ECE 27000 S22 P 4

Chapter 3. Switching Algebra and Combinational Logic (Part 2)

Combinational Circuit Analysis

Analysis of a circuit means obtaining a description of its logic function.
Once the function is available, it can be used for the following.

• Determining the behavior of the circuit for different input combinations.

• Manipulating the function to obtain different circuit elements or structures for the same function.

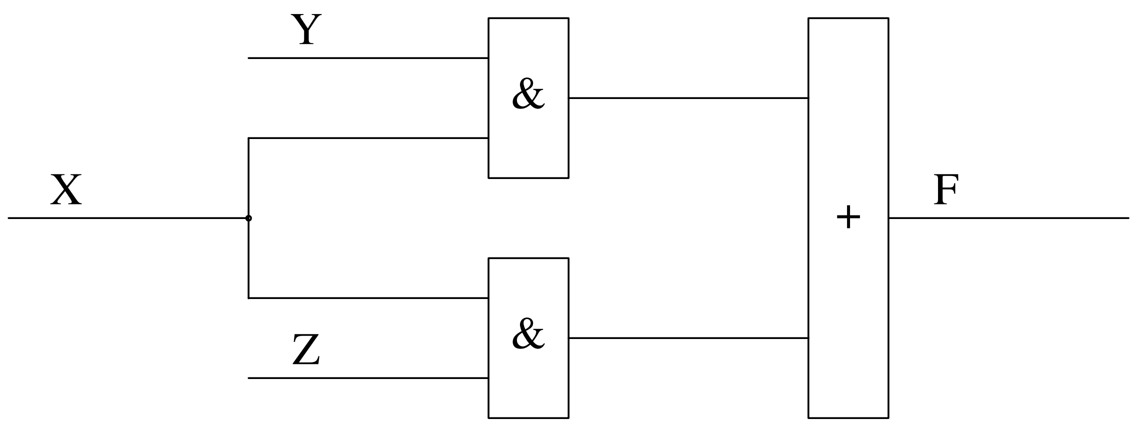
• Transforming the description into a standard form such as a sum-of-products expression.
This may be needed in order to implement the function using certain circuit structures such as PLAs (programmable logic arrays) or FPGAs (field programmable gate arrays).

• Transforming the description into a canonical form such as a canonical sum.
This can be done in order to check if different circuits are functionally identical (a canonical form is unique).

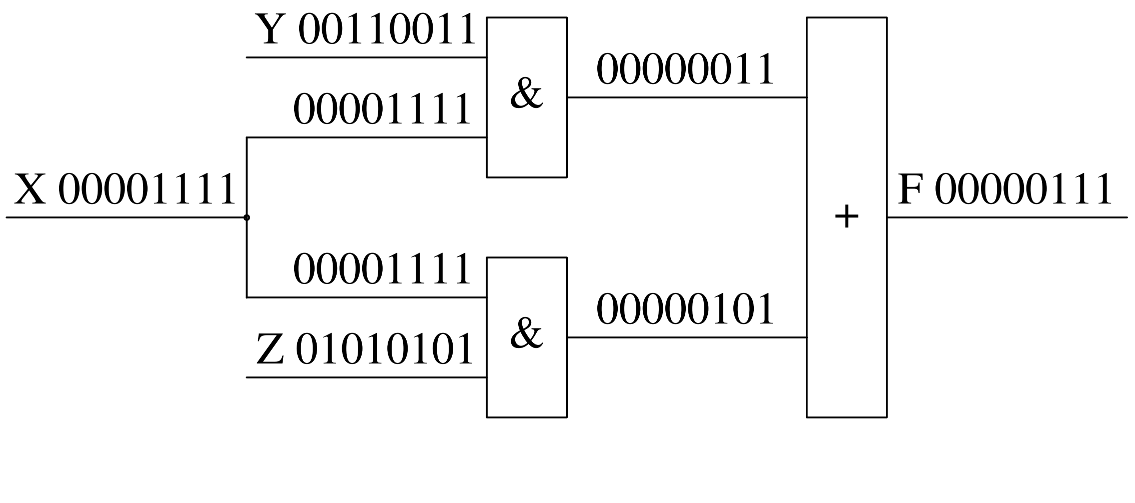
• Obtaining an algebraic description of the function of the circuit.
This can be used in the analysis of a larger systems that contains the circuit.

Given a circuit, it is possible to obtain its truth table by considering all the 2^n input combinations.
Each combination is applied to the inputs.
Gate output values are computed proceeding from the inputs to the output.

Example:



Example:

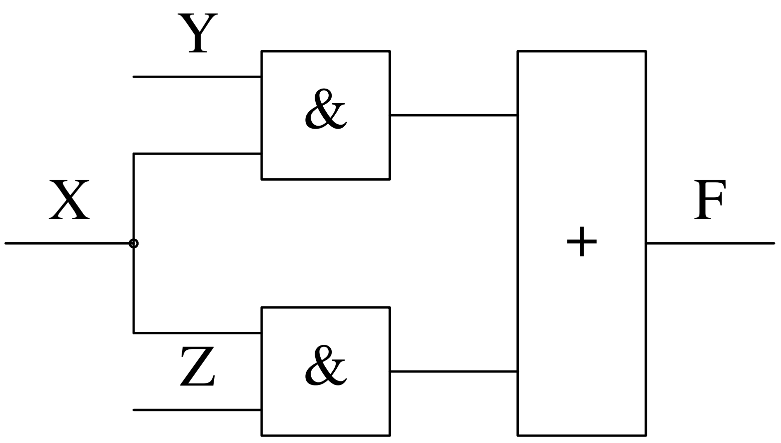


This approach is not feasible for a circuit with *n* inputs when *n* is large.

