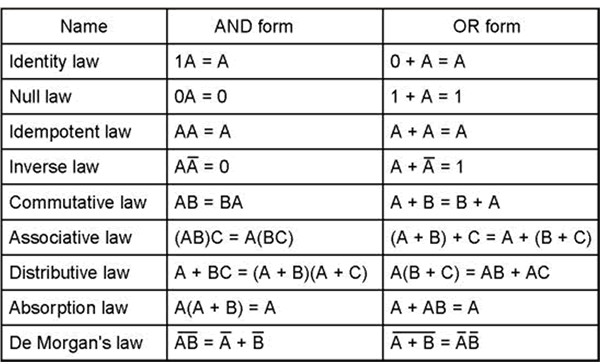
**Switching Theory & Logic Design**

**UNIT-II**

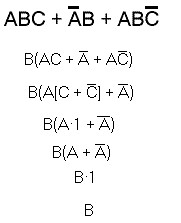
**COMBINATIONAL LOGIC CIRCUITS**

**Boolean Algebra Laws:**



**Boolean expression reduction:**

**Example 1:**



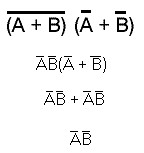
**Example 2:**

A’BC+AC=C(A’B+A)

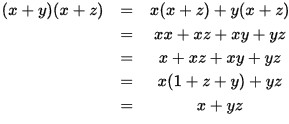
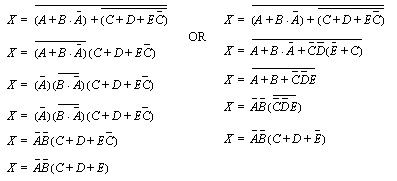
C(A+B)

**Example 3:**

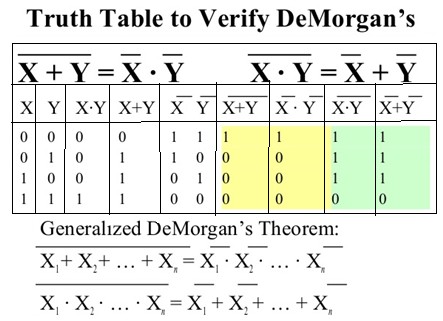
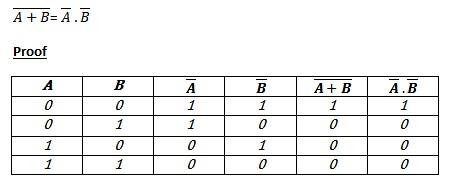
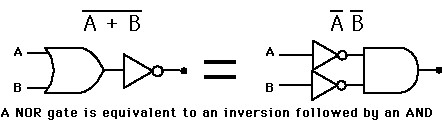
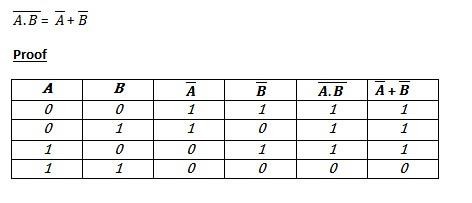
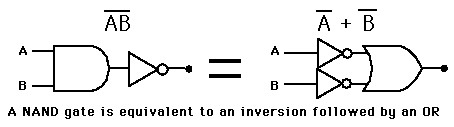
**Example 4:**



**Example 5:**



**Demor gan’ s Theorem laws:**



**Boolean Logic - SOP and POS forms**

The SOP form takes on the appearance of being a sum of several terms, each of which is the product of several factors while the POS form takes on the appearance of being a product of several factors, each of which is the sum of several terms.

For Ex:

SOP form: F= + +



POS form: F= ( )( ) ( )



The first of these, examples, is the Standard Sum of Products, or Standard SOP, form while the second, example, is the Standard Product of Sums, or Standard POS, form.The term "standard" here means that the expression consists exclusively of minterms (in the case of Standard SOP) or maxterms (in the case of Standard POS).

A "product" in Boolean algebra is a logical AND operation while a "sum" is a logical

OR operation.

