- Conversion between number systems:
  - Radix-r to decimal.
  - Decimal to binary.
  - Decimal to Radix-r
  - Binary to Octal
  - Binary to Hex
- Binary arithmetic operations.
- Negative number representations.
- Switching Algebra Axioms & Theorems.
- Proof of identities:
  - Using logic expression algebraic manipulation.
  - Using Truth Table (perfect induction).

- Standard Representations of Logic Functions:
  - Truth Table.
  - Canonical Sum Representation:
    - Full sum of minterms expression, or using S notation.
  - Canonical Product Representation:
    - Full product of maxterms expression, or using P notation.
- Combinational Circuit Analysis/ Synthesis.
- Combinational Circuit Minimization using K-maps:
  - Sum of Products (SOP) Minimization using K-maps:
    - Prime implicants, distinguished 1-cells, essential prime implicants
    - Minimization with Don’t care Input Combinations.
  - Product of Sums (POS) Minimization using K-maps:
    - Prime implicates, distinguished 0-cells, essential prime implicates
- Detecting/Eliminating Static Hazards Using K-maps.

Positional Number Systems

- A number system consists of an order set of symbols (digits) with relations defined for +,-,*, /
- The radix (or base) of the number system is the total number of digits allowed in the the number system.
  - Example, for the decimal number system:
    - Radix, r = 10, Digits allowed = 0,1, 2, 3, 4, 5, 6, 7, 8, 9
- In positional number systems, a number is represented by a string of digits, where each digit position has an associated weight.
- The value of a number is the weighted sum of the digits.
- The general representation of an unsigned number D with whole and fraction portions number in a number system with radix r:

$$D_r = d_{p-1}\, d_{p-2} \ldots\ldots d_1\, d_0 . d_{-1}\, d_{-2} \ldots\ldots D_{-n}$$

- The number above has p digits to the left of the radix point and n fraction digits to the right.
- A digit in position i has as associated weight r^i
- The value of the number is the sum of the digits multiplied by the associated weight r^i :

$$D = \sum_{i=-n}^{p-1} d_i \times r^i$$

EECC341 - Shaaban

Positional Number Systems

Number: $D_r \;=\; d_{p-1}\, d_{p-2}\; \ldots\ldots\; d_1\, d_0 . d_{-1}\, d_{-2}\; \ldots\; D_{-n}$

Value: $$D = \sum_{i=-n}^{p-1} d_i \times r^i$$

- **For example in the decimal number system:**

$5185.68_{10} = 5x10^3 + 1x10^2 + 8x10^1 + 5x10^0 + 6 \text{ x } 10^{-1} + 8 \text{ x } 10^{-2}$

$= 5x1000 + 1x100 + 8x10 + 5 \text{ x } 1 + 6x.1 + 8x.01$

- **For the binary number system with radix = 2, digits 0, 1**

$D_2 = d_{p-1}$ ´ 2^{p-1} ….. d_1 ´ $2^1 + d_0 . 2^0 + d_{-1}$ ´ $2^{-1} + d_{-2}$ ´ 2^{-2} …..

- **For Example:**

10011_2 = 1 ´ 16 + 0 ´ 8 + 0 ´ 4 + 1 ´ 2 + 1 ´ 1 = 19_{10}

| |

MSB LSB (least significant bit)

(most significant bit)

$101.001_2 = 1x4 + 0x2 + 1x1 + 0x.5 + 0x.25 + 1x.125 = 5.125_{10}$

Binary Point

Number Systems Used in Computers

Name of Radix	Radix	Set of Digits	Example
Decimal	r=10	{0,1,2,3,4,5,6,7,8,9}	255_{10}
Binary	r=2	{0,1}	11111111_2
Octal	r= 8	{0,1,2,3,4,5,6,7}	377_8
Hexadecimal	r=16	{0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F}	FF_{16}

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Radix-r to Decimal Conversion

- **The decimal value of a number in any radix r is found by converting each digit to its radix 10 equivalent and expanding the value using radix arithmetic:**

$$D = \sum_{i=-n}^{p-1} d_i \times r^i$$

- **Examples:**

$$1101.101_2 = 1´2^3 + 1´2^2 + 1´2^0 + 1´2^{-1} + 1´2^{-3}$$
$$= 8 + 4 + 1 + 0.5 + 0.125 = 13.625_{10}$$

$$572.6_8 = 5´8^2 + 7´8^1 + 2´8^0 + 6´8^{-1}$$
$$= 320 + 56 + 16 + 0.75 = 392.75_{10}$$

$$2A.8_{16} = 2´16^1 + 10´16^0 + 8´16^{-1}$$
$$= 32 + 10 + 0.5 = 42.5_{10}$$

$$132.3_4 = 1´4^2 + 3´4^1 + 2´4^0 + 3´4^{-1}$$
$$= 16 + 12 + 2 + 0.75 = 30.75_{10}$$

$$341.24_5 = 3´5^2 + 4´5^1 + 1´5^0 + 2´5^{-1} + 4´5^{-2}$$
$$= 75 + 20 + 1 + 0.4 + 0.16 = 96.56_{10}$$

Decimal-to-Binary Conversion

- Separate the decimal number into whole and fraction portions.
- To convert the whole number portion to binary, use successive division by 2 until the quotient is 0. The remainders form the answer, with the first remainder as the *least significant bit (LSB)* and the last as the *most significant bit (MSB)*.
- Example: Convert 179_{10} to binary:

```
179 / 2  =   89  remainder 1   (LSB)
           / 2  =  44 remainder 1
                   / 2  =   22 remainder 0
                            / 2  =  11 remainder 0
                                    / 2 =  5 remainder 1
                                          / 2 =  2 remainder 1
                                                / 2 =  1 remainder 0
                                                       / 2 =  0  remainder  1  (MSB)
```

$179_{10} = 10110011_2$

EECC341 - Shaaban

Decimal-to-Binary Conversion

- To convert decimal fractions to binary, repeated multiplication by 2 is used, until the fractional product is 0 (or until the desired number of binary places). The whole digits of the multiplication results produce the answer, with the first as the MSB, and the last as the LSB.
- Example: Convert 0.3125_{10} to binary

		Result Digit	
$.3125 \times 2$	$= 0.625$	0	(MSB)
$.625 \times 2$	$= 1.25$	1	
$.25 \times 2$	$= 0.50$	0	
$.5 \times 2$	$= 1.0$	1	(LSB)

$$0.3125_{10} = .0101_2$$

Decimal to Radix-r Conversion

- Separate the decimal number into whole and fraction portions.
- To convert the whole number portion to binary, use successive division by r until the quotient is 0. The remainders form the answer, with the first remainder as the *least significant digit (LSD)* and the last as the *most significant digit (MSD)*.
- To convert decimal fractions to radix-r, repeated multiplication by r is used, until the fractional product is 0 (or until the desired number of binary places). The whole digits of the multiplication results produce the answer, with the first as the MSD, and the last as the LSD.
- Example: Convert 467_{10} to octal

467 / 8 = 58 remainder 3 (LSD)

/ 8 = 7 remainder 2

/ 8 = 0 remainder 7 (MSD)

$467_{10} = 723_8$

Binary to Octal Conversion

- **Separate the whole binary number portion into groups of 3 beginning at the binary point and working to the left. Add leading zeroes as necessary.**

- **Separate the fraction binary number portion into groups of 3 beginning at the binary point and working to the right. Add trailing zeroes as necessary.**

- **Convert each group of 3 to the equivalent octal digit.**
- **Example:**

$$3564.875_{10} = 110\ 111\ 101\ 100.111_{2}$$

$$= (6\ \text{´}\ 8^{3}) + (7\ \text{´}\ 8^{2}) + (5\ \text{´}\ 8^{1})+(4\ \text{´}\ 8^{0})+(7\ \text{´}\ 8^{-1})$$

$$= 6754.7_{8}$$

Binary to Hexadecimal Conversion

- Separate the whole binary number portion into groups of 4 beginning at the binary point and working to the left. Add leading zeroes as necessary.
- Separate the fraction binary number portion into groups of 4 beginning at the binary point and working to the right. Add trailing zeroes as necessary.
- Convert each group of 4 to the equivalent hexadecimal digit.
- Example:

3564.875_{10} = $1101\ 1110\ 1100.1110_{2}$

= (D ´ 16^{2}) + (E ´ 16^{1}) + (C ´ 16^{0})+(E ´ 16^{-1})

= $\mathrm{DEC.E}_{16}$

Conversion between Number Systems Summary

- Radix-r to decimal:
  - Multiply digits with their corresponding weights and add

$$D = \sum_{i=-n}^{p-1} d_i \times r^i$$

- Decimal to binary (radix 2)
  - Whole numbers: repeated division by 2
  - Fractions: repeated multiplication by 2
- Decimal to radix-r
  - Whole numbers: repeated division by r
  - Fractions: repeated multiplication by r
- Binary to Octal
  - Substitute groups of three bits with corresponding octal digit.
- Binary to Hexadecimal
  - Substitute groups of four bits with corresponding hexadecimal digit.

Binary Arithmetic Operations
Addition

- **Similar to decimal number addition, two binary numbers are added by adding each pair of bits together with carry propagation.**

- **Addition Example:**

			1 0 1 1 1 1 0 0 0	Carry
X	190		1 0 1 1 1 1 1 0	
Y	+ 141	+	1 0 0 0 1 1 0 1	
X + Y	331		1 0 1 0 0 1 0 1 1	

Binary Arithmetic Operations
Subtraction

- Two binary numbers are subtracted by subtracting each pair of bits together with borrowing, where needed.
- Subtraction Example:

				0 0 1 1 1 1 1 0 0	Borrow
X		229		1 1 1 0 0 1 0 1	
Y	-	46	-	0 0 1 0 1 1 1 0	
		183		1 0 1 1 0 1 1 1	

Negative Binary Number Representations

- **Signed-Magnitude Representation:**

  – **For an *n-bit* binary number:**

  Use the first bit (most significant bit, MSB) position to represent the sign where 0 is positive and 1 is negative.

  Ex. $1\,1\,1\,1\,1\,1\,1\,1_2 = -127_{10}$

  Sign **Magnitude**

  – **Remaining n-1 bits represent the magnitude which may range from:**

  $$\boxed{-2^{(n-1)} + 1 \quad \text{to} \quad 2^{(n-1)} - 1}$$

  – **This scheme has two representations for 0; i.e., both positive and negative 0: for 8 bits: 00000000, 10000000**

  – **Arithmetic under this scheme uses the sign bit to indicate the nature of the operation and the sign of the result, but the sign bit is not used as part of the arithmetic.**

Negative Binary Number Representations

- **Two's complement representation:**
- **MSB is the sign (MSB = 1 indicates a negative number)**
- **To negate a number complement all bits and add 1**
- **ex. 119_{10} = 01110111 complement bits**

10001000

+1 add 1

$10001001_2 = -\ 119_{10}$

Properties of Two's Complement Numbers

- X plus the complement of X equals 0.
- There is one unique 0.
- Positive numbers have 0 as their leading bit (MSB); while negatives have 1 as their MSB.
- The range for an *n-bit* binary number in 2’s complement representation is:

$$\boxed{\text{from} \quad -2^{(n-1)} \quad \text{to} \quad 2^{(n-1)} - 1}$$

- The complement of the complement of a number is the original number.
- Subtraction is done by *addition* to the 2’s complement of the number.

Value of Two's Complement Numbers

- For an n-bit 2’s complement number the weights of the bits is the same as for unsigned numbers except of the MSB or sign bit where the weight is -2^{n-1}, thus the value of the n-bit 2’s complement number is given by:

$$D_{\text{2’s-complement}} = d_{n-1} ´ -2^{n-1} + d_{n-2} ´ 2^{n-2} \ldots\ldots \quad d_1 ´ 2^1 + d_0$$

For example:

the value of the 4-bit 2’s complement number 1011 is given by:

$$\begin{aligned} \text{value} &= d_3 ´ -2^{3} + d_2 ´ 2^2 + d_1 ´ 2^1 + d_0 \\ &= 1 ´ -2^{3} + 0 ´ 2^2 + 1 ´ 2^1 + 1 \\ &= -8 + 0 + 2 + 1 \\ &= -8 + 3 = -5 \end{aligned}$$

Extending Two's Complement Numbers: Sign Extension

- An n-bit 2’s complement number can converted to an m-bit number where m>n by appending m-n copies of the sign bit to the left of the number. This process is called sign extension.