An approach where expressions grow polynomially with the size of the circuit is to build up a logic expression corresponding to the circuit by traversing the circuit from the inputs toward the output(s).

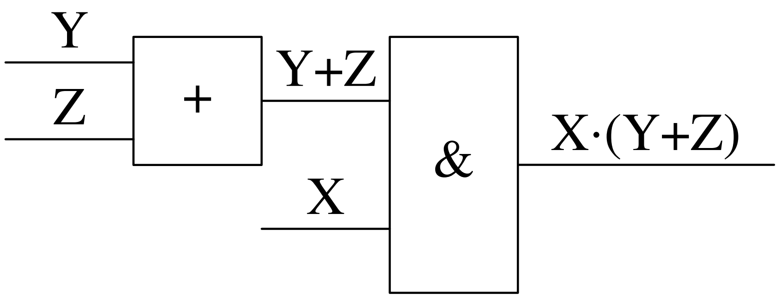
It is possible to simplify the expressions obtained during this process, and at its end, using switching algebra theorems.

Example:



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

Another implementation for the same function:

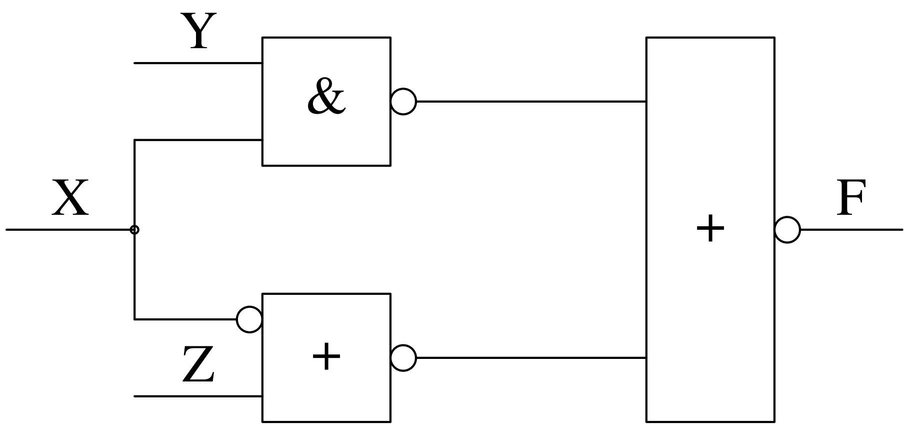


Note that in both examples we traversed the circuit from its inputs to its output.

We could compute an output value or write an output function for a gate only after we knew its input values or input functions.

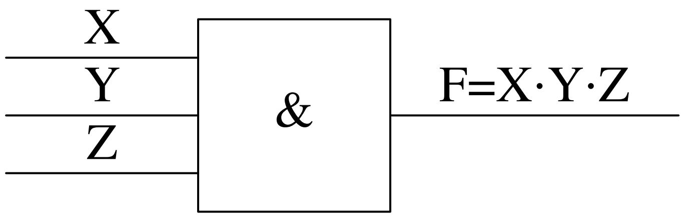
This order is defined by an algorithm called topological sort.

An example using DeMorgan’s theorem:



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

Another implementation of the same function:

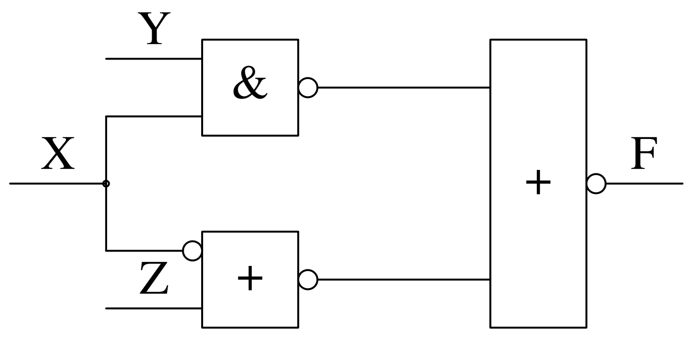


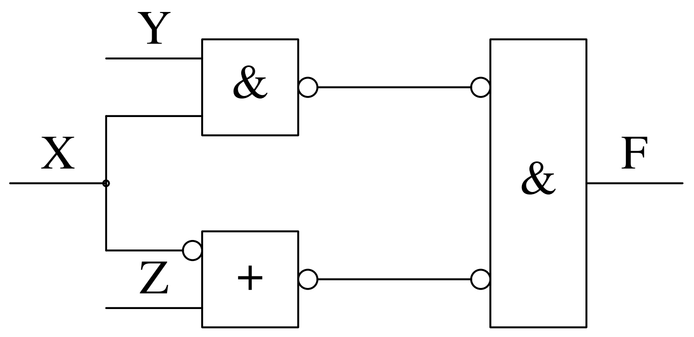
DeMorgan’s theorem can be applied to the description of the circuit in order to simplify the analysis.