**Minterm**

A "minterm" is a Boolean expression that is True for the minimum number of combinations of inputs; this minimum number is exactly one .To minimize the coverage, we want to use the Boolean operation that is the most restrictive, which is the AND operation. Since the AND is true only if ALL of the inputs are True, we can craft an expression that is True for only a single combination of inputs by including each input in the product. If the input is uncomplemented, then we require that it be True, while if it is complemented, then we require it to be False. The minterm expression for each combination of inputs is therefore shown in the below table:

**TABLE-I**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Decimal** | **A** | **B** | **C** | **minterm** |
| 0 | 0 | 0 | 0 |  |
| 1 | 0 | 0 | 1 |  |
| 2 | 0 | 1 | 0 |  |
| 3 | 0 | 1 | 1 |  |
| 4 | 1 | 0 | 0 |  |
| 5 | 1 | 0 | 1 |  |
| 6 | 1 | 1 | 0 |  |
| 7 | 1 | 1 | 1 |  |

**Maxterm**

A "maxterm" is a Boolean expression that is True for the maximun number of combinations of inputs; this maximum number is exactly one fewer than the total number of possibilities (the case of a term being True for all combinations is, of course, simply a hard True and is trivial and uninteresting).

Because we want to maximize the coverage, we want to use the Boolean operation that is the most permissive, which is the OR operation. While the OR is True if ANY of its inputs is True, for our purposes it is more convenient to recognize that the OR is False only if ALL of its inputs are False; thus, we can craft an expression that is False for only a single combination of inputs by including each input in the sum. If the input is uncomplemented, then we require that it be False, while if it is complemented, then we require it to be True.

**TABLE-II**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Decimal** | **A** | **B** | **C** | **maxterm** |
| 0 | 0 | 0 | 0 |  |
| 1 | 0 | 0 | 1 |  |
| 2 | 0 | 1 | 0 |  |
| 3 | 0 | 1 | 1 |  |
| 4 | 1 | 0 | 0 |  |
| 5 | 1 | 0 | 1 |  |
| 6 | 1 | 1 | 0 |  |
| 7 | 1 | 1 | 1 |  |

**Standard SOP (Sum of Products)**

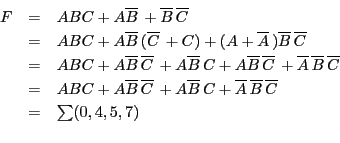
The SOP form takes on the appearance of being a sum of several terms, each of which is the product of several factors. To craft the SOP form of a Boolean logic function, we merely need to OR together the minterms associated with each combination of inputs for which the overall output should be True.

By looking at Table-I we see that we need to sum the minterms associated with rows {1,3,4,6,7}. This is often represented simply as

𝐹 = ∑(1,3,4,6,7)

By expanding the summation and replacing each label with the corresponding minterm, we immediately obtain the canonical disjunctive form. The term "canonical" means "standardized" while the term "disjunctive" means a logical union, which is the same thing as a logical ORing of sets. This form is more commonly known, particularly among the more application-oriented, simply as the Standard SOP form.

**Example 1:**



**Standard POS (Product of Sums)**

The POS form takes on the appearance of being a product of several factors, each of which is the sum of several terms.To craft the POS form of a Boolean logic function, we need to AND together the maxterms associated with each combination of inputs for which the overall output should be False. If we AND together several factors, the output will be False as long as any one of the factors is False. A maxterm is False for exactly one combination of inputs, so using a maxterm for a combination for which we want the overall output to be False will achieve this goal and, since that maxterm is True for all other combinations, it will have no effect on the output for them. As long as we do not include the maxterms for any combinations that we do want the output to be True for, we are guaranteed that all of the other maxterms in the expression will be True and, hence, the overall product of them will be True.

By looking at Table-II we see that we need to take the product of the maxterms associated with rows {0',2',5'}. This is often represented simply as

𝐹 = ��(0’,2’,5’)

By expanding the product and replacing each label with the corresponding maxterm, we

