14:332:231
DIGITAL LOGIC DESIGN

Ivan Marsic, Rutgers University

Electrical & Computer Engineering

Fall 2013

Lecture #6: Combinational Circuit Synthesis I

Combinational Circuit Synthesis

- Recall:
  - ***Combinational circuit analysis***: we are given a logic diagram and need to find its formal description (truth table, logic expression)
- ***Combinational circuit synthesis***: we are given a formal description (truth table, logic expression) and need to find its logic diagram
  - (Reverse from analysis)
  - A circuit ***realizes*** ("makes real") an expression if its output function equals the expression, and the circuit is called a ***realization*** of the function

Some Definitions (from Lecture #4)

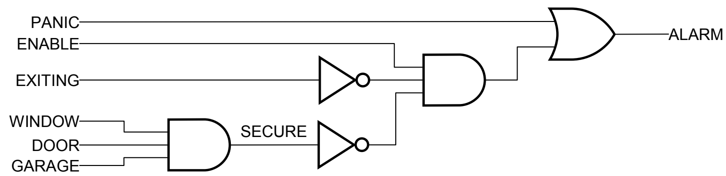
- *Literal*: a variable or its complement
  - X, X′, FRED′, CS_L
- *Expression*: literals combined by AND, OR, parentheses, complementation
  - X + Y
  - P · Q · R
  - A + B · C
  - ((FRED · Z′) + CS_L · A · B′ · C + Q5) · RESET′
- *Equation*: Variable = Expression
  - P = ((FRED · Z′) + CS_L · A · B′ · C + Q5) · RESET′

Combinational Circuit Design

- Sometimes you can write an equation or equations directly using “logic”.
- Example: Given the alarm problem (Eq.1)

ALARM = PANIC + ENABLE · EXITING′ · SECURE′
SECURE = WINDOW · DOOR · GARAGE
ALARM = PANIC + ENABLE · EXITING′ · (WINDOW · DOOR · GARAGE)′

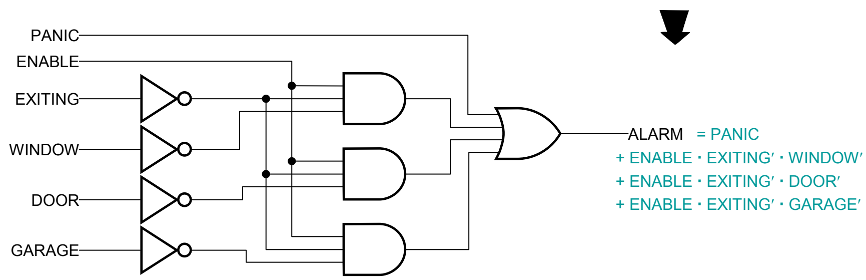
- Find the corresponding circuit:



Alarm-Circuit Transformation

- Sum-of-products form
  - Useful for programmable logic devices
- "Multiply out" the original expression (Eq.1):

(WINDOW · DOOR · GARAGE)′ = WINDOW′ + DOOR′ + GARAGE′



More Definitions (Wakerly, Sec.4.1.6)

- *Product term*: X, X′·Y, X·Y′·Z
- *Sum-of-products expression*: X + X·W′
- *Sum term*: X, X′+Y, X+Y′+Z
- *Product-of-sums expression*: (X′+Y) · (Y+Z)
- *Normal term* -- a product or sum such that no variable appears ≥1 times
- *Minterm* (*n* variables) -- a normal product term with *n* literals
  - 2^n terms, e.g., X·Y′·Z (*n*=3) --- AND terms with every variable present in either true or complemented form
  - is "1" in a given row of the truth table
- *Maxterm* (*n* variables) -- a normal sum term with *n* literals
  - 2^n terms, e.g., X′+Y′+Z --- OR terms with every variable in true or complemented form
  - is "0" in a given row of the truth table

Truth Table vs. Minterms & Maxterms

Row	X	Y	Z	F	Minterm	Maxterm
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

Canonical Sums and Products

- **Canonical sum**: is the sum of the *minterms* corresponding to the truth-table rows of values "1".