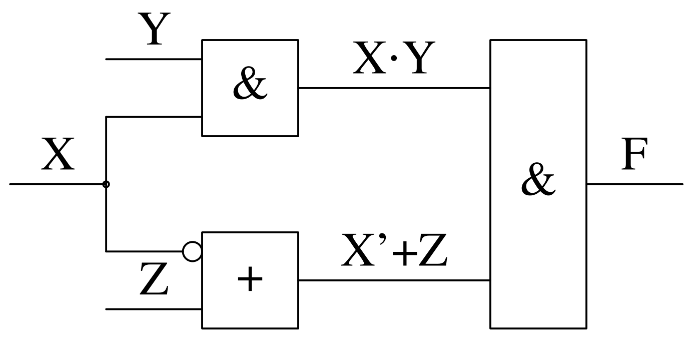
This is done by replacing:
A NAND gate with an OR gate whose inputs are complemented, using the fact that
$(X \cdot Y)' = X' + Y'$.
A NOR gate with an AND gate whose inputs are complemented, using the fact that
$(X + Y)' = X' \cdot Y'$.

When this creates two inverters on the same line, the two inverters can be removed.
This reduces the number of complementations we need to do during the analysis.

In the previous example:







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

Combinational Synthesis

Synthesis starts from a specification (possibly in words) of a function that a circuit needs to implement.
It produces a logic circuit that performs the function.

In a typical design process, a designer writes code for implementing the function in a hardware description language (HDL).
A software tool synthesizes a circuit from the HDL program.

We will start from a word description that has been translated into a specification using a truth table or algebraic equations, and produce a logic circuit made up of gates.

Designers need knowledge of the synthesis process to understand how the synthesis tools work and write better circuit descriptions.
In addition, there are situations where parts of a circuit need to be synthesized manually.

When a specification is translated into a truth table, synthesis starts from a canonical sum or product expression.

Example: a four-input prime number detector can be described as
$F = \sum_{W,X,Y,Z}(1, 2, 3, 5, 7, 11, 13)$.

In other cases it is easier to write expressions without going through the truth table.

Example: a function that indicates there is an error when two or more of four traffic lights are not red can be written as follows.

Let us use $RED1$, $RED2$, $RED3$ and $RED4$ to capture the status of the traffic lights (red or not).

An error should be indicated when any two variables are 0 together.

$$ERROR = RED1' \cdot RED2' + RED1' \cdot RED3' +$$
$$RED1' \cdot RED4' + RED2' \cdot RED3' +$$
$$+RED2' \cdot RED4' + RED3' \cdot RED4'.$$

We can double check that we captured all the possible situations by considering the truth table:

RED1	RED2	RED3	RED4	ERROR
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

It is also possible to write the function considering the cases where there is no error - the four cases where only one traffic light that is not red, and the case where all are red.

$ERROR = (RED1' \cdot RED2 \cdot RED3 \cdot RED4 +$
$RED1 \cdot RED2' \cdot RED3 \cdot RED4 +$
$RED1 \cdot RED2 \cdot RED3' \cdot RED4 +$
$RED1 \cdot RED2 \cdot RED3 \cdot RED4' +$
$RED1 \cdot RED2 \cdot RED3 \cdot RED4)'.$

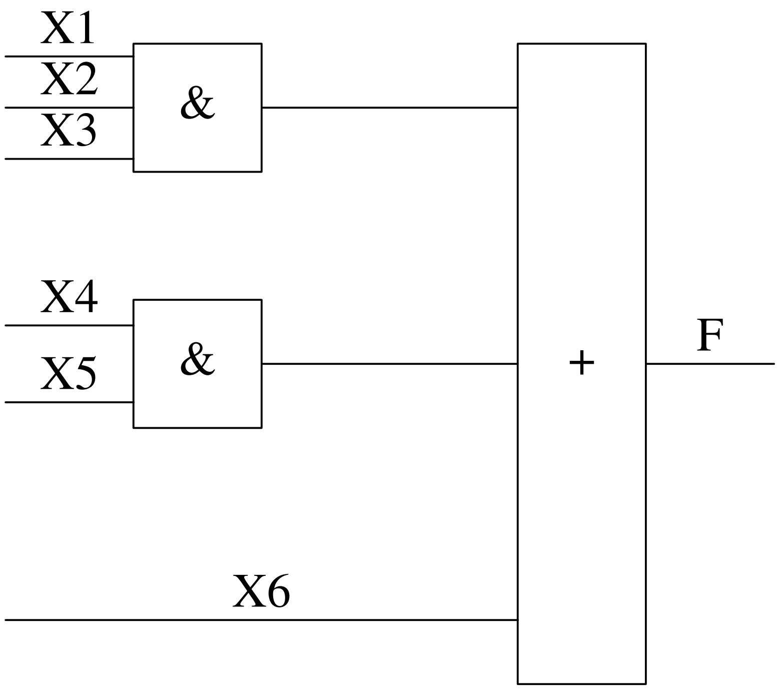
The two functions would give different circuits if we implemented them without further simplification.

The synthesis process includes a simplification process that is likely to produce the same expression regardless of the starting point given to it.