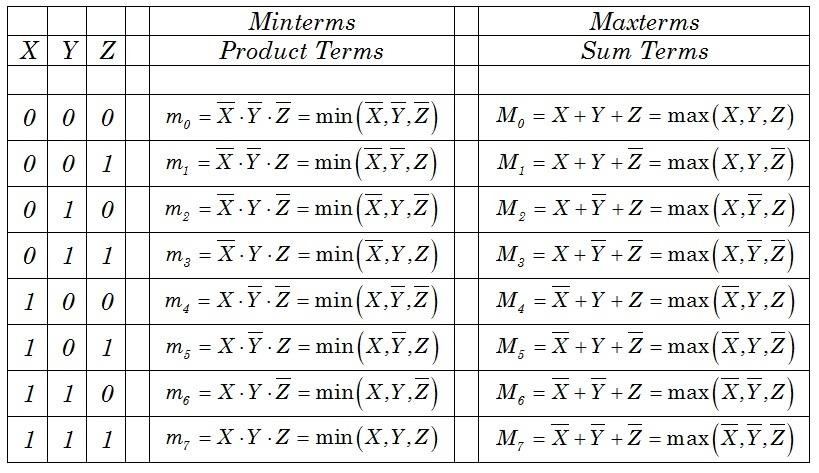
immediately obtain the canonical conjunctive form.The term "canonical" just means

"standardized" while the term "conjunctive" merely means a logical intersection, which is the

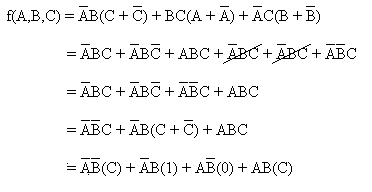
same thing as a logical ANDing of sets. This form is more commonly known, particularly among the more application-oriented, simply as the Standard POS form.

**Converting Boolean Expressions into SOP/POS Form**

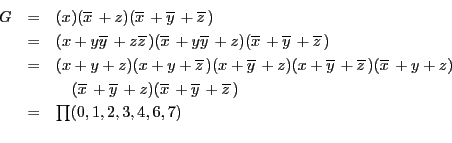
The process of converting any Boolean expression into either POS or SOP form (canonical or otherwise) is very straightforward.To get the expression in SOP form, you simply distribute all AND operations over any OR operations and continue doing this as long as possible. When finished, you will have an expression in SOP form. If you want it in canonical form, then you simply expand each term as necessary.To get the expression in POS form, you simply distribute all OR operations over any AND operations and continue doing this as long as possible. When finished, you will have an expression in POS form. If you want it in canonical form, then you simply exapand each term as necessary.Most people have little problem getting an arbitrary expression into SOP form because the notation used for AND and OR are the same used for multiplication and addition in normal arithmetic and the notion of distributing multiplication over addition has long become internalized. But the fact that addition is not distributable over multiplication has also been internalized and hence doing something that looks the same does not come naturally. But just as AND can be distributed over OR, so too can OR be distributed over AND.



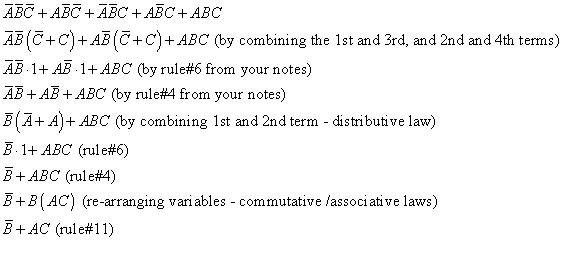
**Example 1:**



**Example 2:**



**Example 3:**



**Karnaugh Maps**

The Karnaugh map, also known as the K-map, is a method to simplify Boolean algebra expressions. [Maurice Karnaugh in](https://en.wikipedia.org/wiki/Maurice_Karnaugh)troduced it in 1953 as a refinement of [Edward Veitch's](https://en.wikipedia.org/wiki/Edward_Veitch)

1952 Veitch diagram. The Karnaugh map reduces the need for extensive calculations by taking

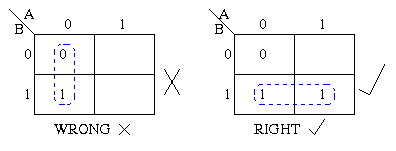
advantage of humans' pattern-recognition capability. It also permits the rapid identification and