  $F = \Sigma_{X,Y,Z}\,(1,2,5,7)$ minterm list

  $= X'{\cdot}Y'{\cdot}Z + X'{\cdot}Y{\cdot}Z' + X{\cdot}Y'{\cdot}Z + X{\cdot}Y{\cdot}Z$
- **Canonical product**: is the product of the *maxterms* corresponding to the truth-table rows of values "0".

  $F = \Pi_{X,Y,Z}\,(0,3,4,6)$ maxterm list

  $= (X+Y+Z) \cdot (X+Y'+Z') \cdot (X'+Y+Z) \cdot (X'+Y'+Z)$
- The two descriptions are in fact the same. The relation between the minterm and the maxterm lists is

  e.g.

  $$\Sigma_{X,Y,Z}\,(1,2,5,7) = \Pi_{X,Y,Z}\,(0,3,4,6)$$

Converting Between Minterm and Maxterm Lists

1*1 = {5,7}   *01 = {1,5}   010 = {2}

$F = X \cdot Z + Y' \cdot Z + X' \cdot Y \cdot Z'$ or   (first)

$F = (X + Y' + Z') \cdot (X' + Z) \cdot (Y + Z)$ (second)

011 = {3}   1*0 = {4,6}   *00 = {0,4}

Row	X	Y	Z	F(first)	F(second)
0	0	0	0	0	**0**
1	0	0	1	**1**	1
2	0	1	0	**1**	1
3	0	1	1	0	**0**
4	1	0	0	0	**0**
5	1	0	1	**1**	1
6	1	1	0	0	**0**
7	1	1	1	**1**	1
				minterms	maxterms

different circuit but the same function

Brute-Force Design

- Truth table → canonical sum (sum of minterms)
- Example: prime-number detector
  - 4-bit input: $N_3N_2N_1N_0$

$F = \Sigma_{N_3N_2N_1N_0}(1,2,3,5,7,11,13)$

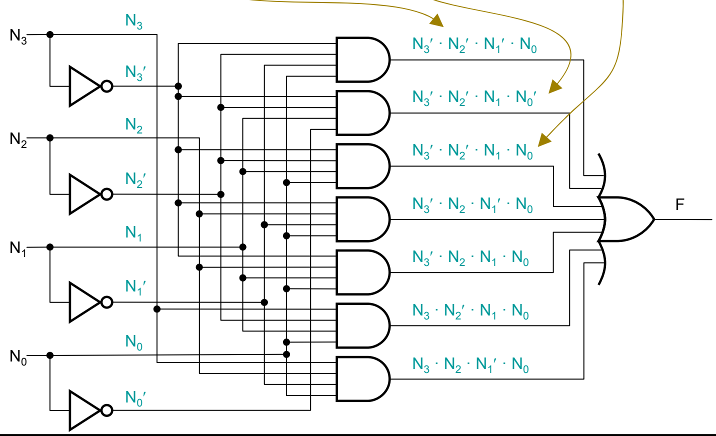
row	N_3	N_2	N_1	N_0	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

Minterm List → Canonical Sum

$F = \sum_{N_3,N_2,N_1,N_0}(1,2,3,5,7,11,13)$

$= N_3'\cdot N_2'\cdot N_1'\cdot N_0 + N_3'\cdot N_2'\cdot N_1\cdot N_0' + N_3'\cdot N_2'\cdot N_1\cdot N_0 + N_3'\cdot N_2\cdot N_1'\cdot N_0$

$+ N_3'\cdot N_2\cdot N_1\cdot N_0 + N_3\cdot N_2'\cdot N_1\cdot N_0 + N_3\cdot N_2\cdot N_1'\cdot N_0$



Combinational Circuit Minimization

- ***Minimize*** a combinational circuit by reducing the number and size of gates needed to build it:
  1. By minimizing the number of first-level gates
  2. By minimizing the number inputs on each first-level gate
  3. By minimizing the number inputs on the second-level gate
     - This is a side effect of the first reduction
- Most minimization methods based on a generalization of the *Combining theorems* (T10) and (T10′)