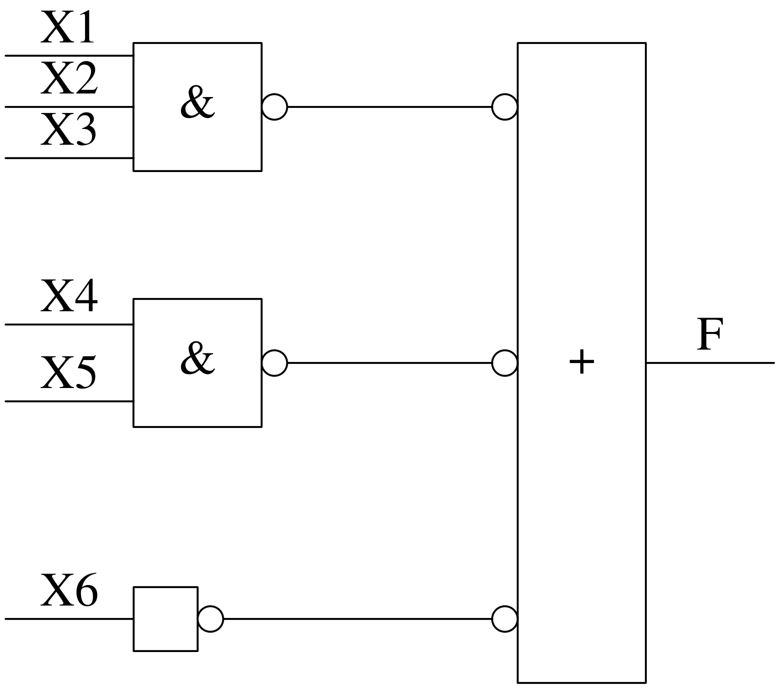
The synthesis process we will see produces minimal sum-of-products or product-of-sums expressions.

If we write a function in sum-of-products form, a natural implementation uses a level of AND gates followed by an OR gate, with inverters on the inputs as needed. This is called an AND-OR circuit.

Example: $F = X_1 \cdot X_2 \cdot X_3 + X_4 \cdot X_5 + X_6$ can be implemented by the following circuit.



It is possible to obtain a NAND-based circuit by adding two inverters on every line connecting an AND gate with the OR gate.
Note that X_6 will need an inverter.
This is called a NAND-NAND circuit.



In a similar way, if we write a function in product-of-sums form, a natural implementation uses a level of OR gates followed by an AND gate, with inverters on the inputs as needed. This is called an OR-AND circuit.

It is possible to obtain a NOR-based circuit by adding two inverters on every line connecting an AND gate with the OR gate.
Other inputs of the AND gate will also need inverters.
This is called a NOR-NOR circuit.

Combinational Circuit Minimization

A given expression for a function may be larger than necessary, and result in a circuit that is larger than necessary.

Canonical sums and products are typically larger than necessary.
Example: $F(X, Y, Z) = X \cdot Y + Y \cdot Z$.
The canonical sum is:
$X' \cdot Y \cdot Z + X \cdot Y \cdot Z' + X \cdot Y \cdot Z$.

The canonical sum and product can be obtained from the truth table of the function, or by adding the missing literal(s) to every product term.

This process is the reverse of the minimization process we will see.

Example: $F(X,Y,Z) = X \cdot Y + Y \cdot Z$

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

Example of a canonical product:

$F(X,Y,Z) = (X + Y) \cdot Z =$

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

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

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

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

$(Z + X + Y) \cdot (Z + X' + Y) \cdot$

$(Z + X + Y') \cdot (Z + X' + Y')$

Minimization is the process of reducing the size of a circuit that implements a given function. It is typically done by manipulating the function so as to:

• Reduce the number of gates.
• Reduce the number of gate inputs.

Inverters on the inputs are typically considered free.

Other considerations include delay and power consumption.

Most minimization methods are based on the application of the following rules:
(T10) $X \cdot Y + X \cdot Y' = X$
(T10D) $(X + Y) \cdot (X + Y') = X$
where X may be an expression.

For example, let us try to reverse the example we saw earlier.
$F(X, Y, Z) = X' \cdot Y \cdot Z + X \cdot Y \cdot Z' + X \cdot Y \cdot Z =$
$(X' + X) \cdot Y \cdot Z + X \cdot Y \cdot Z' =$
$Y \cdot Z + X \cdot Y \cdot Z'$

We need to duplicate the term $X \cdot Y \cdot Z$ before starting the minimization (or use other switching algebra theorems) to complete the minimization.

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

There are methodical ways to do this, which we will see next.

Karnaugh Maps

Karnaugh maps represent the truth table of a logic function in a way that is convenient to apply the minimization rules.

A Karnaugh map for a three-variable function:

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

The numbers represent the minterms:

0	000	$X' \cdot Y' \cdot Z'$
1	001	$X' \cdot Y' \cdot Z$

$\ldots$

A Karnaugh map for a four-variable function:

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

Karnaugh maps for higher numbers of variables exist but they become more and more complex.

To represent a function using a Karnaugh map we enter the 0s and 1s of the function into the corresponding cells of the map.

Example:
$F(X, Y, Z) =$
$= X' \cdot Y \cdot Z + X \cdot Y \cdot Z' + X \cdot Y \cdot Z =$
$= \sum_{X,Y,Z}(3, 6, 7)$

		XY			
		00	01	11	10
Z	0				
	1				

		XY			
		00	01	11	10
Z	0	0	0	1	0
	1	0	1	1	0

A Karnaugh map for the function
$F(W, X, Y, Z) = \sum_{W,X,Y,Z}(1, 2, 3, 5, 7, 11, 13)$:

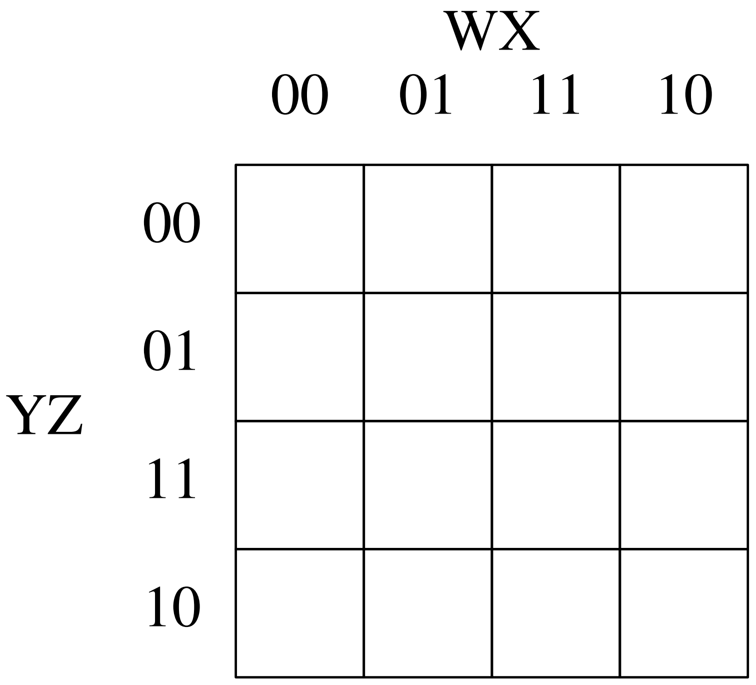
		WX			
		00	01	11	10
	00				
	01				
YZ	11				
	10				

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

The property of a K-map that makes it useful for minimization:
Every two adjacent cells differ in the value of a single variable.
This is achieved by using Gray code for the rows and columns.

Two cells in the two extreme columns of the same row are adjacent.

Two cells in the two extreme rows of the same column are adjacent.



For sums-of-products, we are interested in the minterms where the function is 1 (the 1 cells in the K-map).

Two adjacent 1 cells correspond to two minterms that can be combined into a single product term by using the rule

$$X \cdot Y + X \cdot Y' = X$$

where X may be a product term.

For example, in the following K-map:

		WX			
		00	01	11	10
YZ	00				
	01	1	1	1	
	11	1	1		1
	10	1			

$W' \cdot X' \cdot Y' \cdot Z$ (0001) and $W' \cdot X \cdot Y' \cdot Z$ (0101) can be combined to form the term $W' \cdot Y' \cdot Z$ (0-01).
$W' \cdot X' \cdot Y \cdot Z$ (0011) and $W \cdot X' \cdot Y \cdot Z$ (1011) can be combined to form the term $X' \cdot Y \cdot Z$ (-011).
This is marked by circling the minterms.

If two pairs of minterms (a total of four minterms) are adjacent such that they are the same in all but two variables, it is possible to combine them into a single product term.

For example, in the K-map above,
$W' \cdot X' \cdot Y' \cdot Z$ (0001) and $W' \cdot X \cdot Y' \cdot Z$ (0101) can be combined into $W' \cdot Y' \cdot Z$ (0-01).
$W' \cdot X' \cdot Y \cdot Z$ (0011) and $W' \cdot X \cdot Y \cdot Z$ (0111) can be combined into $W' \cdot Y \cdot Z$ (0-11).
All four can be combined into $W' \cdot Z$ (0--1).
This is marked by circling all four terms.

		WX			
		00	01	11	10
	00				
YZ	01	1	1	1	
	11	1	1		1
	10	1			

Once we circle the adjacent minterms, it is possible to determine the product term from the K-map by using the following rules:

• If a circle covers only areas of the map where a variable is 0, the variable is complemented in the product term.

• If a circle covers only areas of the map where a variable is 1, the variable is uncomplemented in the product term.

• If a circle covers areas of the map where a variable is 0 and 1, the variable does not appear in the product term.

A sum-of-products expression for a function must contain product terms that cover all the 1s and none of the 0s of the map.
It is allowed to cover the same 1 twice.

Example:

		WX			
		00	01	11	10
	00				
	01	1	1	1	
YZ	11	1	1		1
	10	1			

$$F = W' \cdot Z + X \cdot Y' \cdot Z + X' \cdot Y \cdot Z + W' \cdot X' \cdot Z.$$

From this expression it is now possible to obtain a circuit.

For a more formal description of the minimization process we need the following definitions.

A *minimal sum* of a logic function $F(X_1, \cdots, X_n)$ is a sum-of-products expression for F that has the smallest number of product terms, and the smallest number of literals (in this order of importance).

A logic function $P(X_1, \cdots, X_n)$ *implies* a logic function $F(X_1, \cdots, X_n)$ if for every input combination such that $P = 1$, $F = 1$ also.
We also say that F *includes* P, or that F *covers* P.

A *prime implicant* of a logic function $F(X_1, \cdots, X_n)$ is a product term $P(X_1, \cdots, X_n)$ that implies F, such that if any variable is removed from P, then the resulting product term does not imply F.

In a K-map, a prime implicant of F is a circled set of minterms of the largest possible size (if we try to increase the circle by doubling its size, it will have to contain 0s).

$$\begin{array}{cc|c|c|c|c|}
 & & \multicolumn{4}{c}{\text{WX}} \\
 & & 00 & 01 & 11 & 10 \\
\cline{3-6}
 & 00 & & & & \\
\cline{3-6}
 & 01 & 1 & 1 & 1 & \\
\cline{3-6}
\text{YZ} & 11 & 1 & 1 & & 1 \\
\cline{3-6}
 & 10 & 1 & & & \\
\cline{3-6}
\end{array}$$

Prime-Implicant Theorem: A minimal sum is a sum of prime implicants.

This implies that in order to find a minimal sum, we only need to consider prime implicants.