elimination of potential [race conditions.The](https://en.wikipedia.org/wiki/Race_condition) required boolean results are transferred from [a truth table onto](https://en.wikipedia.org/wiki/Truth_table) a two-dimensional grid where the cells are ordered in [Gray code,](https://en.wikipedia.org/wiki/Gray_code) and each cell position represents one combination of input conditions, while each cell value represents the corresponding output value. Optimal groups of 1s or 0s are identified, which represent the terms of a [canonical form of](https://en.wikipedia.org/wiki/Canonical_form_(Boolean_algebra)) the logic in the original truth table.These terms can be used to write a minimal boolean expression representing the required logic.Karnaugh maps are used to simplify real-world logic requirements so that they can be implemented using a minimum number of physical logic gates. A [sum-of-products expression c](https://en.wikipedia.org/wiki/Conjunctive_normal_form)an always be implemented using [AND gates](https://en.wikipedia.org/wiki/AND_gate) feeding into an [OR gate, a](https://en.wikipedia.org/wiki/OR_gate)nd [a product-of-sums expression le](https://en.wikipedia.org/wiki/Disjunctive_normal_form)ads to OR gates feeding an AND gate. Karnaugh maps can also be used to simplify logic expressions in software design. Boolean conditions, as used for example in [conditional statements,](https://en.wikipedia.org/wiki/Conditional_(programming)) can get very complicated, which makes the code difficult to read and to maintain. Once minimised, canonical sum-of- products and product-of-sums expressions can be implemented directly using AND and OR logic operators.

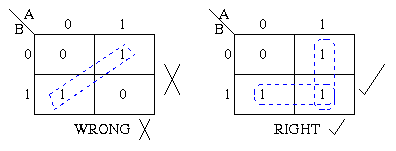
**K- Maps cell numbering:**

**Rules of Simplification of Karnaugh Maps**

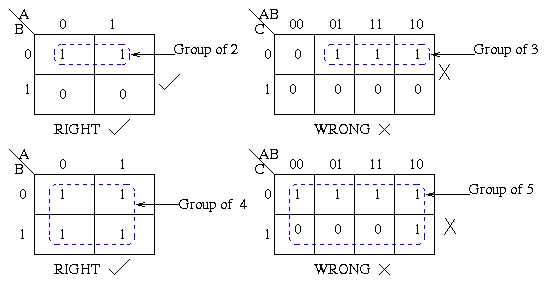
**Groups may not include any cell containing a zero**



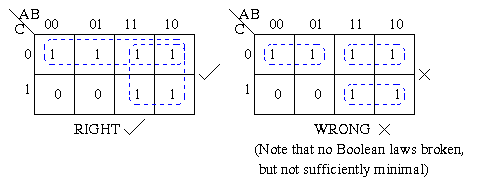
**Groups may be horizontal or vertical, but not diagonal.**



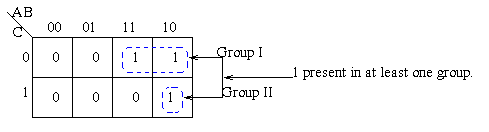
**Groups must contain 1, 2, 4, 8, or in general 2n cells. That is if n = 1, a group will contain two 1's since 21 = 2. If n = 2, a group will contain four 1's since 22 = 4.**



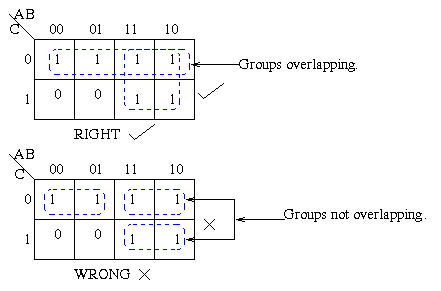
**Each group should be as large as possible.**



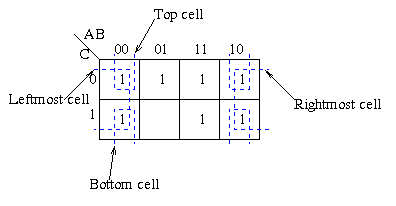
**Each cell containing a *one* must be in at least one group**



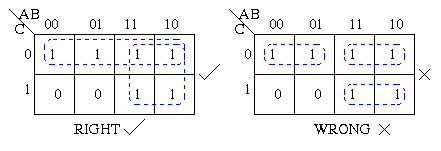
**Groups may overlap.**



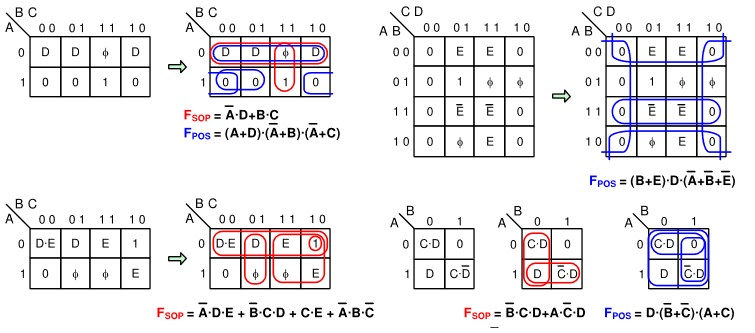
**Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**