Algebraic Simplification

- Combining theorem (T10) $X \cdot Y + X \cdot Y' = X$

$F = \sum_{N_3,N_2,N_1,N_0}(1,2,3,5,7,11,13)$

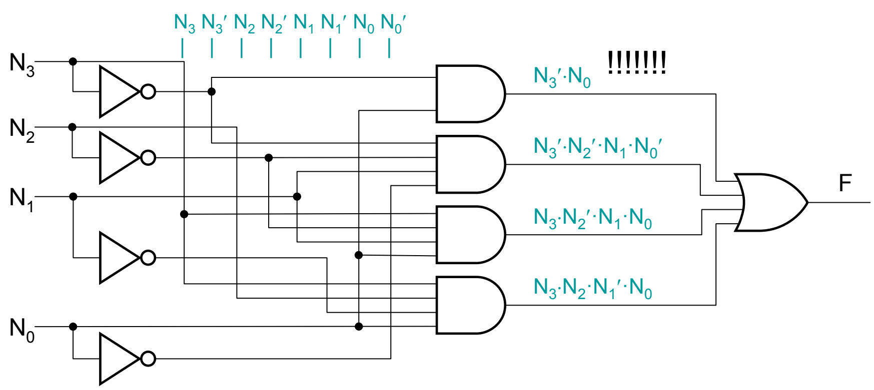
$= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + \ldots$

$= (N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0) + (N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0) + \ldots$

$= N_3' \cdot N_2' \cdot N_0 + N_3' \cdot N_2 \cdot N_0 + \ldots$

- Reduces number of gates and gate inputs…a little

Simplified Circuit



Compared to the first synthesis of 4-bit prime-number detector, this has three fewer gates and two gates have fewer inputs … but there are better ways …

3-variable Karnaugh Map

Graphical representation of the truth table:

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)**

0	2	6	4
1	3	7	5

Mapping from row indices to table cells

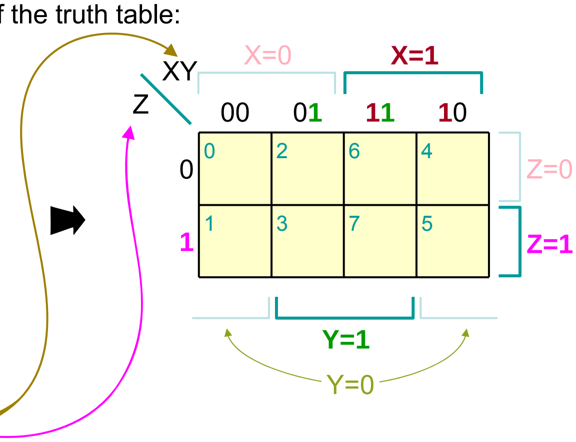
Alternative mapping:

0	1	3	2
4	5	7	6

3-variable Karnaugh Map

Graphical representation of the truth table:

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)**

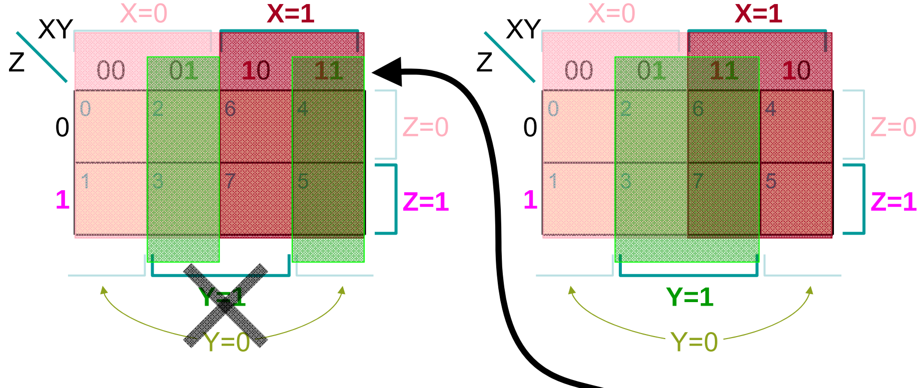


XY is 00 01 11(!) 10 to optimize the mapping…

… but why not use a more “natural” mapping 00 01 10 11 ?