Instead of a non-prime implicant P, it is possible to consider a prime implicant $\hat{P}$ that covers P.
$\hat{P}$ contains fewer literals than P.
Therefore, it is more suitable for a minimal sum.

In some cases a minimal sum includes all the prime implicants of the function.
Example:

		WX			
		00	01	11	10
	00			1	
	01		1	1	
YZ	11		1	1	
	10			1	

The function has two prime implicants and both are included in the minimal sum.

In other cases, the minimal sum includes a subset of the prime implicants.
Example:

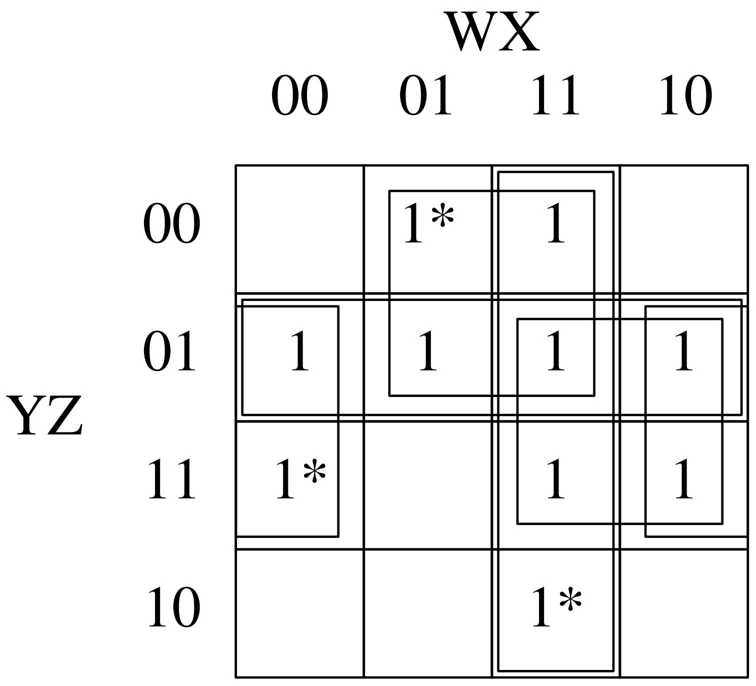
			WX		
		00	01	11	10
	00		1	1	
	01	1	1	1	1
YZ	11	1		1	1
	10			1	

The function has five prime implicants.
The minimal sum includes only three of them.

To select prime implicants for a minimal sum we need the following definitions.

A *distinguished minterm* of a logic function is a minterm 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 minterms.



For a distinguished minterm to be covered by a minimal sum, the minimal sum needs to include the corresponding essential prime implicant.

The first steps of the minimization procedure:
1. Find all the prime implicants.
2. Identify the distinguished minterms and the essential prime implicants.
3. Include the essential prime implicants in the minimal sum.

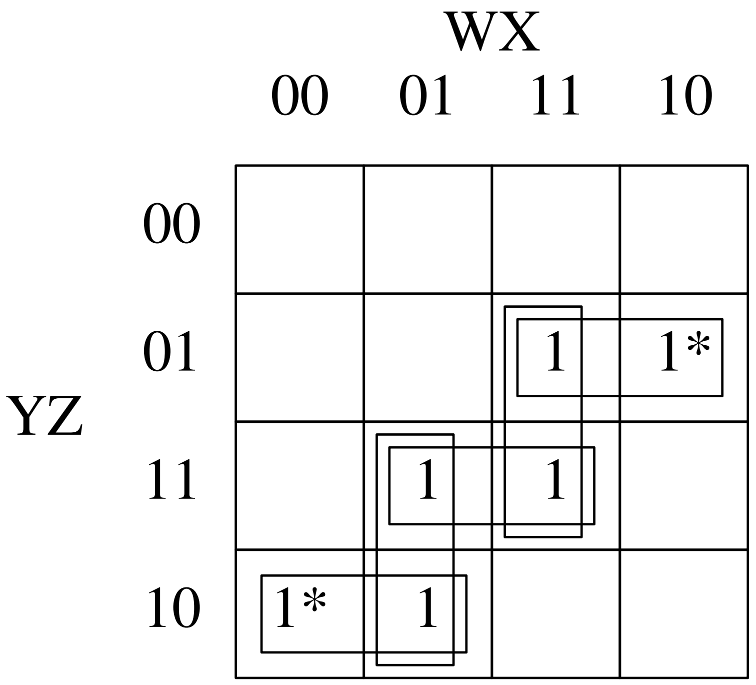
In the last example, no additional minterms remain to be covered.

If any minterms remain, it is necessary to select additional prime implicants in order to cover them.