- Example: To convert the 4-bit 2’s complement number 1011 to an 8-bit representation, the sign bit (here = 1) must be extended by appending four 1’s to left of the number:

$$1011_{\text{4-bit 2's-complement}} = 11111011_{\text{8-bit 2's-complement}}$$

To verify that the value of the 8-bit number is still -5

$$\text{value of 8-bit number} = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2 + 1$$

$$= -128 + 64 + 32 + 16 + 8 + 2 + 1$$

$$= -128 + 123 = -5$$

Two’ complement addition/subtraction

Examples:

$$\begin{array}{rr}
 & 4 \\
+ & -7 \\
\hline
 & -3
\end{array}
\qquad
\begin{array}{r}
0100 \\
1001 \\
\hline
1101
\end{array}
\qquad\qquad
\begin{array}{rr}
 & -2 \\
+ & -6 \\
\hline
 & -8
\end{array}
\qquad
\begin{array}{rr}
 & 1110 \\
 & 1010 \\
\hline
1 & 1000
\end{array}$$

Ignore carry out from MSB

- **Overflow occurs if signs (MSBs) of both operands are the same and the sign of the result is different.**
- **Overflow can also be detected if the carry in the sign position is different from the carry out of the sign position.**

Negative Binary Number Representations

- **One’s-Complement representation**
- **MSB is the sign (MSB = 1 indicates a negative number)**
- **Negative numbers are found by complementing all bits**
- **ex.** $119_{10} = 01110111$

  $-119_{10} = 10001000$
- **The range of values for an *n-bit* binary number in 1’s complement representation is:**
- **from** $-2^{(n-1)} + 1$ **to** $2^{(n-1)} - 1$
- **One’s-complement addition/subtraction:**

If there is a carry out of the sign position add 1

$$\begin{array}{llr} \textbf{Ex.} & \ -2 & 1101 \\ & \underline{+\ -5} & \underline{1010} \\ & \ -7 & 10111 \\ & & +\quad 1 \\ & & 1000 \end{array}$$

Value of One's Complement Numbers

- For an n-bit 2’s complement number the weights of the bits is also the same as for unsigned numbers except of the MSB or sign bit where the weight is -(2^{n-1} +1), thus the value of the n-bit 1’s complement number is given by:

$D_{\text{1's-complement}}$ = d_{n-1} ´ -(2 $^{n-1}$ +1) + d_{n-2} ´ 2^{n-2} ….. d_1 ´ 2^1 + d_0

For example:

the value of the 4-bit 1’s complement number 1011 is given by:

value = d_3 ´ -(2 ^{3} +1) + d_2 ´ 2^2 + d_1 ´ 2^1 + d_0

= 1 ´ -(2 ^{3} +1) + 0 ´ 2^2 + 1 ´ 2^1 + 1

= -7 + 0 + 2 + 1

= - 7 +3 = -4

Binary Multiplication

- **Multiplication is achieved by adding a list of shifted multiplicands according to the digits of the multiplier.**
- **Ex. (unsigned)**

```
   11                    1 0 1 1      multiplicand (4 bits)
X 13              X      1 1 0 1      multiplier  (4 bits)
--------          -----------------
    33                  1 0 1  1
  11                  0 0 0 0
_______             1 0 1 1
  143             1 0 1 1
                 -------------------
                 1 0 0 0 1 1 1 1             Product (8 bits)
```

Binary Division

- Shift and subtract

Example:

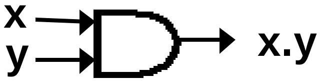
```
      19
11 | 217
     11
     ----
     107
      99
     ----
       8
```

```
              10011   quotient
1011 | 11011001       dividend
       1011           shifted divisor
       --------
        0101          reduced dividend
        0000          shifted divisor
       --------
         1010         reduced dividend
         0000         shifted divisor
         -------
         10100        reduced dividend
          1011        shifted divisor
         -------
          10011       reduced dividend
           1011       shifted divisor
          ------
           1000       remainder
```

Boolean or Switching Algebra

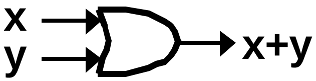
- Set of Elements: {0,1}
- Set of Operations: { . , + , ’ }   AND (logical multiplication, .), OR (logical addition, +) , NOT

x	y	x . y
0	0	0
0	1	0
1	0	0
1	1	1



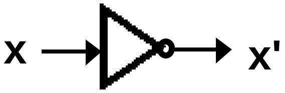
AND

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



OR

x	x’
0	1
1	0



NOT

- Symbolic variables such as X used to represent the condition of a logic signal (0 or 1, low or high, on or off).
- Switching Algebra Axioms (or postulates):
  - Minimal set basic definitions (A1-A5, A1’-A5’) that are assumed to be true and completely define switching algebra.
  - All other switching algebra theorems (T1-T15) can be proven using these axioms as a starting point.

Switching Algebra Axioms & Theorems

(A1)	$X = 0$ if X ¹ 1	(A1')	$X = 1$ if X ¹ 0	
(A2)	If $X = 0$, then $X' = 1$	(A2')	if $X = 1$, then, $X' = 0$	
(A3)	$0 . 0 = 0$	(A3')	$1 + 1 = 1$	
(A4)	$1 . 1 = 1$	(A4')	$0 + 0 = 0$	
(A5)	$0 . 1 = 1 . 0 = 0$	(A5')	$1 + 0 = 0 + 1 = 1$	
(T1)	$X + 0 = X$	(T1')	$X . 1 = X$	(Identities)
(T2)	$X + 1 = 1$	(T2')	$X . 0 = 0$	(Null elements)
(T3)	$X + X = X$	(T3')	$X . X = X$	(Idempotency)
(T4)	$(X')' = X$ (Involution)			
(T5)	$X + X' = 1$	(T5')	$X . X' = 0$	(Complements)
(T6)	$X + Y = Y + X$	(T6')	$X . Y = Y . X$	(Commutativity)
(T7)	$(X + Y) + Z = X + (Y + Z)$	(T7')	$(X . Y) . Z = X . (Y . Z)$	(Associativity)
(T8)	$X . Y + X . Z = X . (Y + Z)$	(T8')	$(X + Y) . (X + Z) = X + Y . Z$	(Distributivity)
(T9)	$X + X . Y = X$	(T9')	$X . (X + Y) = X$	(Covering)
(T10)	$X . Y + X . Y' = X$	(T10')	$(X + Y) . (X + Y') = X$	(Combining)
(T11)	$X . Y + X'. Z + Y . Z = X . Y + X' . Z$			
(T11')	$(X + Y) . (X' + Z) . (Y + Z) = (X + Y) . (X' + Z)$			(Consensus)
(T12)	$X + X + \ldots + X = X$	(T12')	$X . X . \ldots . X = X$	(Generalized idempotency)
(T13)	$(X_1 . X_2 . \ldots . X_n)' = X_1' + X_2' + \ldots + X_n'$			
(T13')	$(X_1 + X_2 + \ldots + X_n)' = X_1' . X_2' . \ldots . X_n'$			(DeMorgan's theorems)
(T14)	$[F(X_1, X_2, \ldots, X_n, +, .)]' = F(X_1', X_2', \ldots, X_n', . , +)$			(Generalized DeMorgran's theorem)

Perfect Induction

- **Most theorems in switching algebra are simple to prove using *perfect induction*:**

Since a switching variable can only take the values 0 and 1 we can prove a theorem involving a single variable X by proving it true for $X = 0$ and $X = 1$

Example: To prove (T1) $X + 0 = X$

$[X = 0]$ $0 + 0 = 0$ true according to axiom A4’

$[X = 1]$ $1 + 0 = 1$ true according to axiom A5’

Theorem Proof using Truth Table

- Can use truth table to prove T8 by perfect induction.
- i.e Prove that: X . Y + X . Z = X . (Y + Z)

(i) Construct truth table for both sides of above equality.

x	y	z	y + z	x.(y + z)	x.y	x.z	x.y + x.z
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

(ii) Check that from truth table check that that X . Y + X . Z = X . (Y + Z)

This is satisfied because output column values for X . Y + X . Z and output column values for X . (Y + Z) are equal for all cases.

Logic Expression Algebraic Manipulation Example

- Prove that the following identity is true using Algebraic expression Manipulation : (one can also prove it using a truth table)

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

– Starting from the left hand side of the identity:

$$\text{Let} \quad F = X.Y + X.Z$$

$$A = X.Y \qquad B = X.Z$$

$$\text{Then} \quad F = A + B$$

– Using DeMorgan's theorem T 13 on F:

$$F = A + B = (A'.B')' \qquad (1)$$

– Using DeMorgan's theorem T 13' on A, B:

$$A = X.Y = (X' + Y')' \qquad (2)$$

$$B = X.Z = (X' + Z')' \qquad (3)$$

– Substituting A, B from (2), (3), back in F in (1) gives:

$$F = (A'.B')' = ((X' + Y').(X' + Z'))'$$

Which is equal to the right hand side of the identity.

Standard Representations of Logic Functions

- Truth table for n-variable logic function:

  Input combinations are arranged in 2^n rows in ascending binary order, and the output values are written in a column next to the rows.

- Practical for functions with a small number of variables.
- The general structure of a 3-variable truth table is given by:

Row	X	Y	Z	F(X,Y,Z)
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)

Truth table for a specific function:

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

Logic Function Representation Definitions

- *A literal:* is a variable or a complement of a variable

  Examples: X, Y, X’, Y’

- *A product term:* is a single literal, or a product of two or more literals.

  Examples: Z’ W.Y.Y X.Y’.Z W’.Y’.Z

- *A sum-of-products expression:* is a logical sum of product terms.

  Example: Z’ + W.X.Y + X.Y’.Z + W’.Y’.Z

- *A sum term:* is a single literal or logical sum of two or more literals

  Examples: Z’ W + X + Y X + Y’ + Z W’ + Y’ + Z

- *A product-of-sums expression:* is a logical product of sum terms.

  Example: Z’. (W + X + Y) . (X + Y’ + Z) . (W’ + Y’ + Z)

- *A normal term:* is a product or sum term in which no variable appears more than once

  Examples of non-normal terms: W.X.X.Y’ W+W+X’+Y X.X’.Y

  Examples of normal terms: W . X . Y’ W + X’ + Y

Logic Function Representation Definitions

- **Minterm**

  An n-variable minterm is a normal product term with n literals.

  There are 2^n such products terms.

  Example of 4-variable minterms:

  W.X'.Y'.Z'   W.X.Y'.Z   W'.X'.Y.Z'

- **Maxterm**

  An n-variable maxterm is a normal sum term with n literals.

  There are 2^n such sum terms.

  Examples of 4-variable maxterms:

  W' + X' + Y + Z'   W + X' + Y' + Z   W' + X' + Y + Z

- **A minterm can be defined as as product term that is 1 in exactly one row of the truth table.**

- **A maxterm can similarly be defined as a sum term that is 0 in exactly one row in the truth table.**

Minterms/Maxterms for A 3-variable function F(X,Y,Z)

Row	X	Y	Z	F	Minterm	Maxterm
0	0	0	0	F(0,0,0)	X’.Y’.Z’	X + Y + Z
1	0	0	1	F(0,0,1)	X’.Y’.Z	X + Y + Z’
2	0	1	0	F(0,1,0)	X’.Y.Z’	X + Y’ + Z
3	0	1	1	F(0,1,1)	X’.Y.Z	X + Y’ + Z’
4	1	0	0	F(1,0,0)	X.Y’.Z’	X’ + Y + Z
5	1	0	1	F(1,0,1)	X.Y’.Z	X’ + Y + Z’
6	1	1	0	F(1,1,0)	X.Y.Z’	X’ + Y’ + Z
7	1	1	1	F(1,1,1)	X.Y.Z	X’ + Y’ + Z’

Minterms/Maxterms for
A 4-variable function F(W,X,Y,Z)

Row	W	X	Y	Z	F	Minterm	Maxterm
0	0	0	0	0	F(0,0,0,0)	W’.X’.Y’.Z’	W + X + Y + Z
1	0	0	0	1	F(0,0,0,1)	W’. X’.Y’.Z	W + X + Y + Z’
2	0	0	1	0	F(0,0,1,0)	W’. X’.Y.Z’	W + X + Y’ + Z
3	0	0	1	1	F(0,0,1,1)	W’. X’.Y.Z	W + X + Y’ + Z’
4	0	1	0	0	F(0,1,0,0)	W’.X.Y’.Z’	W + X’ + Y + Z
5	0	1	0	1	F(0,1,0,1)	W’.X.Y’.Z	W + X’ + Y + Z’
6	0	1	1	0	F(0,1,1,0)	W’.X.Y.Z’	W + X’ + Y’ + Z
7	0	1	1	1	F(0,1,1,1)	W’.X.Y.Z	W+ X’ + Y’ + Z’
8	1	0	0	0	F(1,0,0,0)	W.X’.Y’.Z’	W’ + X + Y + Z
9	1	0	0	1	F(1,0,0,1)	W.X’.Y’.Z	W’ + X + Y + Z’
10	1	0	1	0	F(1,0,1,0)	W.X’.Y.Z’	W’ + X + Y’ + Z
11	1	0	1	1	F(1,0,1,1)	W.X’.Y.Z	W’ + X + Y’ + Z’
12	1	1	0	0	F(1,1,0,0)	W.X.Y’.Z’	W’ + X’ + Y + Z
13	1	1	0	1	F(1,1,0,1)	W.X.Y’.Z	W’ + X’ + Y + Z’
14	1	1	1	0	F(1,1,1,0)	W.X.Y.Z’	W’ + X’ + Y’ + Z
15	1	1	1	1	F(1,1,1,1)	W.X.Y.Z	W’+ X’ + Y’ + Z’

Canonical Sum Representation

- Minterm number:

  minterm i refers to the minterm corresponding to row i of the truth table. For n-variables i is in the set

  {0,1, …, 2^n-1}

- The canonical sum representation of a logic function is a sum of the minterms corresponding to the truth table rows for which the function produces a 1 output.
- A short-hand representation of the minterm list uses the S notation and minterm numbers to indicate the sum of minterms of the function.
- This representation is usually realized using 2-level AND-OR logic circuits with inverters at AND gates inputs as needed.

Canonical Sum Example

- **The function represented by the truth table:**

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

has the canonical sum representation:

$$F = S_{X,Y,Z}\,(0, 3, 4, 6, 7)$$

Minterm list using Σ notation

$$= X'.Y'.Z' + X'.Y.Z + X.Y'.Z' + X.Y'.Z' + X.Y.Z$$

Algebraic canonical sum of minterms

Canonical Product Representation

- Maxterm i refers to the maxterm corresponding to row i of the truth table. For n-variables i is in the set

  {0,1, …, 2^n-1}

- The canonical product representation of a logic function is the product of the maxterms corresponding to the truth table rows for which the function produces a 0 output.

- The product of such minterms is called a maxterm list
- A short-hand representation of the maxterm list uses the P notation and maxterm numbers to indicate the product of maxterms of the function.
- This representation is usually realized using 2-level OR-AND logic circuits with inverters at OR gates inputs as needed.

Canonical Product Example

- **The function represented by the truth table:**

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

has the canonical product representation:

$$F \quad = \mathbf{P}_{X,Y,Z}\,(1,2,5) \longleftarrow \text{Maxterm list using } \Pi \text{ notation}$$

$$= (X + Y + Z') \,.\, (X + Y' + Z) \,.\, (X' + Y + Z')$$

Algebraic canonical product of maxterms

Conversion Between Minterm/Maxterm Lists

- To convert between a minterm list and a maxterm list take the set complement.

