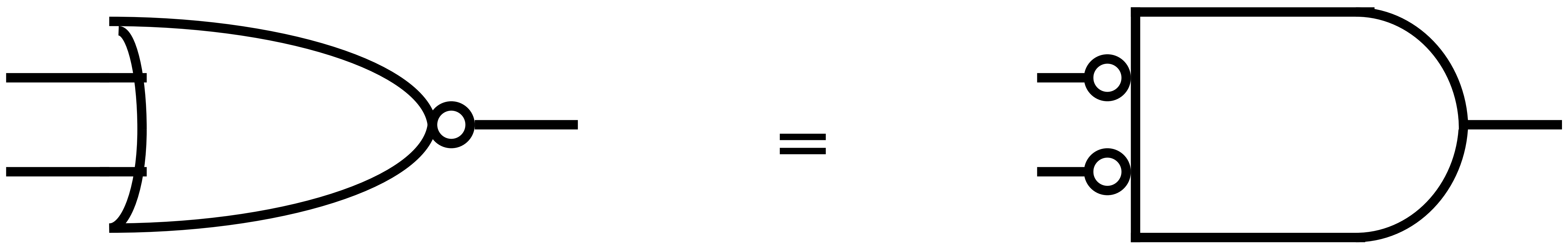
- From a POS form
- e.g. F = (A+B)(C+D)E = ((A+B)'+(C+D)'+E')'





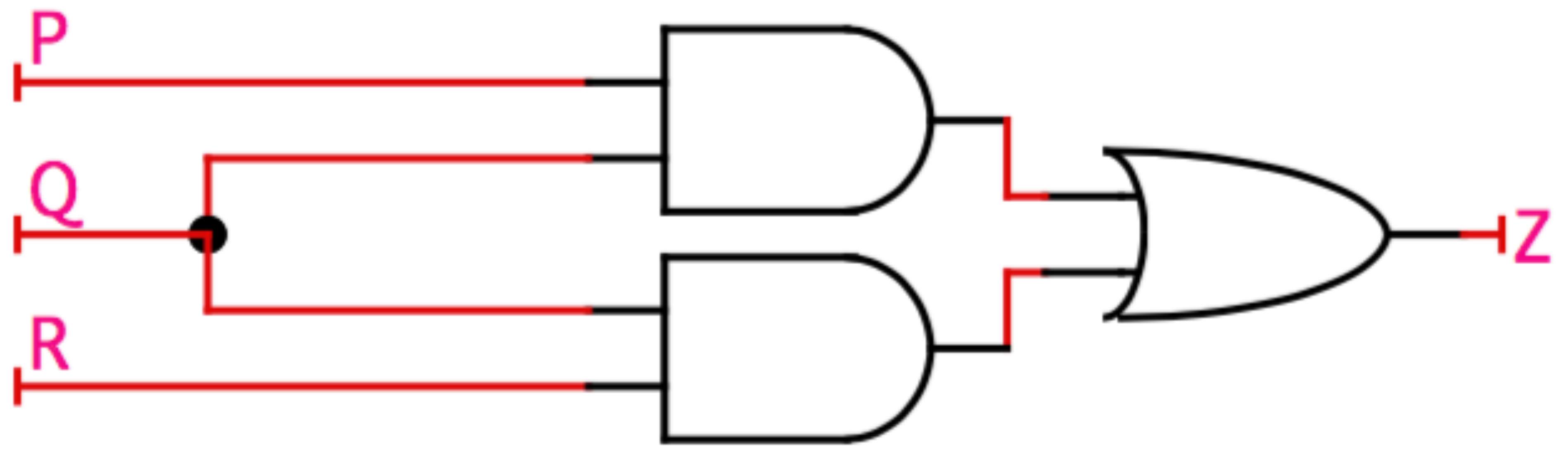
NOR implementation

- Note the gate equivalence:

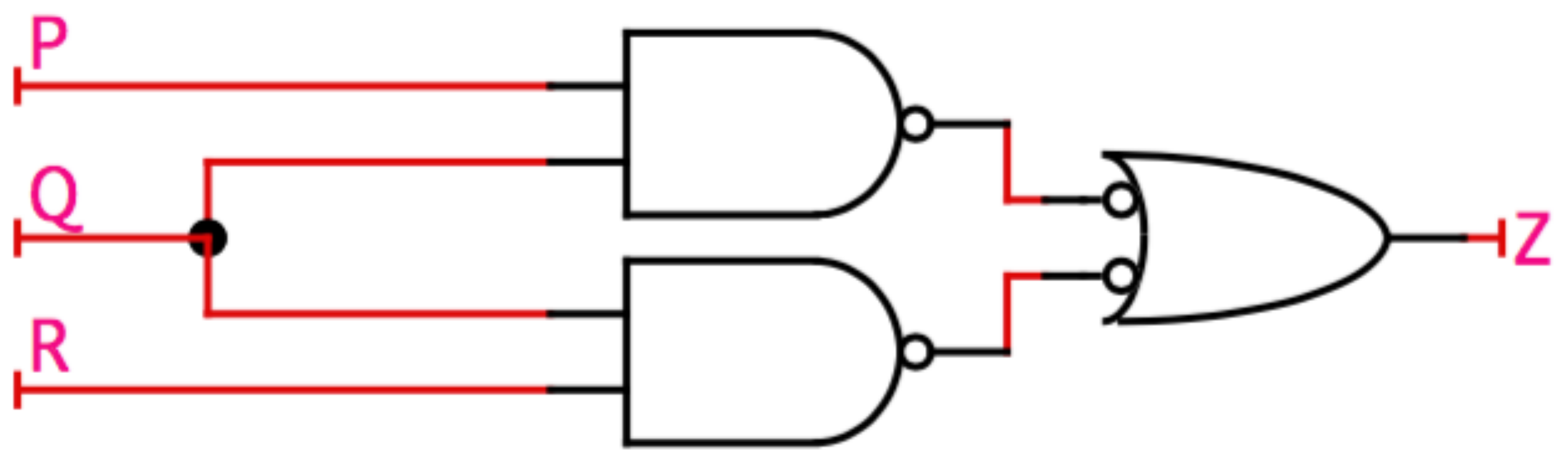


- This is a circuit-level implementation of demorgan's law: (A+B)' = A'B'
- "bubbles" represent inversions
- When replacing gates, two "bubbles" cancel out.

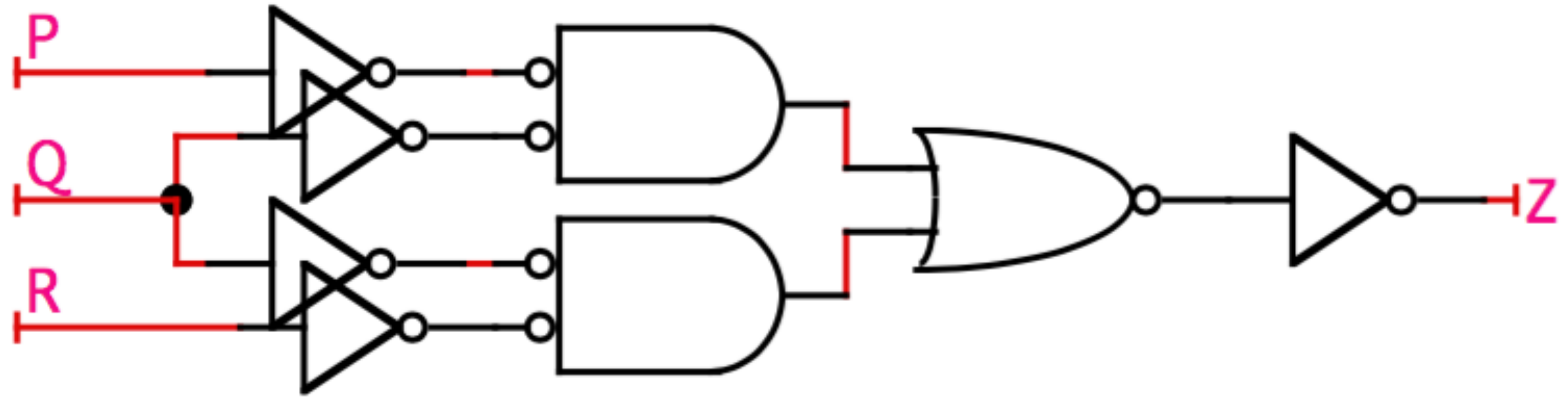
NAND / NOR implementation: Circuit-based



- Original Circuit: SOP
  - Z=PQ+QR



- NAND implantation of SOP
  - replace gates with NAND
  - added bubbles cancel out