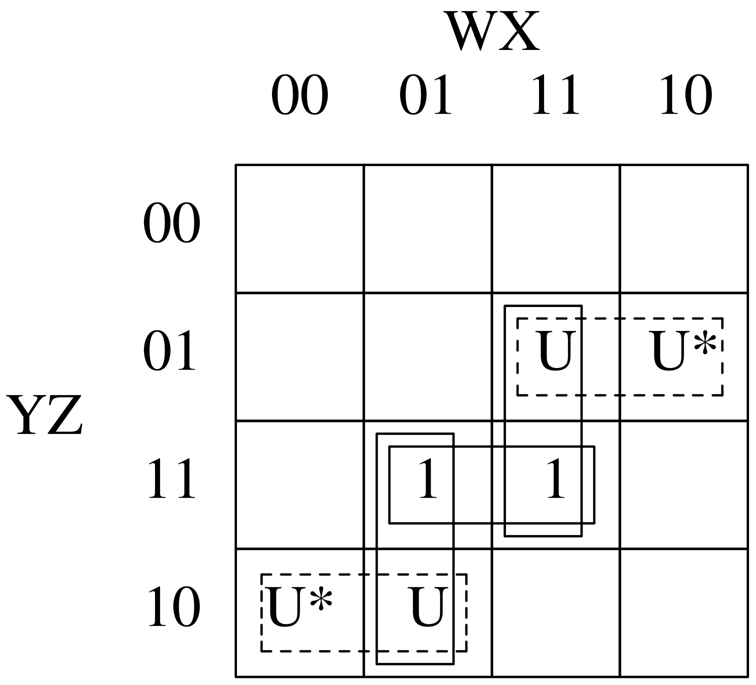
The following rule can help in the selection:
If *P* and *Q* are prime implicants, the cost of *P* is not higher than the cost of *Q*, and *P* covers all the remaining minterms that *Q* covers, remove *Q* from consideration.

After removing certain prime implicants, some of the remaining ones may be necessary to select.
These are referred to as *secondary essential prime implicants*.

Example:

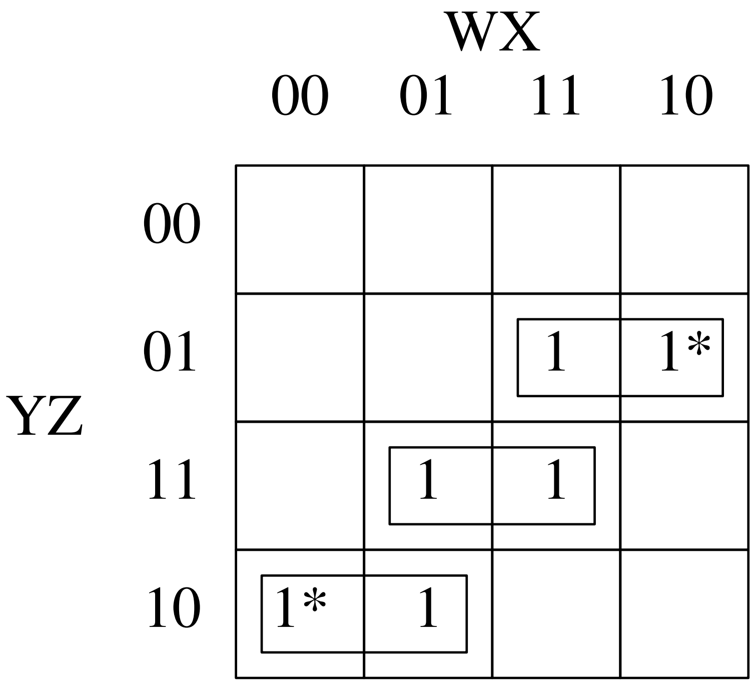


After selecting the essential prime implicants:
Two minterms remain uncovered.
We have three prime implicants to select from.



$X \cdot Y \cdot Z$ covers all the uncovered minterms covered by $W' \cdot X \cdot Y$. The latter can be removed.
$X \cdot Y \cdot Z$ covers all the uncovered minterms covered by $W \cdot X \cdot Z$. The latter can be removed.

Select:



The *prime implicant table* can be used to facilitate the selection of prime implicants for a minimal sum.

The prime implicant table contains:
A row for every prime implicant.
A column for every minterm.
An x in row i, column j if prime implicant i covers minterm j.

It helps to use a binary notation for the prime implicants. In the previous example:

$W \cdot Y' \cdot Z$	1-01
$X \cdot Y \cdot Z$	-111
$W' \cdot Y \cdot Z'$	0-10
$W' \cdot X \cdot Y$	011-
$W \cdot X \cdot Z$	11-1

The prime implicant table:

	0010	0110	0111	1001	1101	1111
1-01				x	x	
-111			x			x
0-10	x	x				
011-		x	x			
11-1					x	x

Essential prime implicants are the only ones to cover a minterm.
They are identified by a column with a single x.
0-10 is the only one to cover 0010.
1-01 is the only one to cover 1001.
Select them, and mark the covered minterms.
Remove:
The corresponding rows.
The columns of the minterms that are already covered.

	0111	1111
-111	x	x
011-	x	
11-1		x

The three prime implicants left have the same cost.
-111 covers all the minterms covered by 011-. Remove 011-.
-111 covers all the minterms covered by 11-1. Remove 11-1.

Select -111 to cover the remaining minterms.

We selected 0-10, 1-01 and -111.

A minimal sum:
$W' \cdot Y \cdot Z' + W \cdot Y' \cdot Z + X \cdot Y \cdot Z.$

Example: The following is the prime implicant table for the function $F(V, W, X, Y, Z) =$ $\sum(0, 1, 3, 4, 7, 13, 15, 19, 20, 22, 23, 29, 31)$.
Prime implicants 1-3 have three literals.
Prime implicants 4-9 have four literals.

	0	1	3	4	7	13	15	19	20	22	23	29	31
1						x	x					x	x
2					x		x				x		x
3			x		x			x			x		
4										x	x		
5									x	x			
6				x					x				
7		x	x										
8	x			x									
9	x	x											

Prime implicants 1 and 3 are essential prime implicants.

After selecting them and removing the minterms already covered we obtain the following table.

	0	1	4	20	22
2					
4					x
5				x	x
6			x	x	
7		x			
8	x		x		
9	x	x			

There is no need to select PI 2. It can be removed.