  Examples:

$$\mathbf{S}_{X,Y,Z}(0,1,2,3) = \mathbf{P}_{X,Y,Z}(4,5,6,7)$$

$$\mathbf{S}_{X,Y}(1) = \mathbf{P}_{X,Y}(0,2,3)$$

$$\mathbf{S}_{W,X,Y,Z}(0,1,2,3,5,7,11,13) = \mathbf{P}_{W,X,Y,Z}(4,6,8,9,12,14,15)$$

- **Combinational Circuit Analysis:**
  - Start with a logic diagram of the circuit.
  - Proceed to a formal description of the function of the circuit using truth tables or logic expressions.
- **Combinational Circuit Synthesis:**
  - May start with an informal (possibly verbal) description of the function performed.
  - A formal description of the circuit function in terms of a truth table or logic expression.
  - The logic expression is manipulated using Boolean (or switching) algebra and optimized to minimize the number of gates needed, or to use specific type of gates.
  - A logic diagram is generated based on the resulting logic expression.

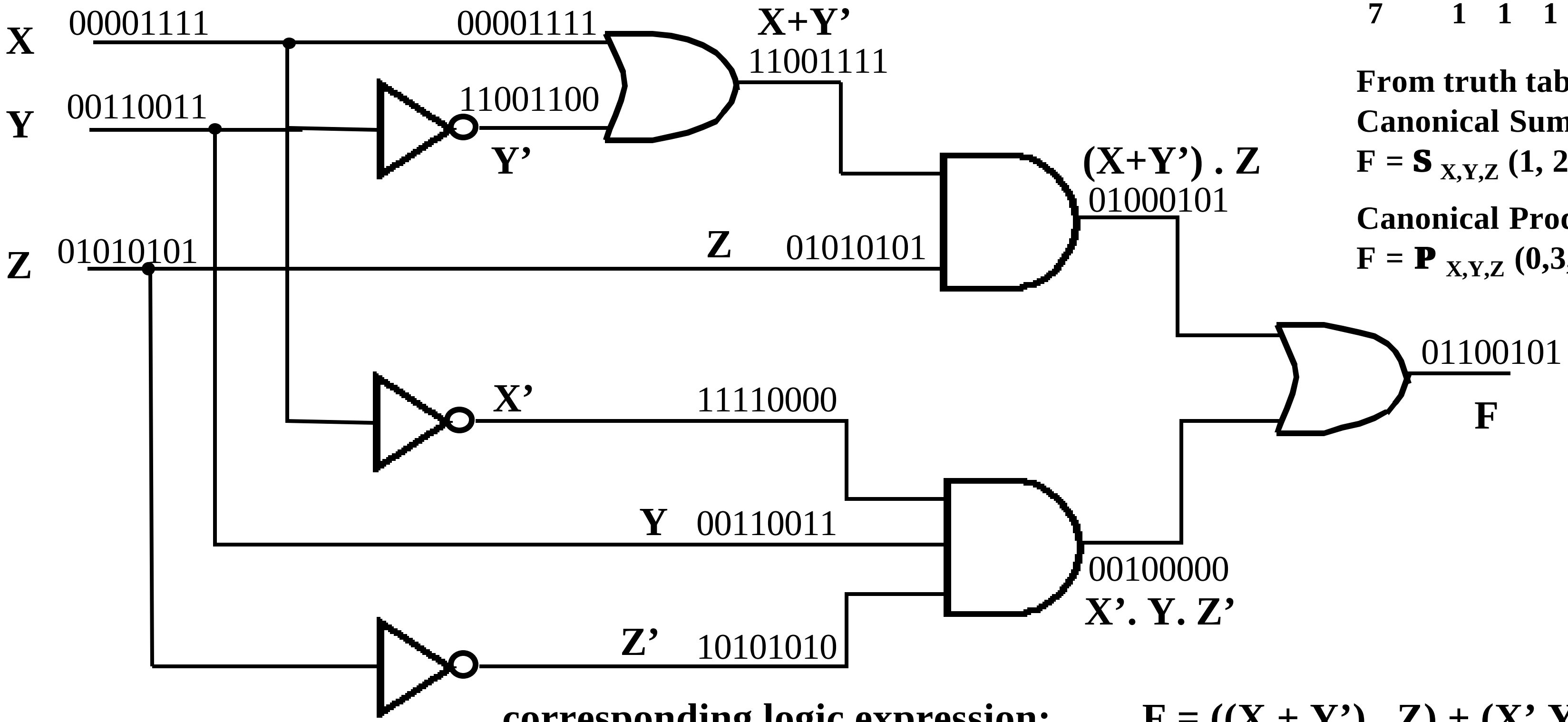
Combinational Circuit Analysis Example

Given this logic circuit we can :

- **Find corresponding logic expression from circuit**
- **Create truth table by applying all input combinations:**
  - **From truth table find Canonical Sum/Product Representations**
- **Manipulate logic expression to other forms using theorems.**

Truth Table

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



From truth table:
Canonical Sum
$F = S_{X,Y,Z}$ (1, 2, 5,7)

Canonical Product
$F = P_{X,Y,Z}$ (0,3,4,6)

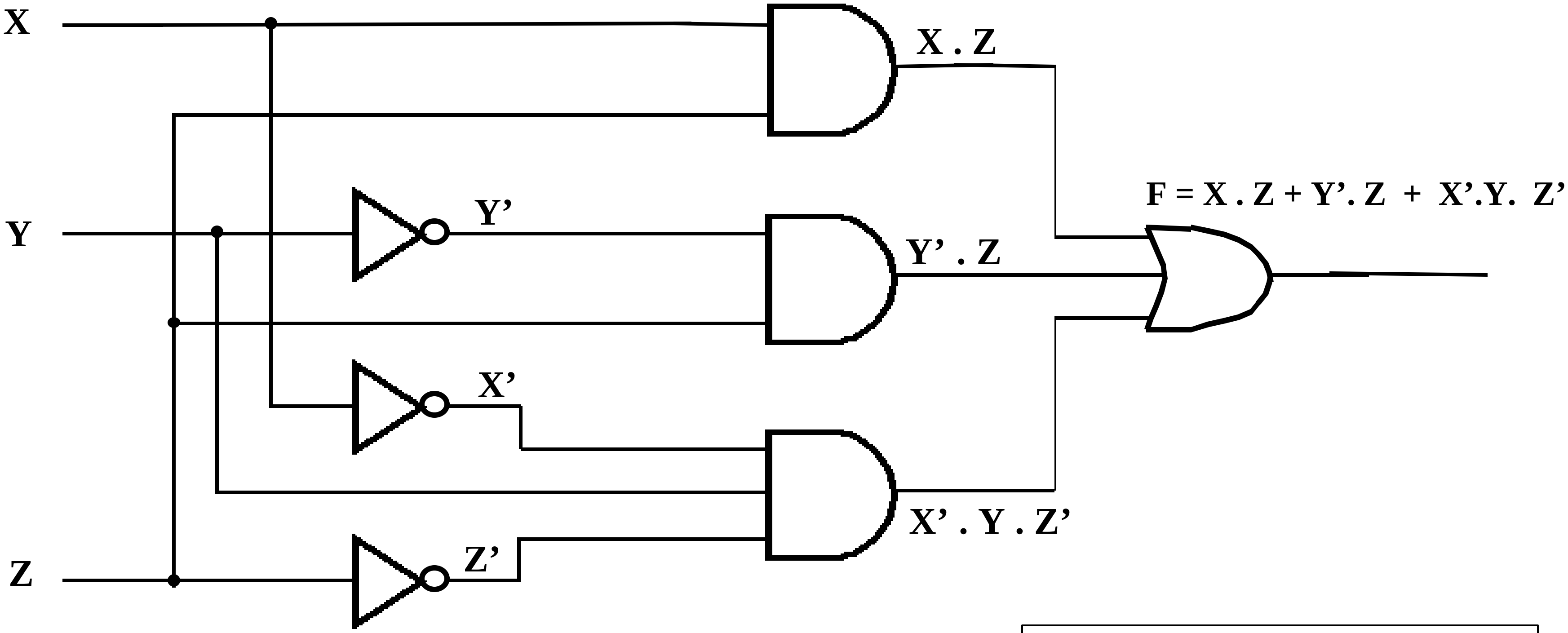
corresponding logic expression: **F = ((X + Y') . Z) + (X'.Y. Z')**

Combinational Circuit Analysis Example (continued)

- The previous circuit logic expression F can be transformed into sum of products by multiplying out (Using T8') and written as :

$$F = X . Z + Y'. Z + X'.Y. Z'$$

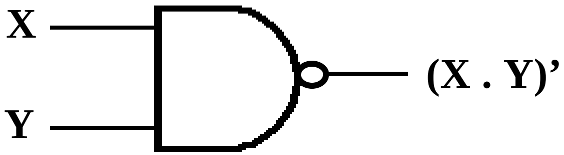
Realized using a 2-level AND-OR circuit:



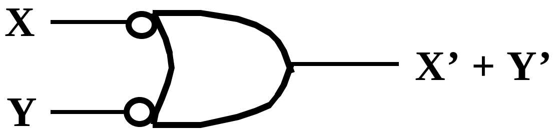
Equivalent Symbols of NAND, NOR Gates

NAND Symbols

Normal Symbol



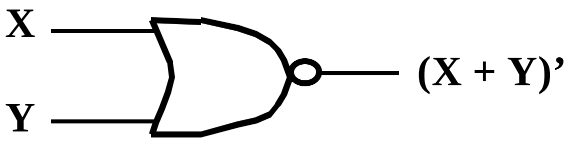
Alternate NAND Symbol



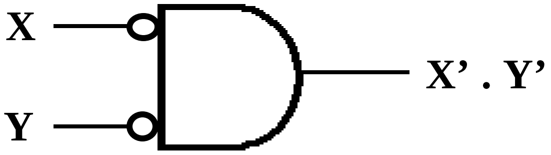
According to DeMorgan’s theorem T13:   (X . Y)’ =   X’ +   Y’

NOR Symbols

Normal NOR Symbol



Alternate NOR Symbol



According to DeMorgan’s theorem T13’:   (X + Y)’ =   X’ . Y’

NAND-NAND Logic Circuits for Sum of Products

- A sum of products logic expression can be realized by NAND gates by replacing all AND gates and the OR GATE in the usual realization with NAND gates as follows:

F = A + B + C + D ...

where A, B, C, …. are product terms of the input variables e.g. A= x.y.z

F = (A')'+(B')'+(C')'+(D')' + …. from T4

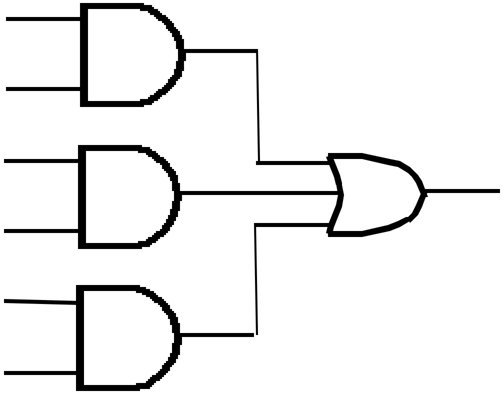
= (A'.B'.C'.D'…)' (from DeMorgan's theorem T13)

This is a 2-level NAND representation.

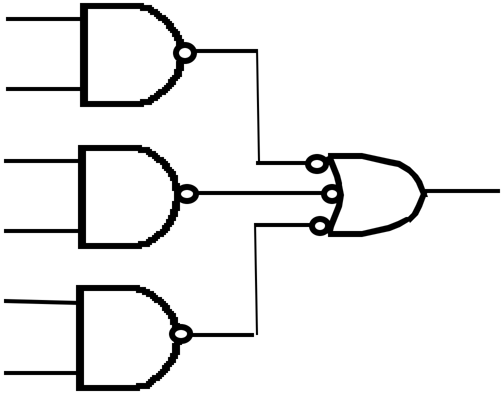
Alternate Sum of Products Realizations

(Applying DeMorgan’s theorem T13 Graphically)

AND-OR



NAND-NAND



EECC341 - Shaaban

NAND-NAND Sum of Products Example

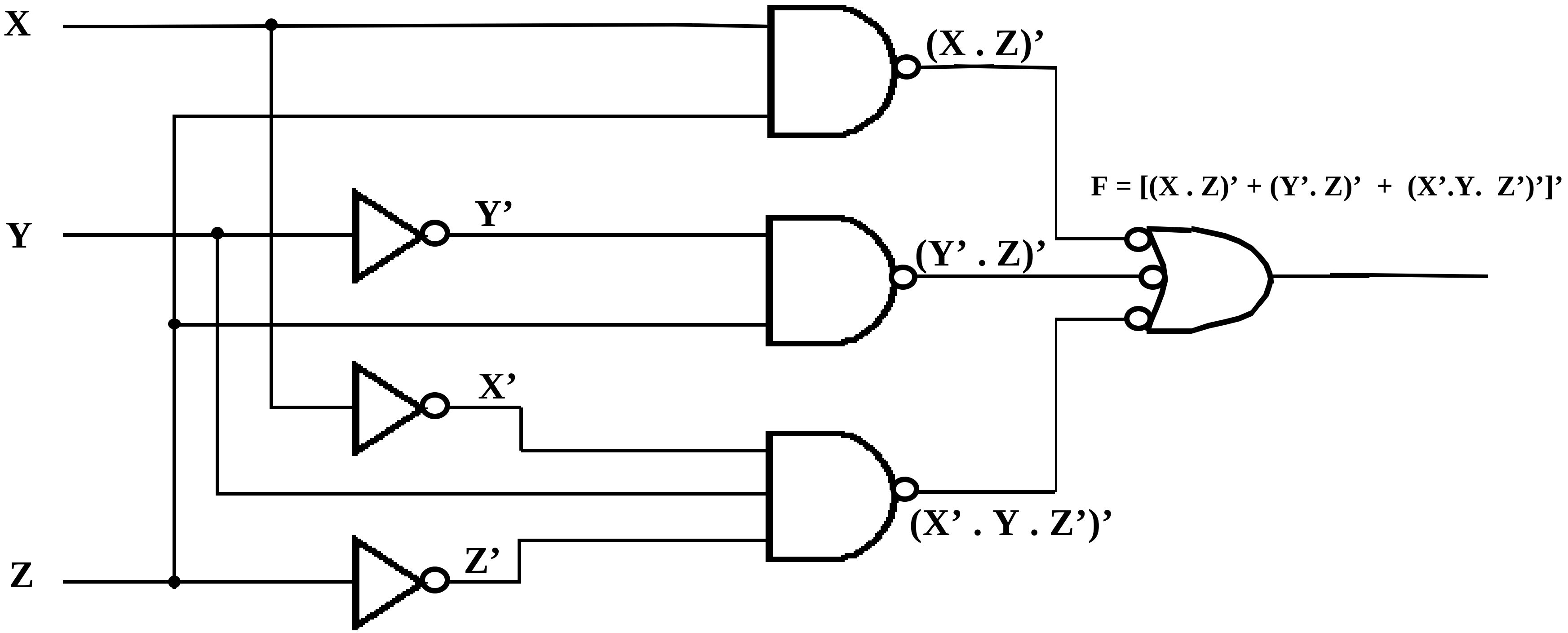
- **The sum of products expression**

F = X . Z + Y'. Z + X'.Y. Z'

F = ((X . Z)')' + ((Y'. Z)')' + ((X'.Y. Z')')' double negate T4

F = [(X . Z)' . (Y'. Z)' . (X'.Y. Z')']' DeMorgan's theorem T13

Can be realized using the 2-level NAND-NAND circuit:



NOR-NOR Circuits for Product of Sums

- A product of sums expression can be realized by NOR gates by replacing all the OR gates and the AND gate with NOR gates as follows:

F = A.B.C.D. ….