**There should be as few groups as possible, as long as this does not contradict any of the previous rules.**

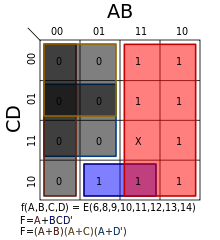


**2,3,4-variable map examples:**



**Don't cares**

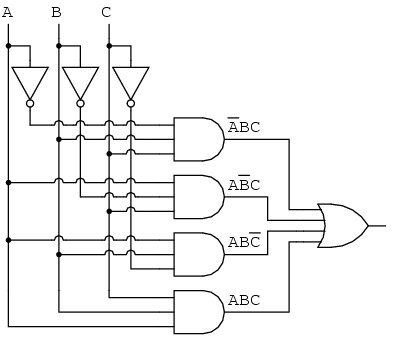
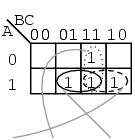
The value of f(A,B,C,D) for ABCD = 1111 is replaced by a "don't care". This removes the green term completely and allows the red term to be larger. It also allows blue inverse term to shift and become larger



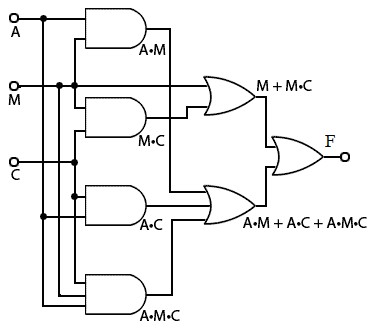
Karnaugh maps also allow easy minimizations of functions whose truth tables include "[don't care"](https://en.wikipedia.org/wiki/Don%27t-care_(logic)) conditions. A "don't care" condition is a combination of inputs for which the designer doesn't care what the output is. Therefore, "don't care" conditions can either be included in or excluded from any rectangular group, whichever makes it larger. They are usually indicated on the map with a dash or X.

Logical simplification using K-map:

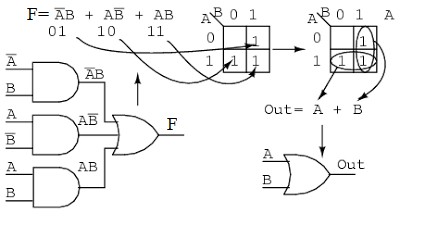
**Example 1:**



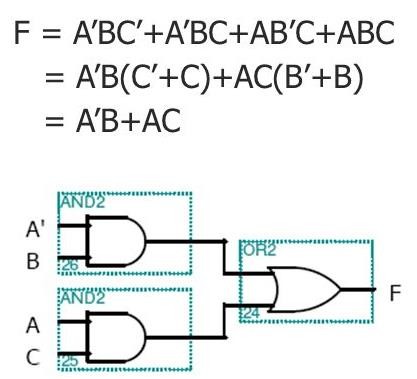
**Example 2:**



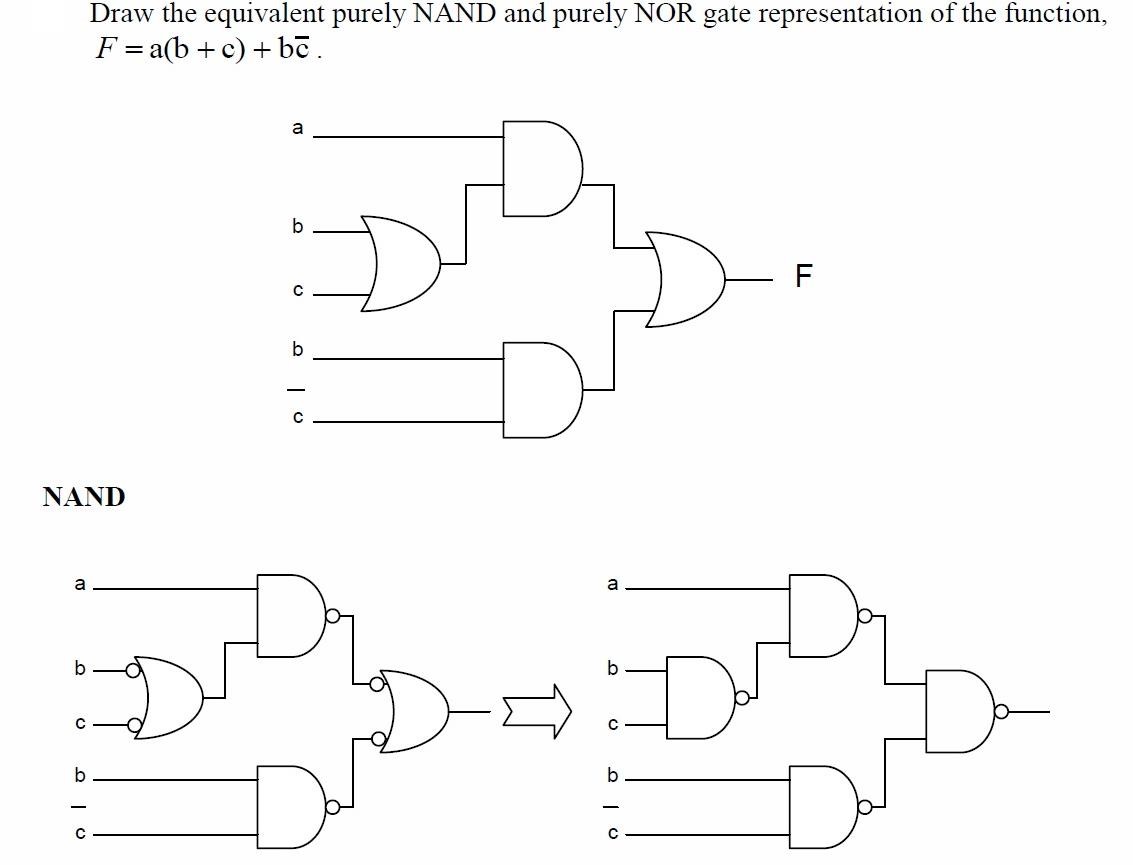
**Example 2:**



Example 3:



Example 5:



**Implicants,Prime Implicants and essential Prime**

**Implicant:**

**Implicants:**

A product term that could be used to cover one or more minterms

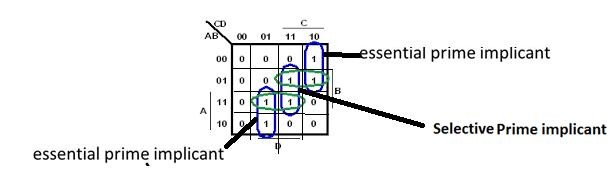
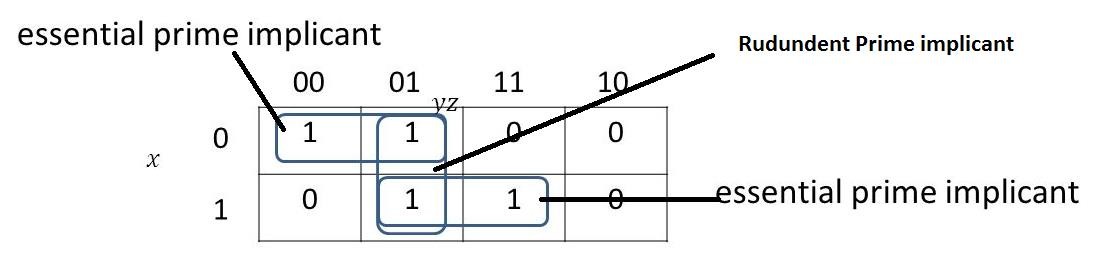
**PRIME IMPLICANTS:**

A product term obtained by combining the maximum number of adjacent squares in

the map

**ESSENTIAL PRIME IMPLICANTS:**

A prime implicant that covers at least one minterm that is not covered by any other prime implicant



**Exmple 1:**

Show prime implicants in the below shown k-map.