3-variable Karnaugh Map

Graphical representation of the truth table:



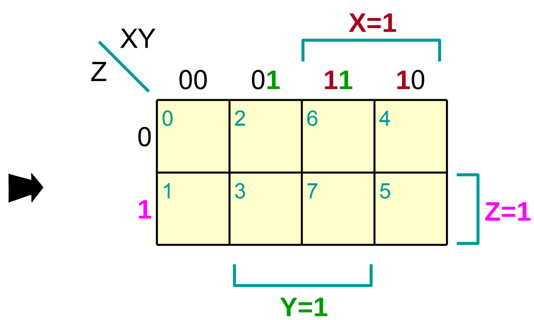
… but why not use a more “natural” mapping 00 01 10 11 ?

➔ Because some variables will not have contiguous values!

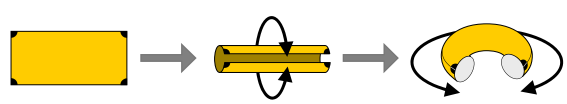
3-variable Karnaugh Map

Graphical representation of the truth table:

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)**

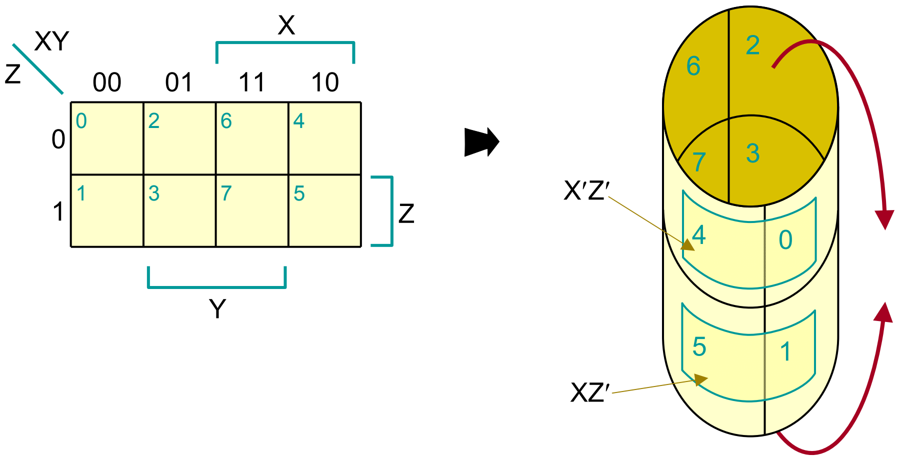


Karnaugh Map wraps around to form a *torus* (doughnut shape):



3-variable Karnaugh Map

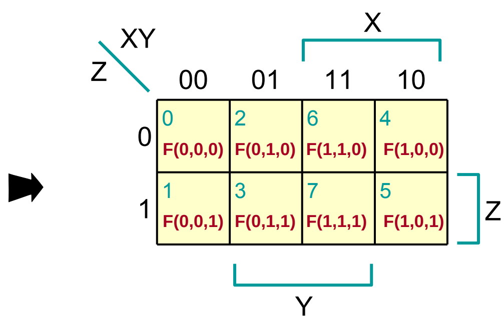
Karnaugh Map wraps around to form a *torus* (doughnut shape):



3-variable Karnaugh Map

Graphical representation of the truth table:

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)



!!!
Almost all of the examples will be sum-of-products.
(AND-OR circuits)

Example: F = $\sum_{X,Y,Z}(1,2,5,7)$

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

Z \ XY	00	01	11	10
0	(0) 0	(2) 1	(6) 0	(4) 0
1	(1) 1	(3) 0	(7) 1	(5) 1

X (columns 11, 10); Y (columns 01, 11); Z (row 1)