Where A, B, C are sum terms of the input variables (e.g. A = x+y+z)

F = (A')'.(B')'.(C')'.(D')' …. using T4

= (A' + B' + C' + D' + …)'

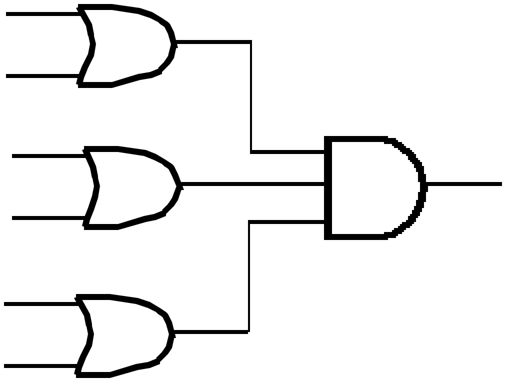
(using Demorgan's theorem T13')

This is a 2-level NOR-NOR representation

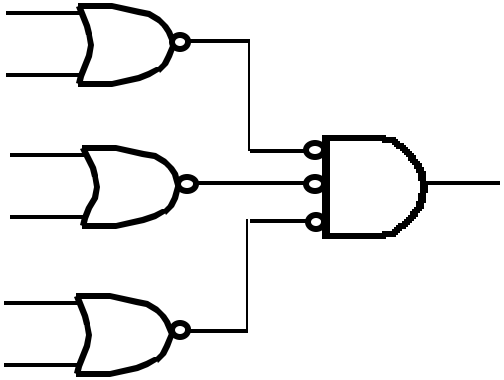
Alternate Product of Sums Realizations

(Applying DeMorgan’s theorem T13’ Graphically)

OR-AND



NOR-NOR



Combinational Circuit Synthesis

- An example of a combinational circuit description:

  Create a logic function in 4 input variables $N=N_3N_2N_1N_0$ whose output is 1 only if the input is a prime number.

- This function is 1 when the input N =1,2,3,5,7,11 can be written in the canonical sum of products representation as:

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

$$= N_3'N_2'N_1'N_0 + N_3'N_2'N_1N_0' + N_3'N_2'N_1N_0$$

$$+ N_3'N_2N_1'N + N_3'N_2N_1N_0 + N_3N_2'N_1N_0 + N_3N_2N_1'N_0$$

A Verbal Synthesis Example: An Alarm Circuit

- **A verbal logic description:**
  - **The ALARM output is 1 if the panic input is 1, or if the ENABLE input is 1, the EXISTING input is 0, and the house is not secure.**
  - **The house is secure if the WINDOW, DOOR, GARAGE inputs are all 1**

- **This can be put in logic expressions as follows:**

ALARM = PANIC + ENABLE . EXISTING' . SECURE'

SECURE = WINDOW. DOOR. GARAGE

ALARM = PANIC + ENABLE . EXISTING'. (WINDOW . DOOR . GARAGE)'

In sum of products form as (by using DeMorgan T13 and multiplying out) :

ALARM = PANIC + ENABLE. EXISTING' . WINDOW'
+ ENABLE . EXISTING'. DOOR'+ ENABLE. EXISTING'. GARAGE'

Combinational Circuit Minimization

- Canonical sum and product logic expressions do not provide a circuit realization with the minimum number of gates.
- Minimization methods reduce the cost of two level AND-OR, NAND-NAND, OR-AND, NOR-NOR circuits in three ways:

1 By minimizing the number of first level gates
2 By minimizing the number of inputs of each first-level gate.
3 Minimizing the inputs of the second level gate

- Most minimization methods are based on the combining theorems T10, T10’:

given product term.Y + given product term.Y’ = given product term

(given sum term+Y).(given sum term + Y’) = given sum term

Karnaugh Maps

- A Karnaugh Map or (K-map for short) is a graphical representation of the truth table of a logic function.
- The K-map for an n-input logic function is an array with 2^n cells or squares, one for each input combination or minterm.
- The rows and columns are labeled so that the input combination for any cell is determined from the row and column headings.
- The row and columns of the map are ordered in such a way that each cell differs from an adjacent cell in only one input variable:
  - Thus for an n-variable K-map, each cell has n adjacent cells.
- The K-map for a function is filled by putting:
  - a '1' in the square corresponding to a minterm
  - a '0' otherwise (maybe omitted)

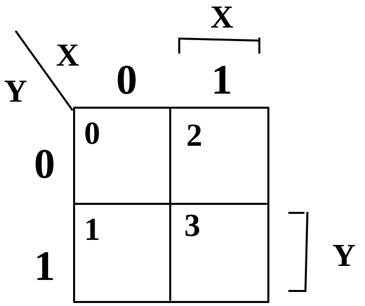
2-Variable K-map

For a 2-variable logic function F(X,Y):

Truth Table:

Row	X	Y	F	Minterm
0	0	0	F(0,0)	X’.Y’
1	0	1	F(0,1)	X’.Y
2	1	0	F(1,0)	X.Y’
3	1	1	F(1,1)	X .Y

K-map

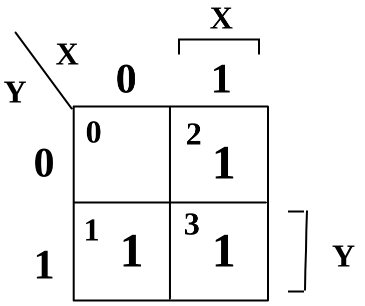


Example: For the function F(X,Y) = $S_{X,Y}$ (1,2,3)

Truth Table:

Row	X	Y	F
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

K-map



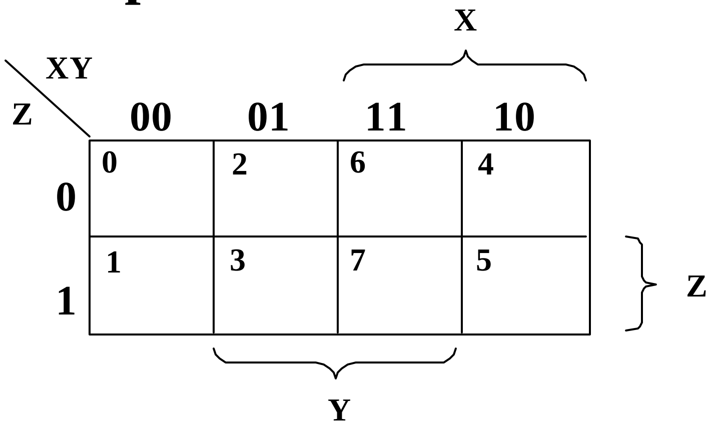
3-Variable K-map

For a 3-variable logic function F(X,Y,Z):

Truth Table:

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

K-map



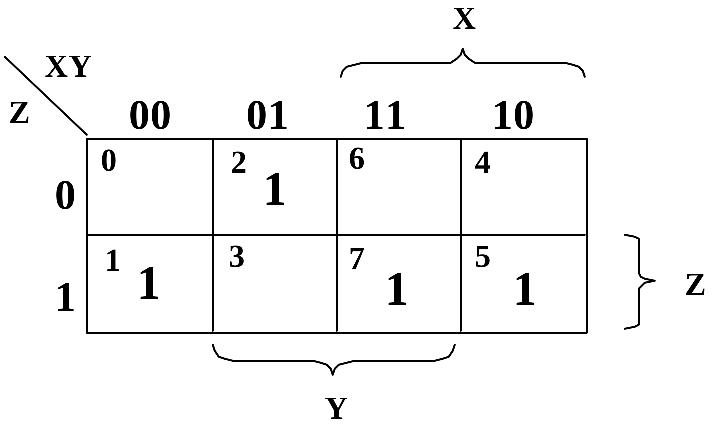
Z \ XY	00	01	11	10
0	0	2	6	4
1	1	3	7	5

Example: For the function F(X,Y,Z) = s $_{X,Y,Z}$ (1,2,5,7)

Truth Table:

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

K-map



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

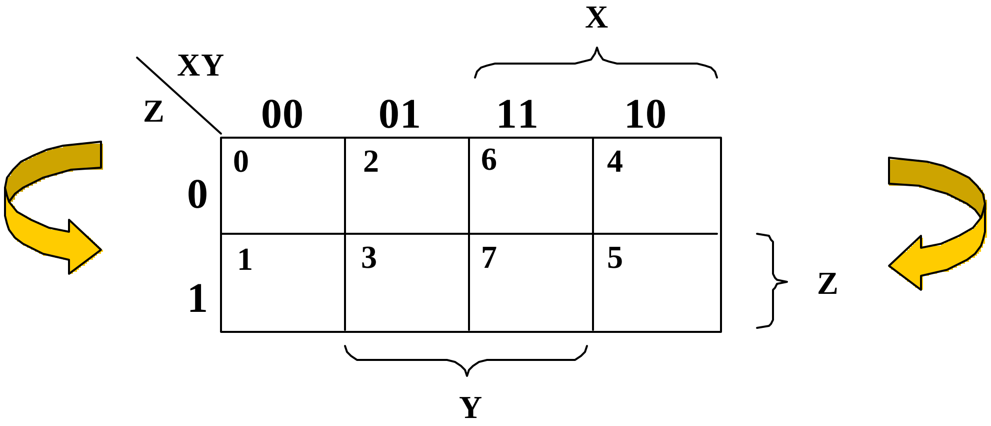
3-Variable K-map (continued)

- There is a horizontal adjacency wrap-around in the 3-variable K-map:

  For example:

  – Cell 0 (minterm 0 = X’.Y’.Z’) is adjacent to:
    - cell 4 (minterm 4, = X.Y’.Z’) by wrap-around.
    - in addition to being adjacent to cells 1, 2 (minterm 1 = X’.Y’.Z minterm 2, = X’.Y.Z’)

  – Cell 1 (minterm 1, X’.Y’.Z) is adjacent to:
    - cell 5 (minterm 5, X.Y’.Z) by wrap-around.
    - in addition to being adjacent to cells 0 , 2 (minterm 0 = X’.Y’.Z’ minterm 3 = X’.Y.Z)



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

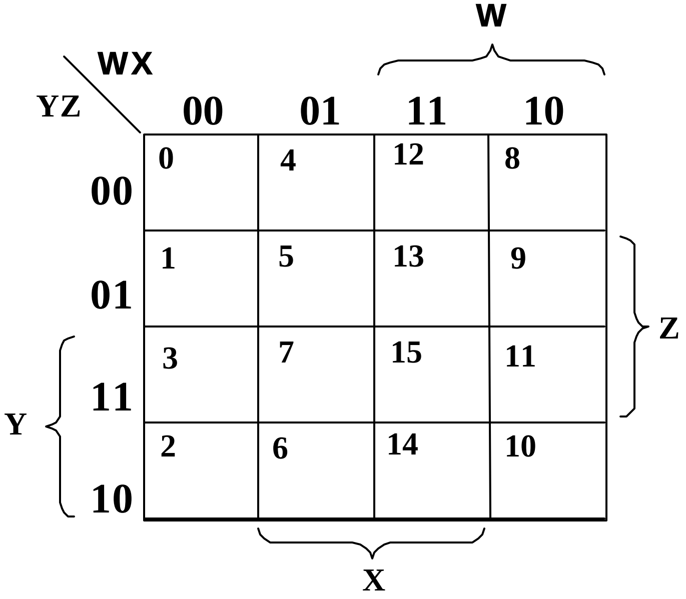
4-Variable K-map

For a 4-variable logic function F(W,X,Y,Z):

Truth Table:

Row	W	X	Y	Z	F	Minterm
0	0	0	0	0	F(0,0,0,0)	W’.X’.Y’.Z’
1	0	0	0	1	F(0,0,0,1)	W’. X’.Y’.Z
2	0	0	1	0	F(0,0,1,0)	W’. X’.Y.Z’
3	0	0	1	1	F(0,0,1,1)	W’. X’.Y.Z
4	0	1	0	0	F(0,1,0,0)	W’. X.Y’.Z’
5	0	1	0	1	F(0,1,0,1)	W’.X.Y’.Z
6	0	1	1	0	F(0,1,1,0)	W’.X.Y.Z’
7	0	1	1	1	F(0,1,1,1)	W’.X.Y.Z
8	1	0	0	0	F(1,0,0,0)	W.X’.Y’.Z’
9	1	0	0	1	F(1,0,0,1)	W.X’.Y’.Z
10	1	0	1	0	F(1,0,1,0)	W.X’.Y.Z’
11	1	0	1	1	F(1,0,1,1)	W.X’.Y.Z
12	1	1	0	0	F(1,1,0,0)	W.X.Y’.Z’
13	1	1	0	1	F(1,1,0,1)	W.X.Y’.Z
14	1	1	1	0	F(1,1,1,0)	W.X.Y.Z’
15	1	1	1	1	F(1,1,1,1)	W.X.Y.Z