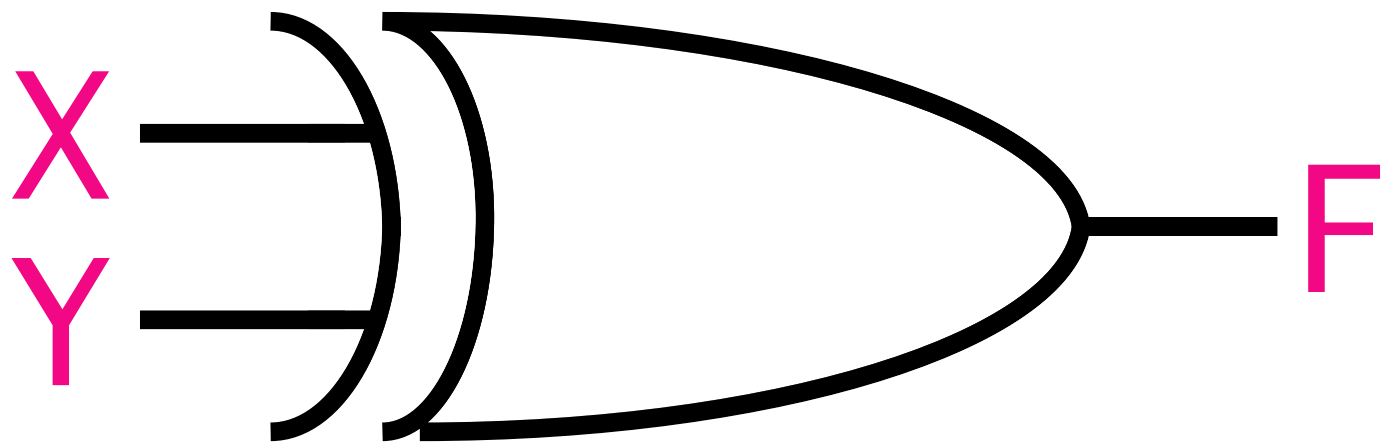
- NOR implementation of SOP
  - replace gates with NOR
  - added bubbles must be negated