PI 5 covers every minterm that PI 4 covers, and more.
Therefore, any minimal sum that includes 4 can be replaced by a minimal sum that contains 5.
PI 4 can be removed.

	0	1	4	20	22
5				x	x
6			x	x	
7		x			
8	x		x		
9	x	x			

PI 9 covers every minterm that PI 7 covers, and more.
Therefore, PI 7 can be removed.

	0	1	4	20	22
5				x	x
6			x	x	
8	x		x		
9	x	x			

PIs 5 and 9 must now be selected (they are now essential, since we removed some PIs).
PIs 5 and 9 cover minterms 0, 1, 20 and 22.
These minterms can now be removed, in addition to removing PIs 5 and 9.

	4
6	x
8	x

Either PI 6 or PI 8 can be selected to cover the remaining minterm.

It is possible to find minimal products by considering the 0s of a K-map and applying the principle of duality.
The process is similar to the process for minimal sums.

An alternative is to consider F'.

Example: $F = \Pi_{W,X,Y,Z}(0, 2, 6, 7, 8, 10)$

	WX				
		00	01	11	10
	00	0			0
YZ	01				
	11		0		
	10	0	0		0

Consider $F' = \Sigma_{W,X,Y,Z}(0, 2, 6, 7, 8, 10)$

	WX			
YZ	00	01	11	10
00	1			1
01				
11		1		
10	1	1		1

$F' = X' \cdot Z' + W' \cdot X \cdot Y.$

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

A function may be incompletely-specified. In this case, some of its output values are $don't - cares$.

The function will be specified as $\sum(\cdots) + \sum_d(\cdots)$.

To minimize such a function it is possible to use the same process as for a fully-specified function, with the following changes:
Prime implicants can cover don't-cares, but they should not cover only don't-cares.
Don't-cares do not need to be covered when selecting a minimal sum.

In the prime implicant table:
Include a row for every prime implicant.
Do not include columns for minterms where the function is a don't-care.

A function may have multiple outputs.
Minimizing each output separately may not lead to a minimal circuit.
There are procedures for multi-output minimization that address this.
For functions f_1, f_2, f_3 they consider the prime implicants of f_1, f_2, f_3, $f_1 \cdot f_2$, $f_1 \cdot f_3$, $f_2 \cdot f_3$, $f_1 \cdot f_2 \cdot f_3$.

Instead of using a K-map, the Quine-McCluskey method finds all the prime implicants of a function by combining pairs of minterms, and then pairs of product terms with decreasing numbers of literals.
It is applicable to functions with large numbers of variables.

$F(W, X, Y, Z) = \sum(2, 6, 7, 9, 13, 15).$

$$\begin{array}{c} 0010 \\ \hline 0110 \\ 1001 \\ \hline 0111 \\ 1101 \\ \hline 1111 \end{array}$$

$$\begin{array}{c} 0\text{-}10 \\ \hline 011\text{-} \\ 1\text{-}01 \\ \hline \text{-}111 \\ 11\text{-}1 \end{array}$$

Obtaining a product-of-sums expression for a function given as a sum-of-products expression (or vice versa):

Suppose that F is given as a sum-of-products expression.

Compute F' as a product-of-sums expression using DeMorgan’s theorem.

Apply the distributive law to obtain F' as a sum-of-products expression.

Apply DeMorgan’s theorem again to obtain F as a product-of-sums expression.

Example: $F(X, Y, Z) = X \cdot Y + X \cdot Z$
We expect the result $F(X, Y, Z) = X \cdot (Y + Z)$.

Applying the algorithm above:

$$\begin{aligned}
F(X, Y, Z) &= X \cdot Y + X \cdot Z \\
F'(X, Y, Z) &= (X' + Y') \cdot (X' + Z') \\
&= X' + Y' \cdot Z' \\
F(X, Y, Z) &= X \cdot (Y + Z)
\end{aligned}$$

Starting from a minimal sum the result is not guaranteed to be a minimal product.

Example of a function for which the minterms are essential prime implicants:
Even parity function.
$n = 3$:

X_3 \ X_1X_2	00	01	11	10
0	1	0	1	0
1	0	1	0	1

$n = 4$:

X_3X_4 \ X_1X_2	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

Example of minimization of a function with don't-cares.

Find a minimal sum for the function

$$F(V, W, X, Y, Z) = \sum(8, 9, 10, 11, 14, 30) + \sum_d(7, 15, 23, 31)$$

V	W	X	Y	Z
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	1	0
1	1	1	1	1

V	W	X	Y	Z
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	1	0
1	1	1	1	1

V	W	X	Y	Z
0	1	0	0	-
0	1	0	-	0
0	1	0	-	1
0	1	0	1	-
0	1	-	1	0
0	-	1	1	1
-	0	1	1	1
0	1	-	1	1
0	1	1	1	-
-	1	1	1	0
-	1	1	1	1
1	-	1	1	1
1	1	1	1	-

V	W	X	Y	Z
0	1	0	-	-
0	1	-	1	-
-	-	1	1	1
-	1	1	1	-

Prime implicant table:

$$F(V, W, X, Y, Z) = \sum(8, 9, 10, 11, 14, 30) + \sum_d(7, 15, 23, 31)$$

The prime implicant table does not include the minterms where the function is a don't-care since these minterms do not need to be covered.

	8	9	10	11	14	30
010--	x	x	x	x		
01-1-			x	x	x	
--111						
-111-					x	x

Select the essential prime implicants
010-- and -111-.

$F = V' \cdot W \cdot X' + W \cdot X \cdot Y.$