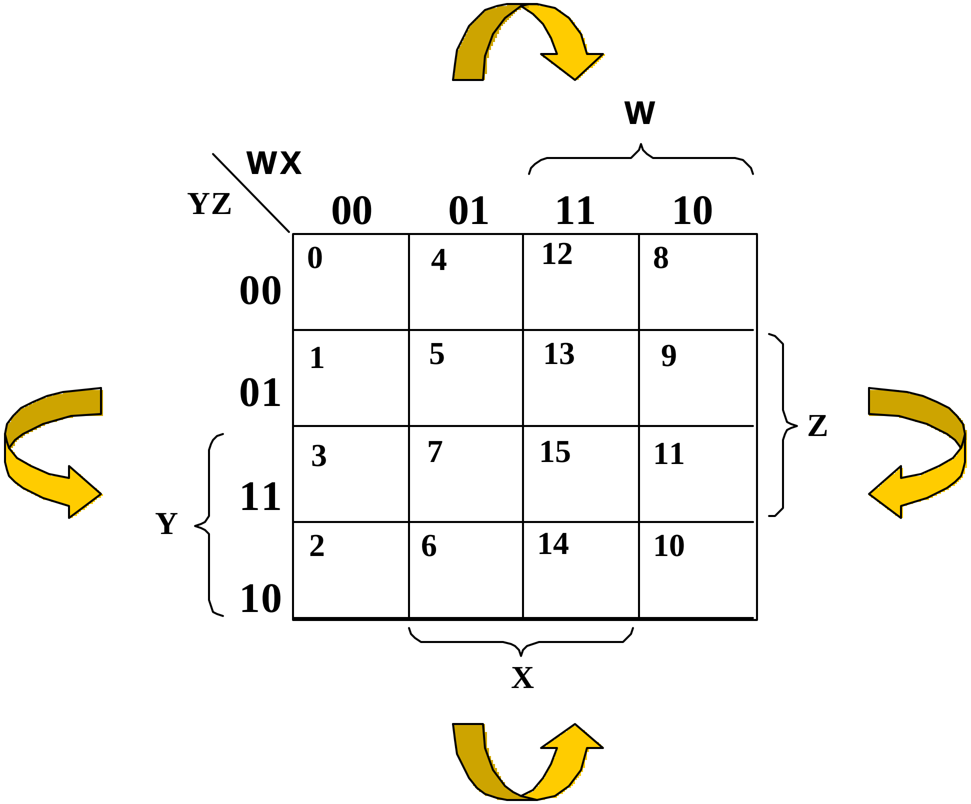
K-map



YZ \ WX	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

4-Variable K-map (continued)

- **There are 2 adjacency wrap-arounds in the 4-variable K-map : a horizontal wrap-around and a vertical wrap-around.**
- **Every cell thus has 4 neighbours. For example, cell 0 corresponding to minterm 0 is adjacent to: cells *1, 2, 4, 8***



YZ \ WX	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

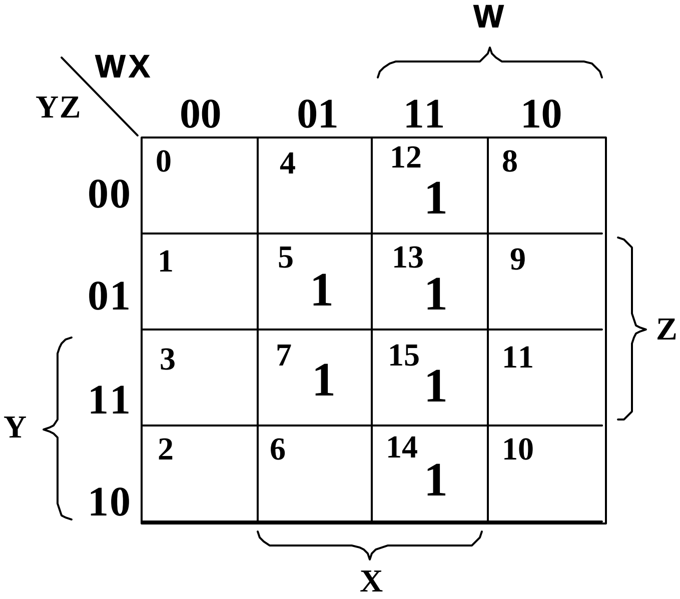
4-Variable K-map Example

For the function F(W,X,Y,Z) = $S_{W,X,Y,Z}$ (5,7,12,13,14,15)

Truth Table:

Row	W	X	Y	Z	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
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	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

K-map



YZ \ WX	00	01	11	10
00	0	4	12 1	8
01	1	5 1	13 1	9
11	3	7 1	15 1	11
10	2	6	14 1	10

Minimizing Sum of Products using K-maps

- **Each input combination with "1" in a Karnaugh map or truth table correspond to a minterm in the function's canonical sum representation.**
- **Pairs of adjacent "1" cells in the Karnaugh map indicate minterms that differ in only one variable.**
- **Using the generalization of T10, such adjacent minterm pairs can be combined into a single product term.**
- **In general, one can simplify a logic function by combining pairs of adjacent 1-cell minterms and writing a sum of products expression to cover all of the 1-cells.**

K-Map Minimization Rules and Definitions

- A minimal sum of a logic function $F(X_1, X_2, ..X_n)$ is a sum-of-products expression for F such that no other similar expression for F has fewer product terms, and other expressions with the same number of product terms have at least the same number of literals.

- A set of 2^i 1-cells are combined into a single square or rectangle if i variables take all 2^i possible combinations within the set while the remaining variables have the same value.

- The corresponding product term for the combined cells has n-i literals.

- Only the variables that have the same value appear in the resulting product term:
  - A variable in the resulting product term is complemented if it appears as 0 in all the 1-cells, and uncomplemented if it appears as 1.

Sum of Products Minimization Using K-maps

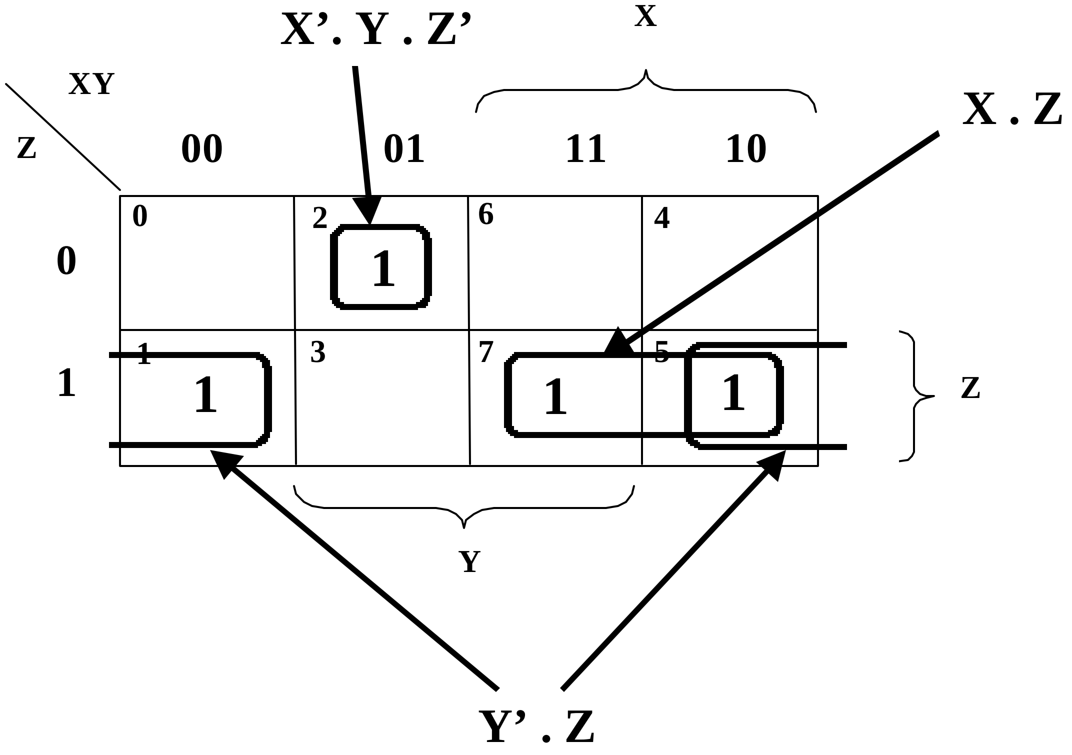
- **Group or combine as many adjacent 1-cells as possible:**
  - **The larger the group is, the fewer the number of literals in the resulting product term.**
  - **Each group of combined adjacent 1-cells must have a number of cells equal to *powers of two*: 1, 2, 4, 8, …**
  - **Grouping 2 adjacent 1-cells eliminates 1 variable, grouping 4 1-cells eliminates 2 variables, grouping 8 1-cells eliminates 3 variables, and so on. In general, grouping 2^n squares eliminates *n* variables.**
- **Select as few groups as possible to cover all the 1-cells (minterms) of the function:**
  - **The fewer the groups, the fewer the number of product terms in the minimized function.**

3-Variable K-map Minimization Example

- Using K-map, find a minimal sum of products (SOP) expression expression for the function:

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

K-map



Truth Table

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

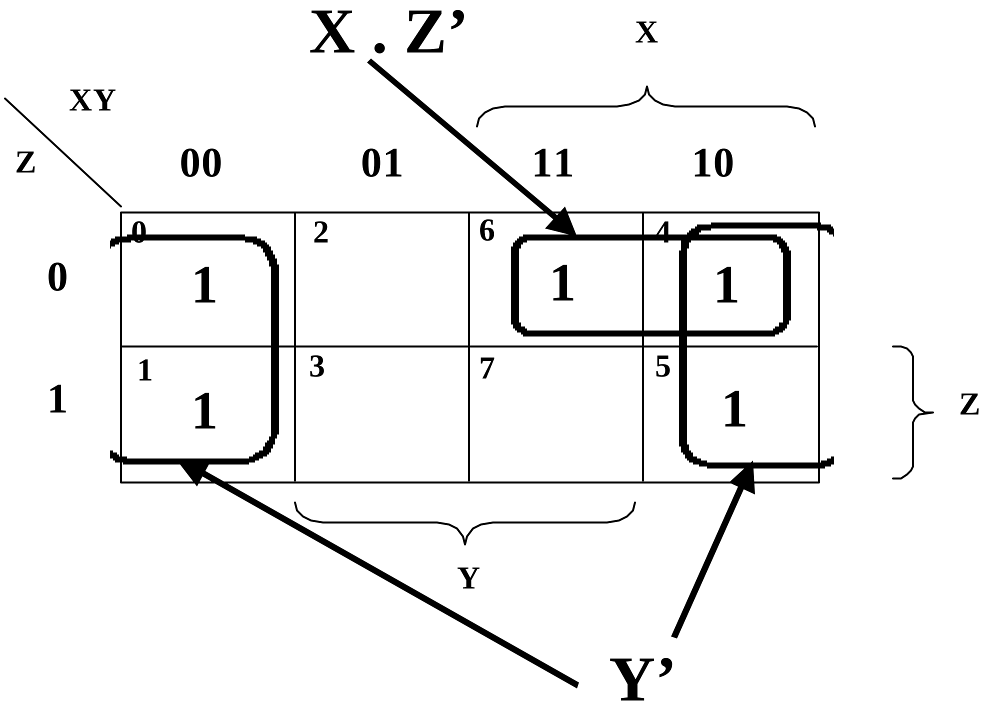
Minimum SOP for F = $X' . Y . Z' + X . Z + Y' . Z$

3-Variable K-map Minimization Example

- Using K-map, find a minimal sum of products (SOP) expression expression for the function:

$$F(X,Y,Z) = \Sigma_{X,Y,Z}(0,1,4,5,6)$$

K-map



Truth Table

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

Minimum SOP for F = Y' + X . Z'

4-Variable K-map Minimization Example

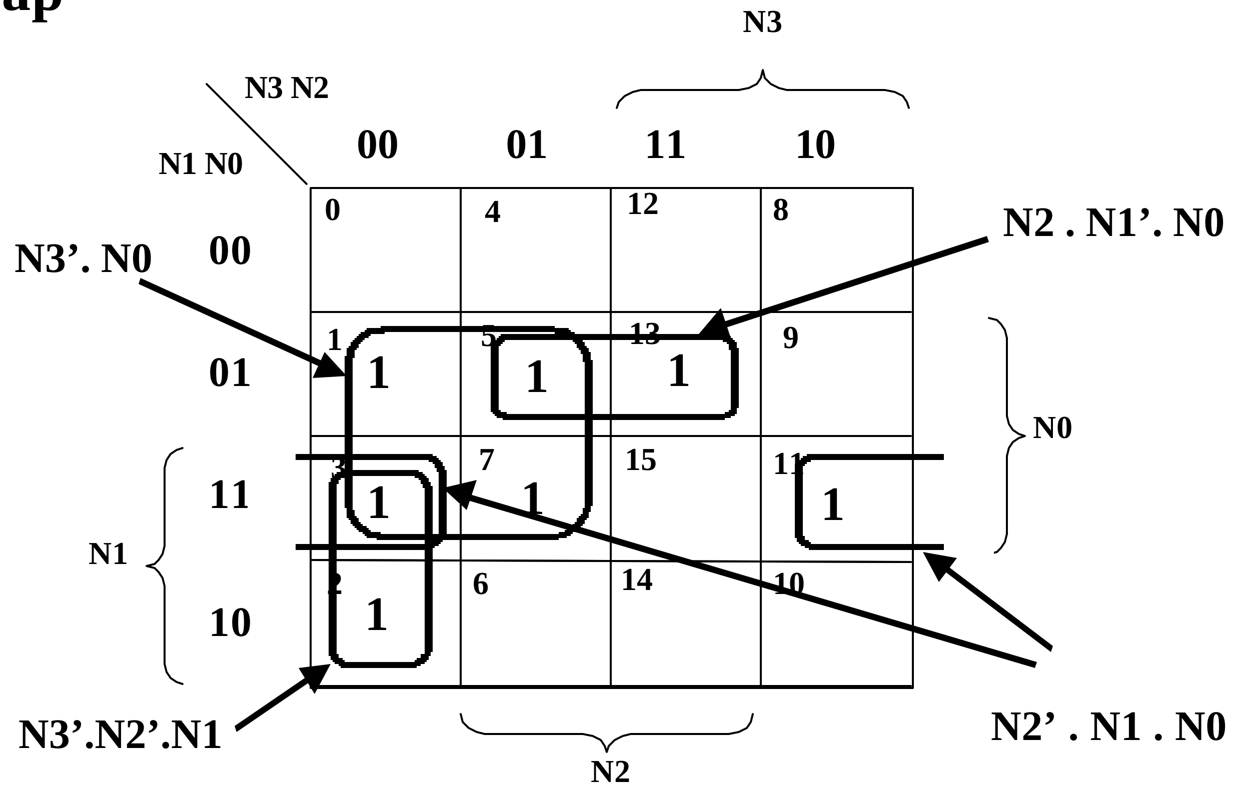
- Using K-map, find a minimal sum of products (SOP) expression expression for the function:

$$F(N3,N2,N1,N0) = S_{N3,N2,N1,N0}\ (1,2,3,5,7,11,13)$$

Truth Table:

Row	W	X	Y	Z	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

K-map



Minimum SOP for F = N3'. N0 + N3'. N2' . N1 + N2'. N1 . N0 + N2 . N1'.N0

K-Map Minimization Rules and Definitions

- A logic function P(X_1, X_2, ..X_n) implies a logic function F(X_1, …, X_n) if for every input combination such that P=1, then F=1 (F includes P , or F covers P).
- A prime implicant of a logic function F(X_1, ..X_n) is a normal product term P(X_1, ..X_n) that implies F, such that if any variable is removed from P, the the resulting product term does not imply F.
- A minimal sum is a sum of prime implicants (not necessarily all of them).
- A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.
- An essential prime implicant of a logic function is a prime implicant that covers one or more distinguished 1-cells and *must* be included every minimal sum expression for the function.

4-Variable K-map Minimization Example

- Using K-map, find a minimal sum of products (SOP) expression expression for the function: F(W,X,Y,Z) = S $_{W,X,Y,Z}$ (2,3,4,5,6,7,11,13,15)
- Also identify all prime implicants, distinguished 1-cells and the corresponding essential prime implicants that cover them.

K-map

From K-map:

Prime Implicants:

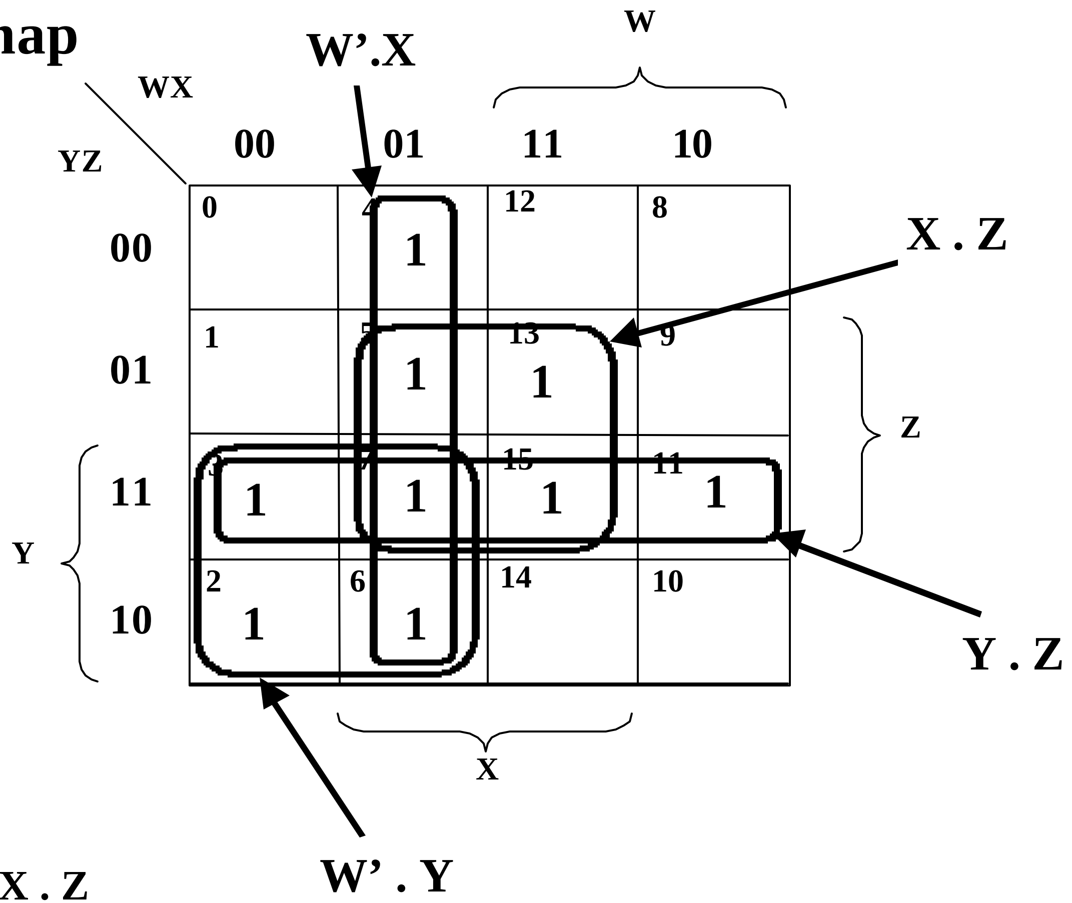
W'. Y   W' . X   Y . Z   X . Z

Distinguished 1-cells:

Cell 2 covered by W' . Y
Cell 4 covered by W' . X
Cell 11 covered by Y . Z
Cell 13 covered by X . Z

Here all prime implicants are essential prime implicants and all of them must be included in minimum SOP expression:

F = W' . Y + W'. X + Y . Z + X . Z



YZ \ WX	00	01	11	10
00	0	4 1	12	8
01	1	5 1	13 1	9
11	3 1	7 1	15 1	11 1
10	2 1	6 1	14	10

Minimization with Don't care Input Combinations

- In some cases, the output of a combinational circuit doesn't matter for certain input combinations.

- Such combinations are called don't cares and the output is represented in the truth table and K-maps as "d".

- When using K-maps to minimize such functions:
  - Allow d's to be included when circling sets of 1's to make the sets as large as possible.
  - Do not circle any set that only contains d's.

4-Variable K-map Minimization Example With Don't cares

- Using K-map, find a minimal sum of products (SOP) expression for prime BCD-digit detector which gives 1 when the input BCD digit is prime,
- Since the values 10-15 do not occur in a BCD digit minterms 10-15 are treated as don't cares giving the expression:

$$F(N3,N2,N1,N0) = \mathbf{S}_{N3,N2,N1,N0}\,(1,2,3,5,7) + d(10,11,12,13,14,15)$$

From K-map:

Prime Implicants:

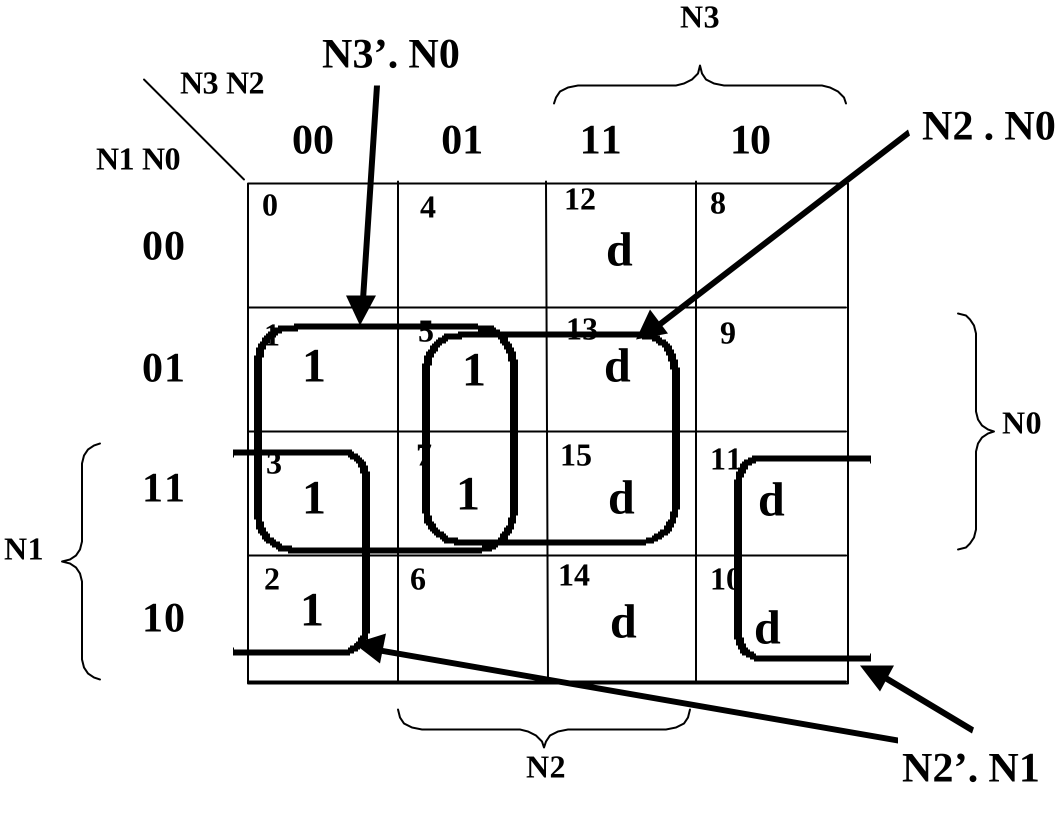
N3'. N0 N2'. N1 N2 . N0

Distinguished 1-cells:

Cell 1 covered by N3'. N0
Cell 2 covered by N2'. N1

Here not all prime implicants are essential prime implicants that must be included minimum SOP expression:

F = N3' . N0 + N2' . N1



N1 N0 \ N3 N2	00	01	11	10
00	0	4	12 d	8
01	1 1	5 1	13 d	9
11	3 1	7 1	15 d	11 d
10	2 1	6	14 d	10 d

K-map Minimization of Product of Sums

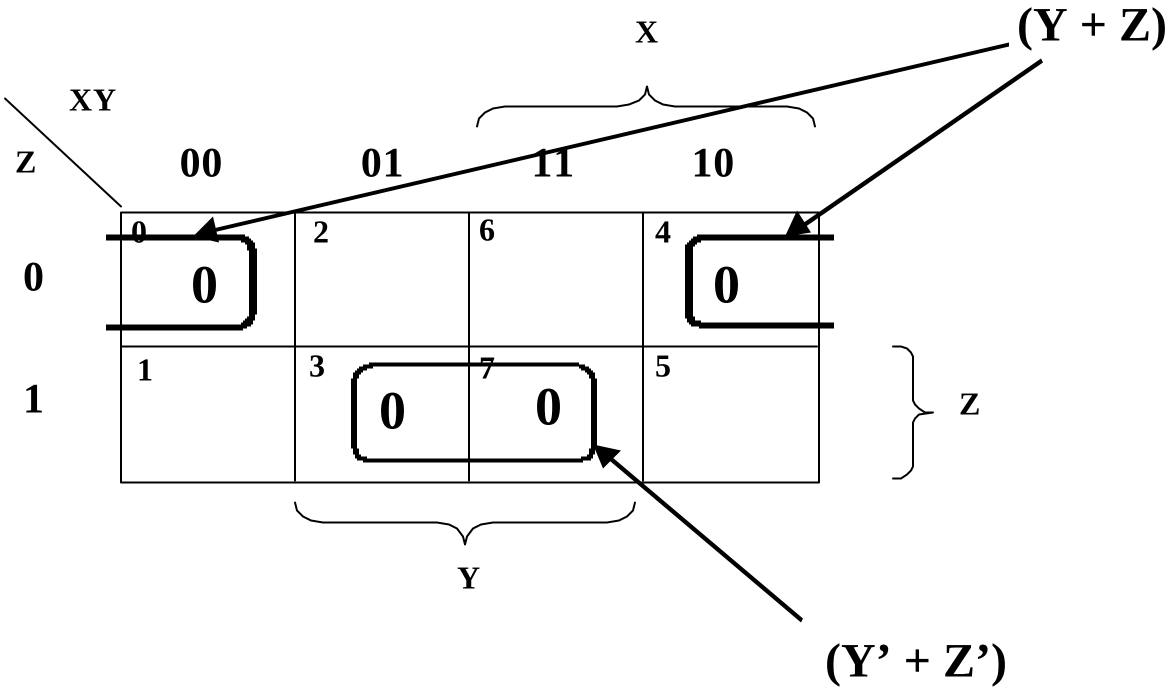
- **Similar to K-map minimization of sum of products by using duality and looking at 0-cells instead of 1-cells.**
- **A set of 2^i 0-cells may be combined if i variables take all 2^i possible combinations within the set while the remaining variables have the same value.**
- **In the resulting n-i literals sum term, a variable is complemented if it appears as 1 in all the 0-cells, and uncomplemented if it appears as 0.**
- **A prime implicate of a logic function $F(X_1, ..X_n)$, is a normal sum term $S(X_1, ..X_n)$ implied by F, such as if any variable is removed from S, then the resulting sum term is not implied by F.**
- **A minimal product is a product of prime implicates.**