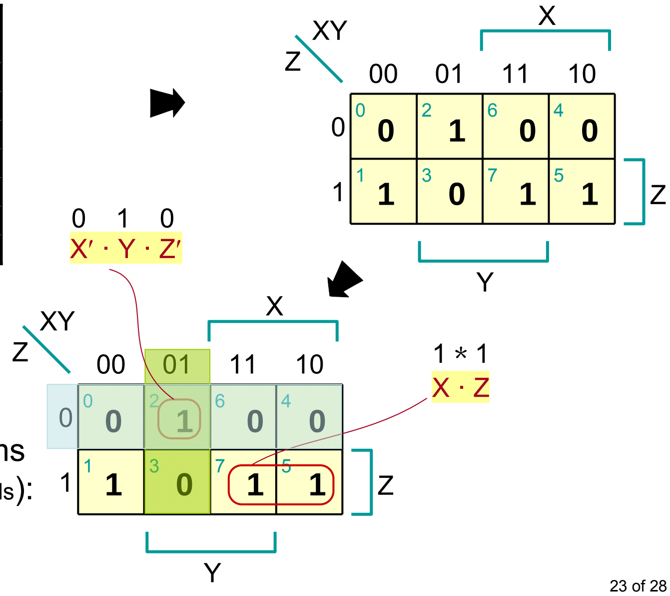
Karnaugh-map Usage:

Minimizing Sums-of-Products

- Show 1's corresponding to minterms of function.
- Circle the largest possible rectangular sets of 1's.
  - # of 1's in set that must be power of 2
  - OK to cross edges across the borders
- Read off product terms, one per circled set
  - An input variable is "1" ⇒ include the variable
  - A variable is "0" ⇒ include the complement of variable
  - A variable is both "0" and "1" ⇒ *variable not included*
- Circled sets and corresponding product terms are called "***prime implicants***"
- Yields *minimum number of gates and gate inputs*

Example: F = $\sum_{X,Y,Z}(1,2,5,7)$

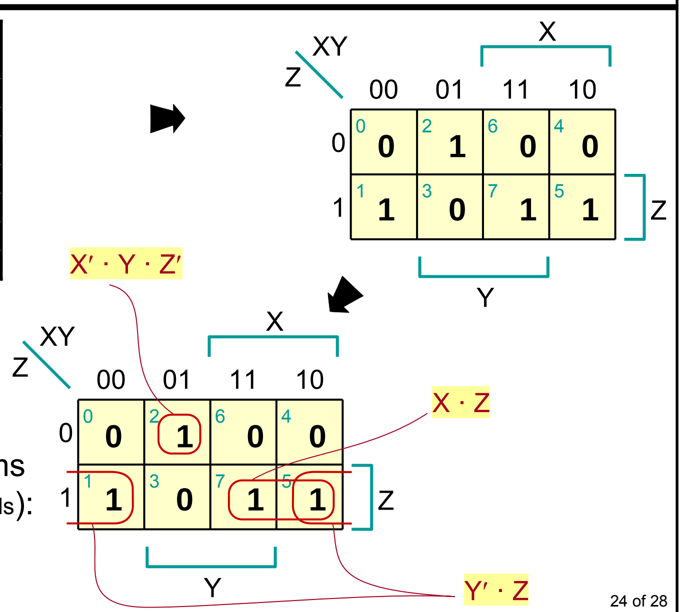
Row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



Circled product terms (combining adjacent "1"-cells):

Example: F = $\sum_{X,Y,Z}(1,2,5,7)$

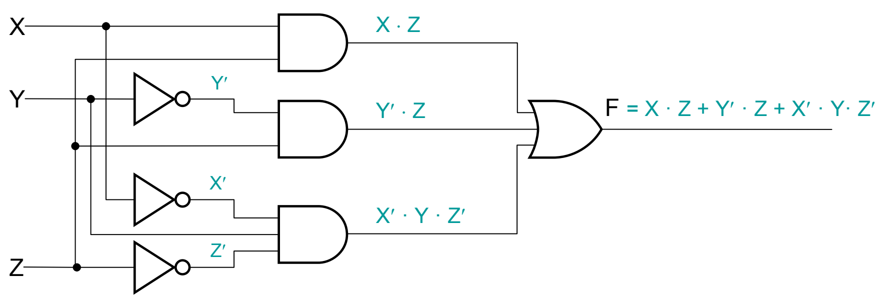
Row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