More gates

XOR

$F = XY' + X'Y$

$= X \oplus Y$

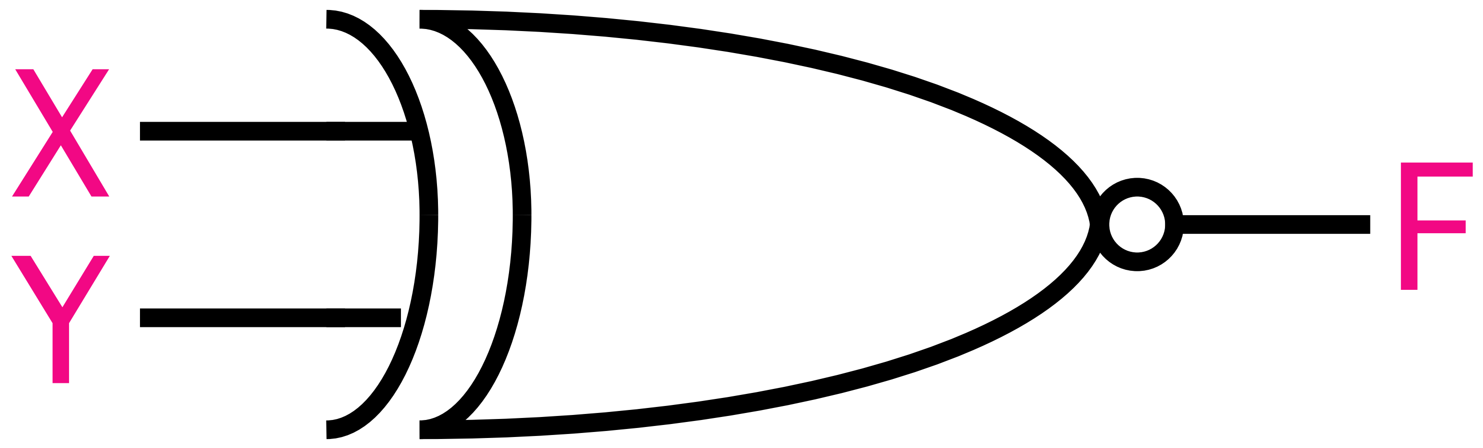


X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

$F = XY + X'Y'$

$= (X \oplus Y)'$

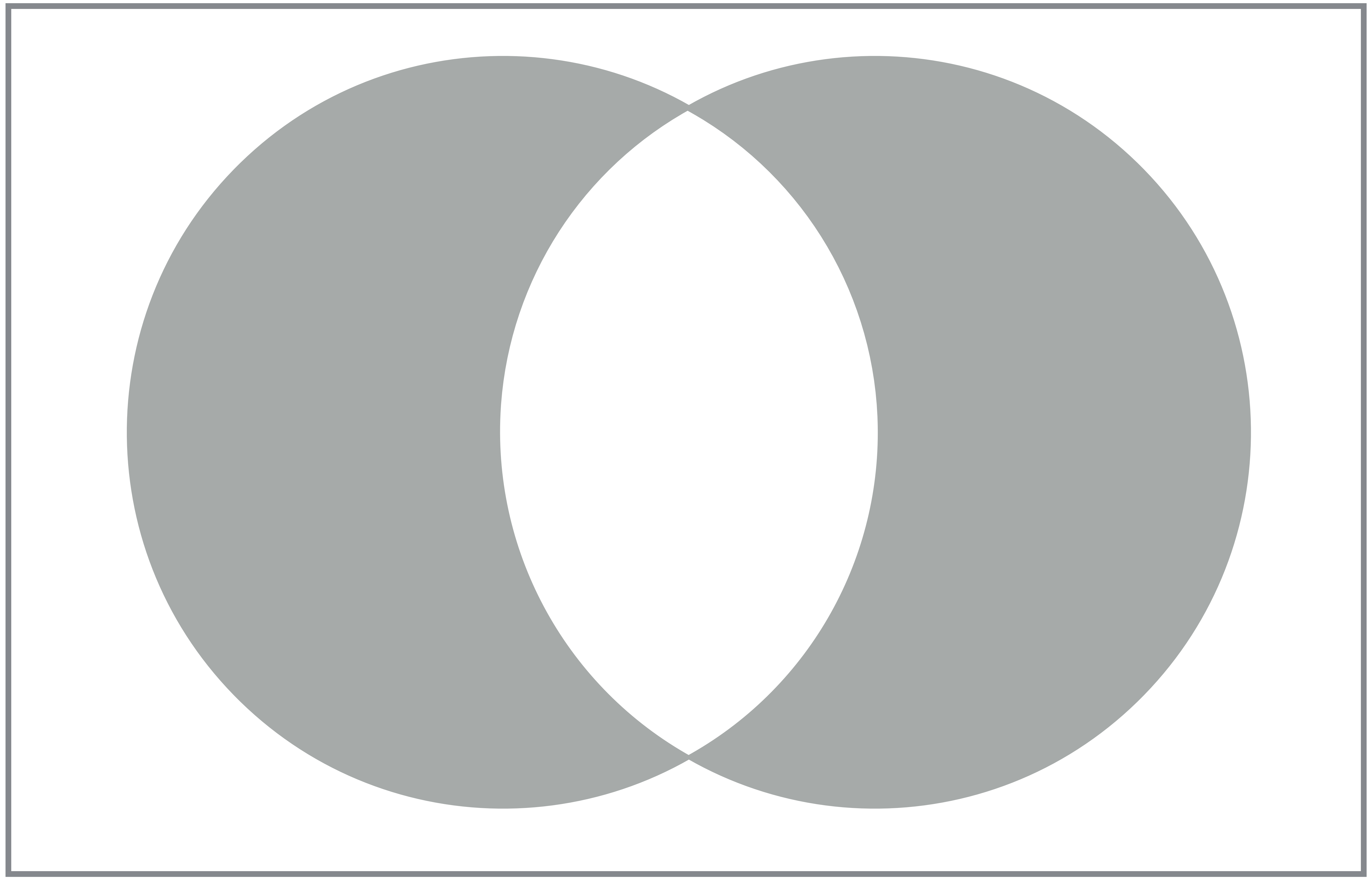


X	Y	$(X \oplus Y)'$
0	0	1
0	1	0
1	0	0
1	1	1

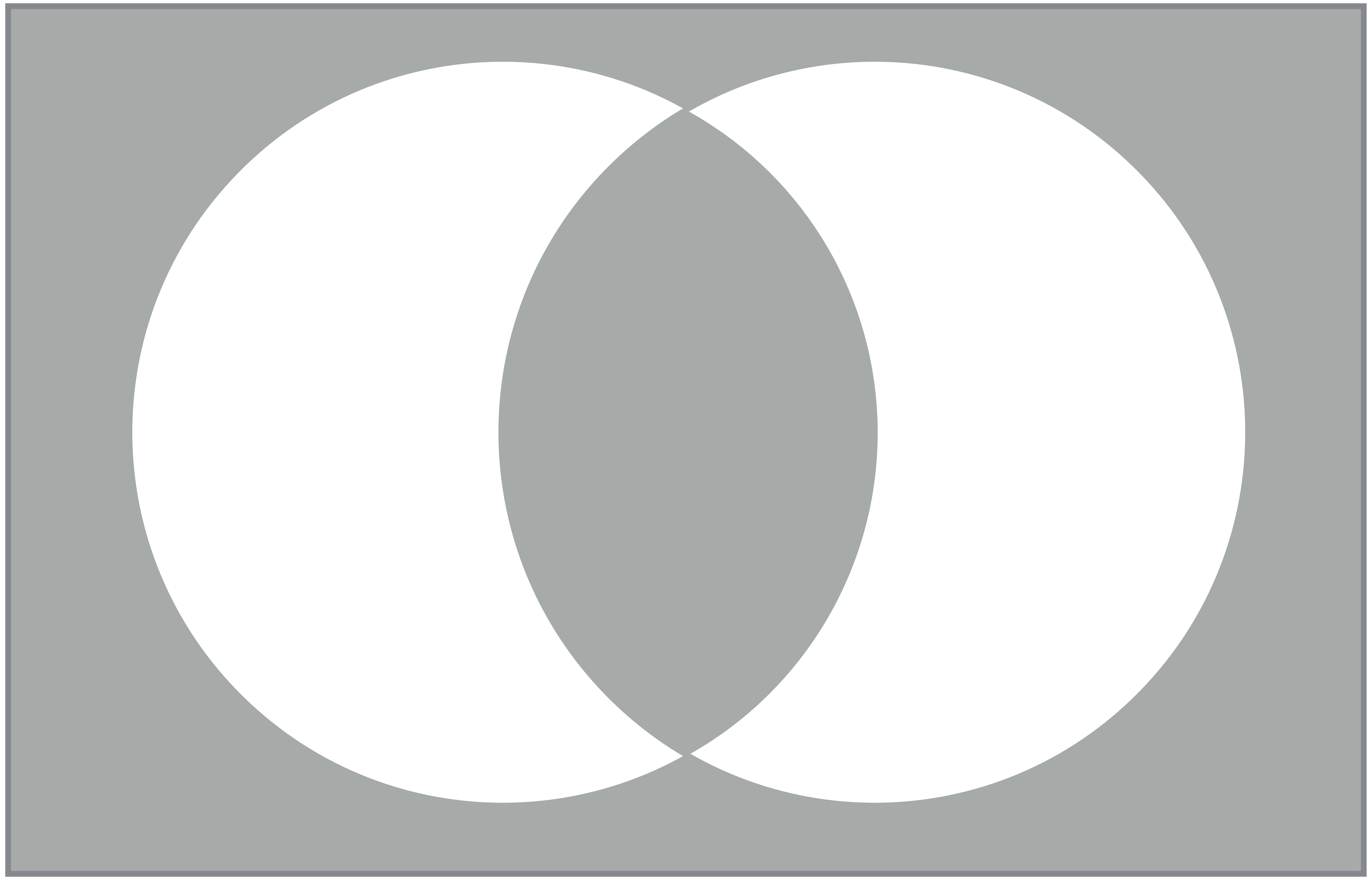
XOR and XNOR

- Prove that $(X\overline{Y} + \overline{X}Y)' = XY + \overline{X}\,\overline{Y}$

Exclusive-OR and Exclusive-NOR Venn Diagram



$A \oplus B$



$(A \oplus B)'$

XOR and parity

- ***Parity*** refers to the number of 1s (or 0s) in a string of bits
- A bitstream exhibits ***even parity*** if there is an even number of 1s, and ***odd parity*** if there is an odd number of 1s
- XOR can be used to calculate the parity of a bitstream
    - XOR will be true if there is an odd number of 1s
    - XNOR will be true if there is an even number of 1s

Error Checking using Parity

- You can enforce even parity by adding a single bit to the bitstream
  - Set to 1 or 0 to make total number of 1s even
  - Odd parity - make an odd number of 1s
- Parity bit generator
  - Given a bit string, generate the parity bit
- Parity bit checker
  - Given a bit string with parity, verify that the bit stream is correct.

Error Checking using Parity

- Parity verification
  - Will only detect an odd number of errors.
    - if there are, say, two errors, they will "cancel out"
  - e.g.: 3 bit message, even parity
    - $P = A1 \oplus A2 \oplus A3$ $\quad$ $C = A1 \oplus A2 \oplus A3 \oplus P$