K-map Product of Sums Minimization Example 1

- **Using K-map, find a minimal product of sums (POS) expression expression for the function:**

$$F(X,Y,Z) = P_{X,Y,Z}\ (0,3,4,7)$$

K-map



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

Truth Table

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	1
7	1	1	1	0

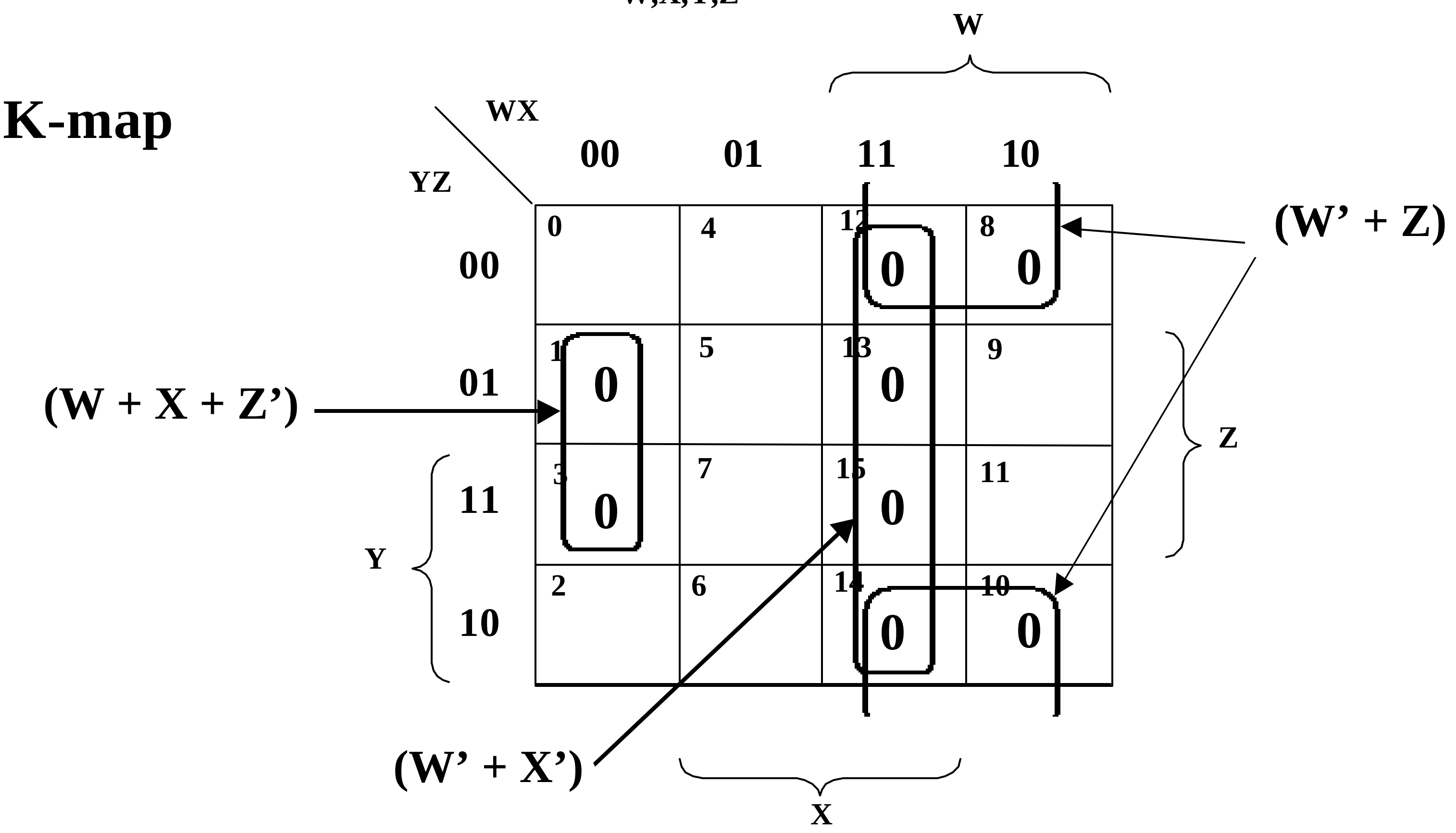
Minimum POS for F = (Y + Z) . (Y’ + Z’)

K-map Product of Sums Minimization Example 2

- Using K-map, find a minimal product of sums (POS) expression expression for the function:

$$F(W,X,Y,Z) = \mathbf{P}_{W,X,Y,Z}\ (1,3,8,10,12,13,14,15)$$

K-map

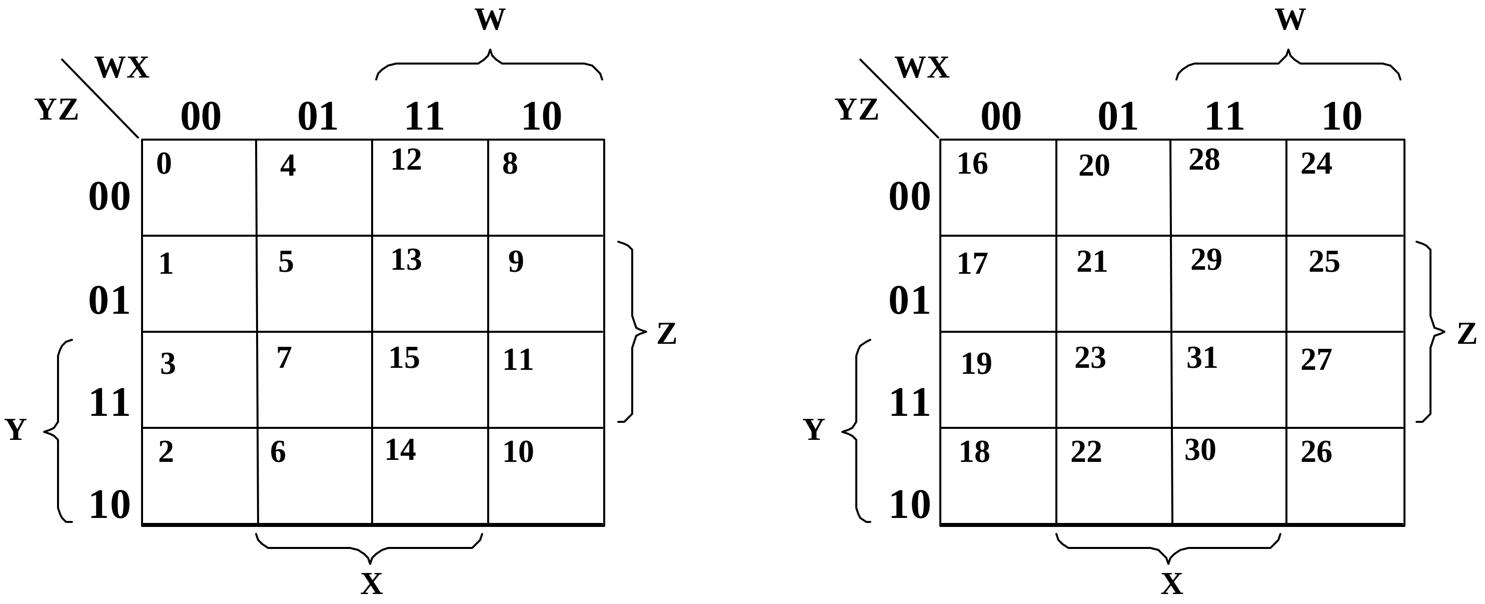


YZ \ WX	00	01	11	10
00	0	4	12 0	8 0
01	1 0	5	13 0	9
11	3 0	7	15 0	11
10	2	6	14 0	10 0

Minimum POS for F = (W + X + Z') . (W' + Z) . (W' + X')

5-variable K-maps

- The K-map for a 5-variable logic function F(V,W,X,Y,Z) is organized as two 4-variable K-maps:



YZ \ WX	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

V = 0

YZ \ WX	00	01	11	10
00	16	20	28	24
01	17	21	29	25
11	19	23	31	27
10	18	22	30	26

V = 1

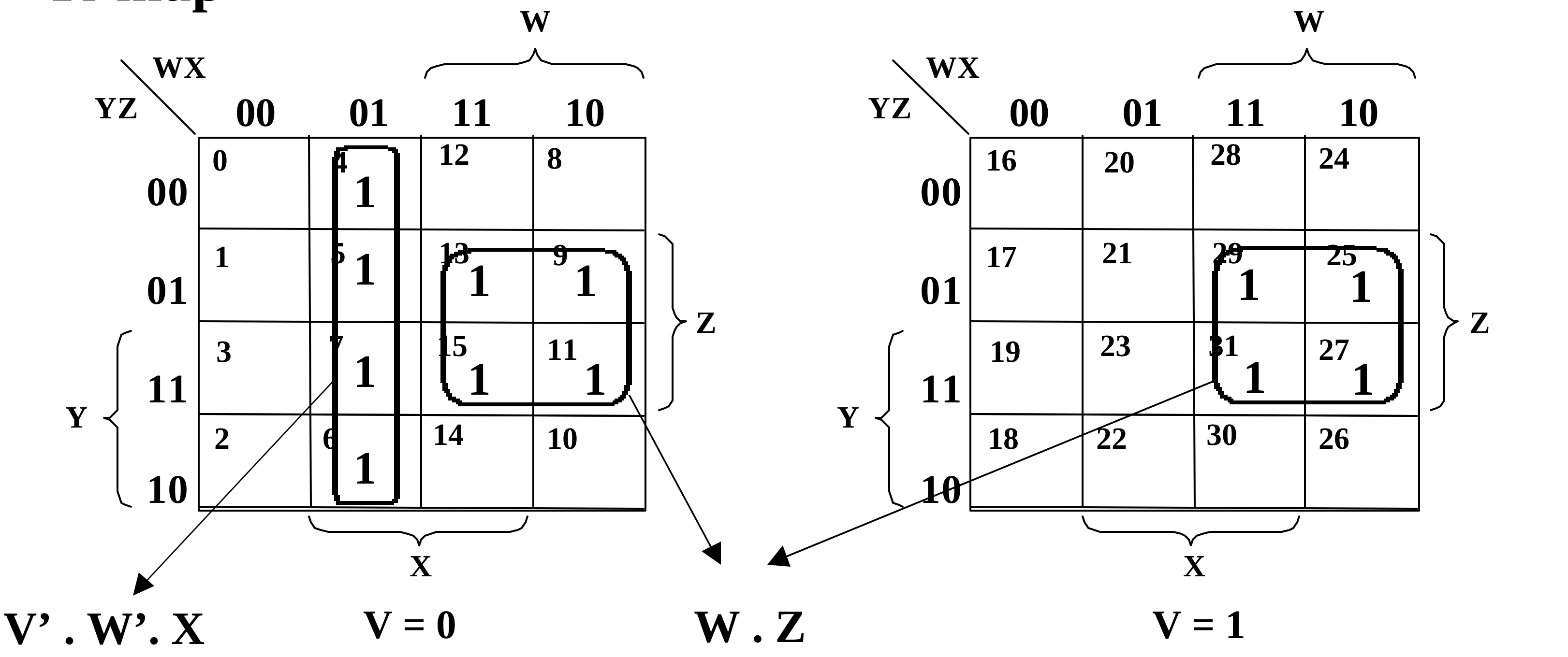
Corresponding squares of each map are adjacent.
Can be visualised as being one 4-variable map on
top of another 4-variable map.

5-Variable K-map SOP Minimization Example

- **Using K-map, find a minimal sum of products (SOP) expression expression for the function:**

$$F(V,W,X,Y,Z) = S_{V,W,X,Y,Z}\ (4,5,6,7,9,11,13,15,25,27,29,31)$$

K-map



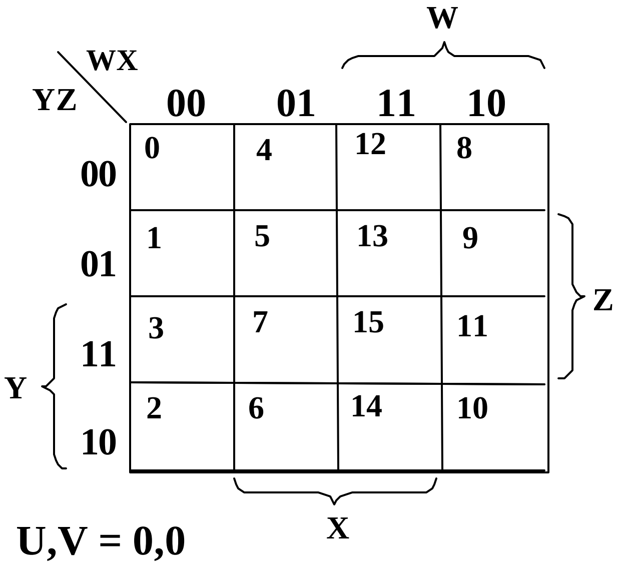
YZ \ WX	00	01	11	10
00	0	4 1	12	8
01	1	5 1	13 1	9 1
11	3	7 1	15 1	11 1
10	2	6 1	14	10

YZ \ WX	00	01	11	10
00	16	20	28	24
01	17	21	29 1	25 1
11	19	23	31 1	27 1
10	18	22	30	26

Minimum SOP for F = V' . W'. X + W . Z

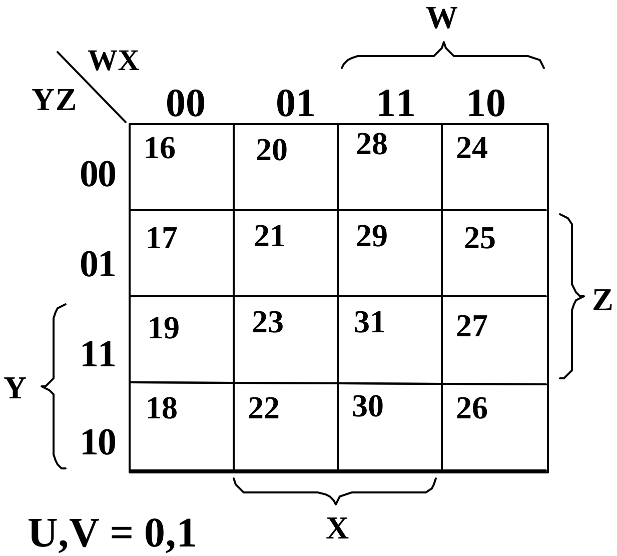
6-variable K-maps

K-map for a 6-variable logic function F(U,V,W,X,Y,Z) is organized as two 5-variable K-maps:



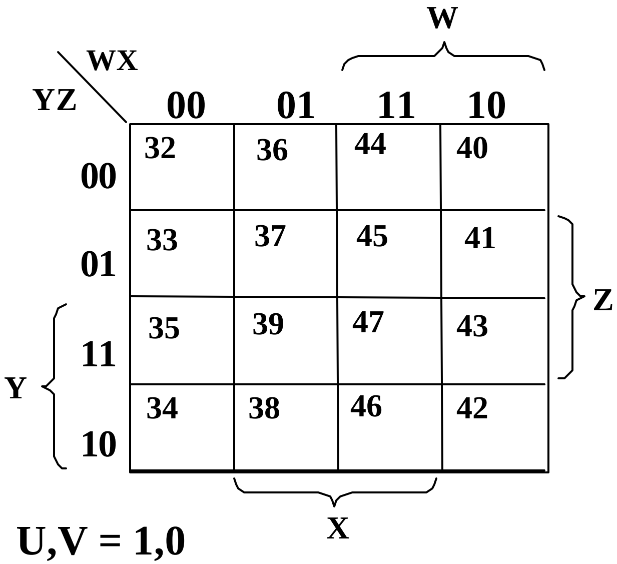
YZ \ WX	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

U,V = 0,0



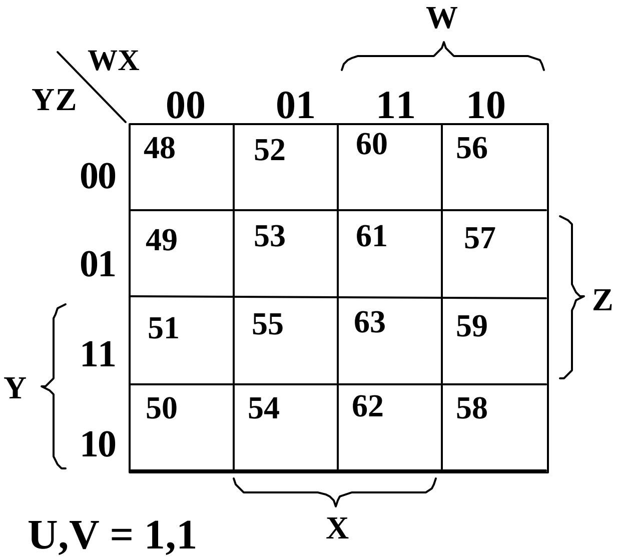
YZ \ WX	00	01	11	10
00	16	20	28	24
01	17	21	29	25
11	19	23	31	27
10	18	22	30	26

U,V = 0,1



YZ \ WX	00	01	11	10
00	32	36	44	40
01	33	37	45	41
11	35	39	47	43
10	34	38	46	42

U,V = 1,0

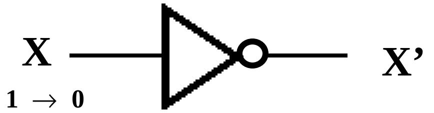


YZ \ WX	00	01	11	10
00	48	52	60	56
01	49	53	61	57
11	51	55	63	59
10	50	54	62	58

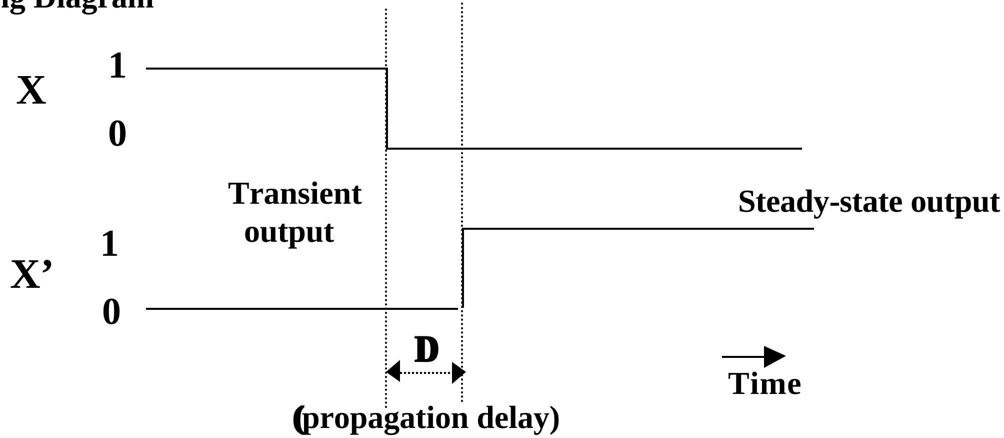
U,V = 1,1

Combinational Logic Circuit Transient Vs. Steady-state Output

- **Gate propagation delay: The time between an input change and the corresponding change of the output.**
- **Circuit steady-state output: The output is evaluated when the inputs have been stable for a long time relative to the gate delays.**
- **Circuit transient output behavior: The circuit output when one or more inputs change values.**
- **Example: For an inverter with propagation delay, D when input changes from 1 to 0:**



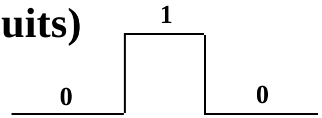
Timing Diagram



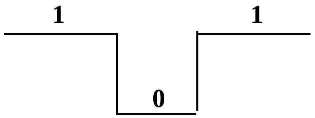
- **The circuit analysis done so far ignores propagation delays and considers only steady-state output when all propagation delays have completed though all the circuit gates.**

Combinational Logic Hazards

- **Output glitch: A momentary unexpected transient output change (short pulse) when an input changes and usually caused by gate propagation delays.**
- **Hazards: A hazard exists in a combinational circuit when it produces an output glitch when one or more inputs change.**
- **Types of combinational logic hazards:**
- **Static Hazards:**
  - **Static-1 Hazard: The output should be 1 but goes momentary to 0 as a result of an input change. (possible in AND-OR circuits)**
  - **Static-0 Hazard: The output should be 0 but goes momentary to 1 as a result of an input change. (possible in OR-AND circuits)**

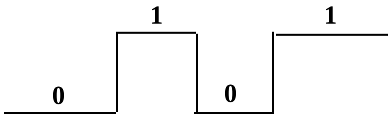


Static-0 Hazard



Static-1 Hazard

- **Dynamic Hazards: The output changes more than once as a result of a single input change (impossible in 2-level circuits).**



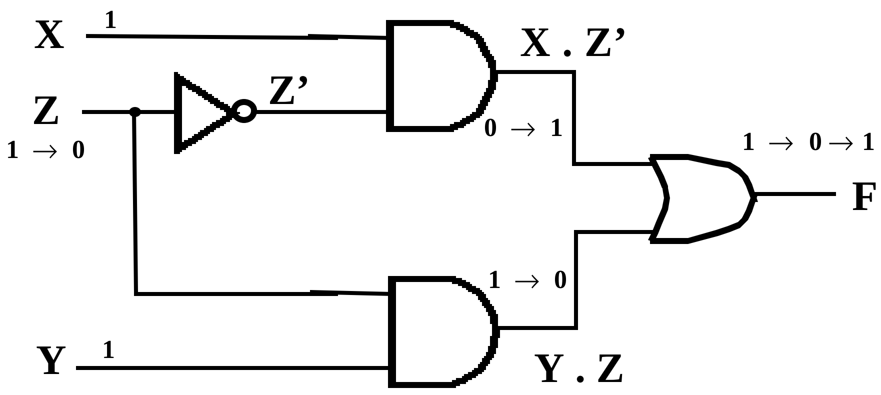
Dynamic Hazard Example

- **Static hazards can be detected and eliminated for 2-level logic circuits using K-maps.**

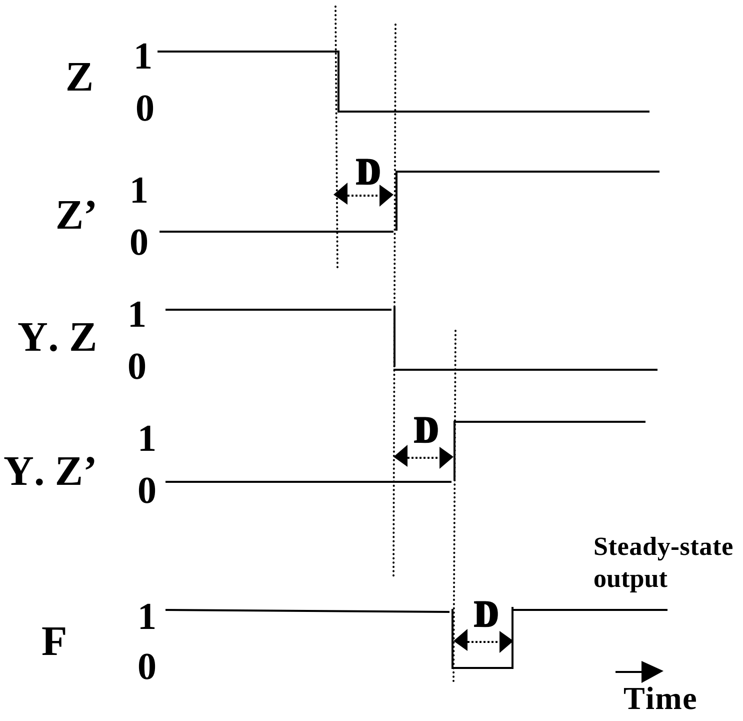
Example: Circuit with Static-1 Hazard

- A static-1 hazard exists in the following AND-OR circuit when X = 1, Y = 1 and Z changes from 1 to 0 (assume all gates have propagation delay D):

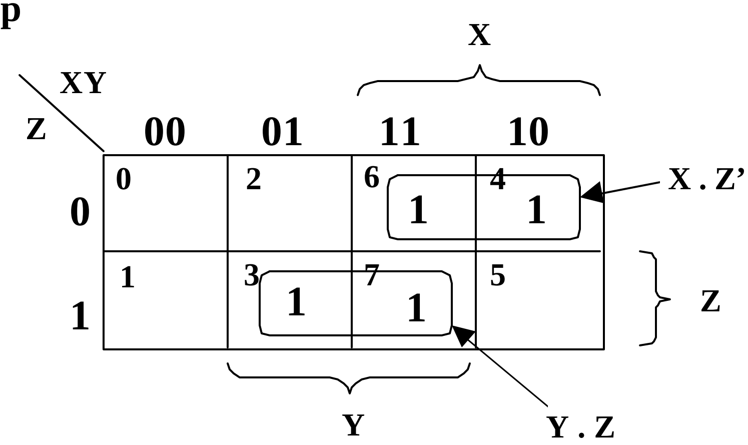
Circuit



Timing Diagram



K-map

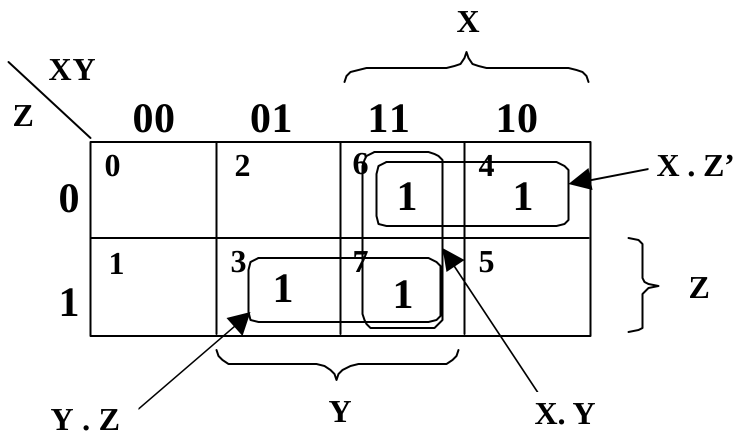


Z \ XY	00	01	11	10
0	0	2	6 1	4 1
1	1	3 1	7 1	5

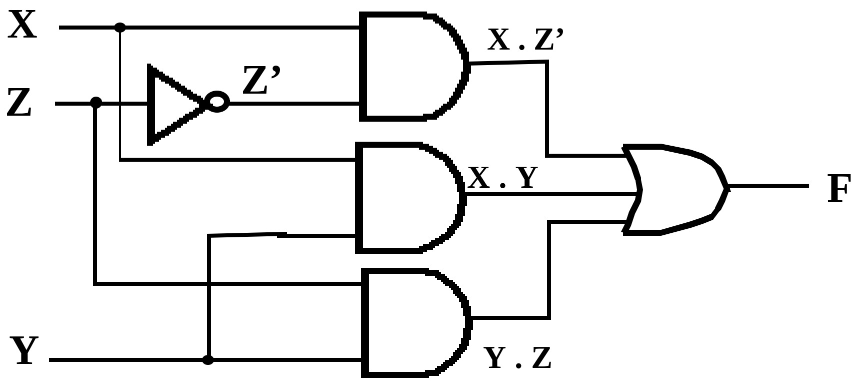
$$F = X \, . \, Z' \; + \; Y \, . \, Z$$

Eliminating Static-1 Hazards Using K-maps

- **A static-1 hazard occurs in AND-OR circuits when an input variable and its complement are connected to two different AND gates.**
- **Static-1 hazards are found using k-maps by finding adjacent 1 cells that are covered by different product terms.**
- **To eliminate static-1 hazards, additional product terms (prime implicants) are needed to cover such cells thus covering the transition of the variable causing the hazard.**
- **For in the previous example the static-1 hazard is eliminated by including the additional product term X . Y**



Circuit with static-1 hazard eliminated



New F = X . Z’ + Y . Z + X. Y

Eliminating Static-0 Hazards Using K-maps

- A static-0 hazard occurs in OR-AND circuits when an input variable and its complement are connected to two different OR gates.
- The procedure to find and eliminate static-0 hazards using K-maps is done in a dual way to finding static-1 hazards.
- Static-0 hazards are found using k-maps by finding adjacent 0 cells that are covered by different sum terms.
- To eliminate static-0 hazards, additional sum terms (prime implicates) are needed to cover such cells thus covering the transition of the variable causing the hazard.