Circled product terms (combining adjacent "1"-cells):

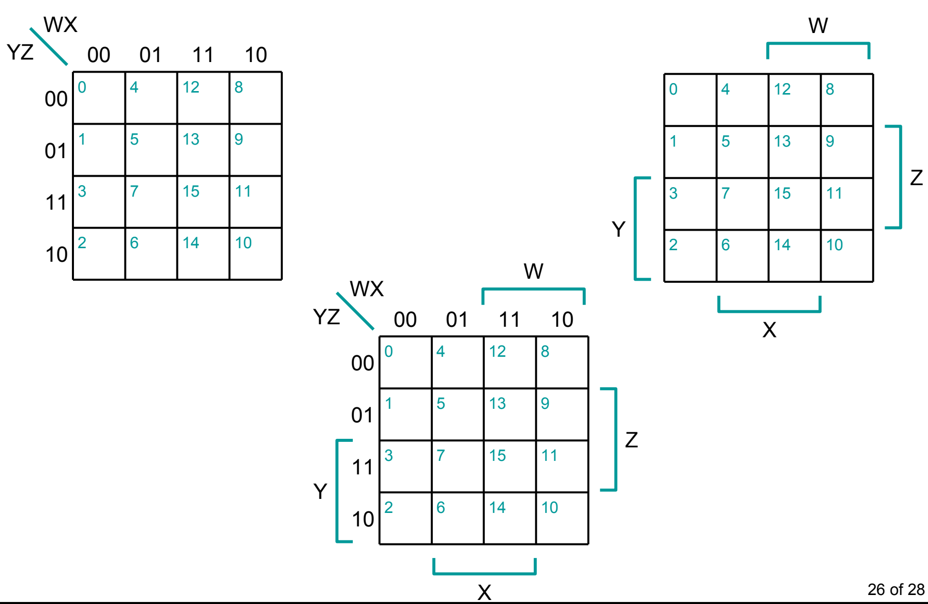
Minimized AND-OR Circuit

…and the circuit is (like before):

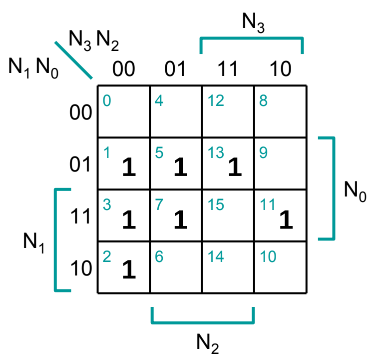


AND-OR circuit, a sum of products

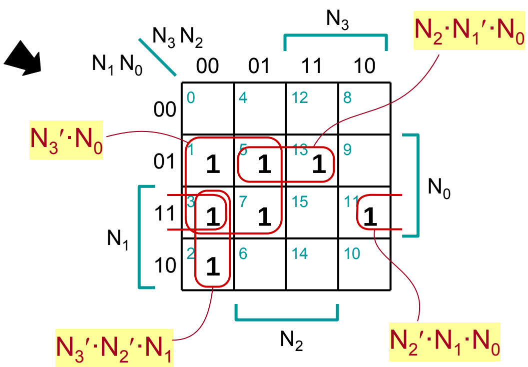
Karnaugh Maps -4 variables-



Prime-number Detector (again)



F = $\sum_{N_3,N_2,N_1,N_0}(1,2,3,5,7,11,13)$



F = $N_3'\cdot N_0 + N_3'\cdot N_2'\cdot N_1 + N_2'\cdot N_1\cdot N_0 + N_2\cdot N_1'\cdot N_0$

Simplified Circuit

- When we solved algebraically, we missed one simplification for the other prime implicants

  the circuit below has three less gate inputs