CMPT-150

Introduction to Computer Design

SFU, Harbour Centre, Spring 2007

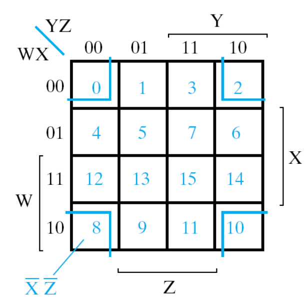
Lecture 5: Jan. 23, 2007

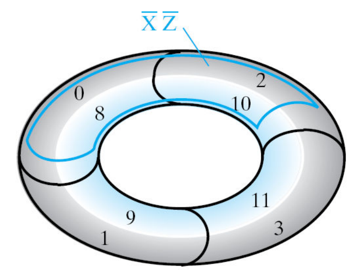
- Minimization of Boolean Functions Using K-Maps
  - 4-variables K-Maps
  - Prime and Essential Implicants
  - Optimization of Product of Sums
  - Don't-cares

Four Variables K-Map

Can have rectangles corresponding to:

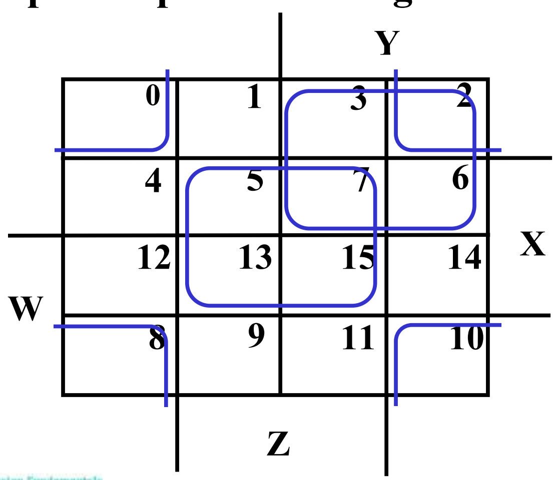
- A single 1 = 4 variables, (i.e. Minterm)
- Two 1s = 3 variables,
- Four 1s = 2 variables
- Eight 1s = 1 variable,
- Sixteen 1s = zero variables (i.e. Constant "1")





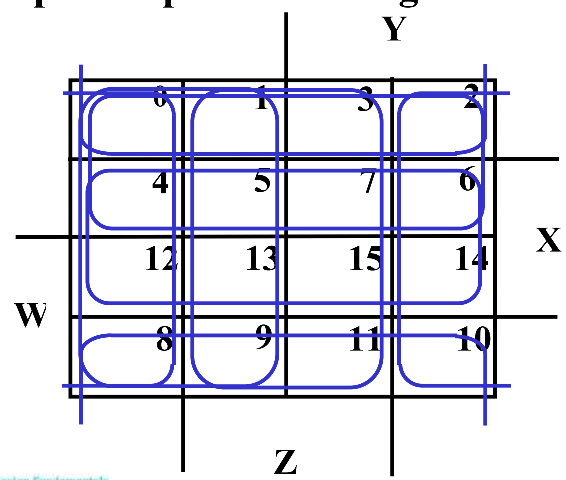
Four-Variable Maps

- **Example Shapes of Rectangles:**



Four-Variable Maps

- **Example Shapes of Rectangles:**

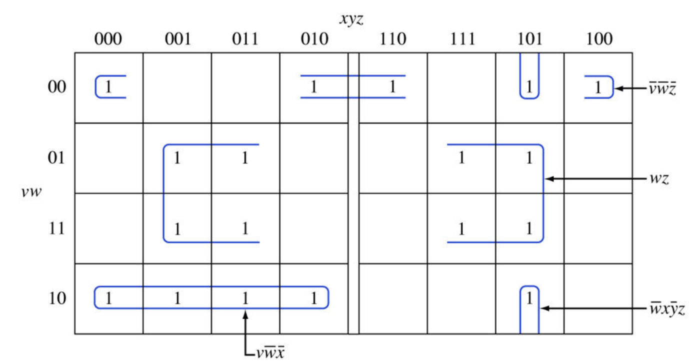


Four-Variable Map Simplification

- $F(W,X,Y,Z) = \Sigma_m(0, 2,4,5,6,7,8,10,13,15)$

Five Variable or More K-Maps

- For five variable problems, we use two adjacent K-maps. It becomes harder to visualize adjacent minterms. You can extend the problem to six variables by using four K-Maps.

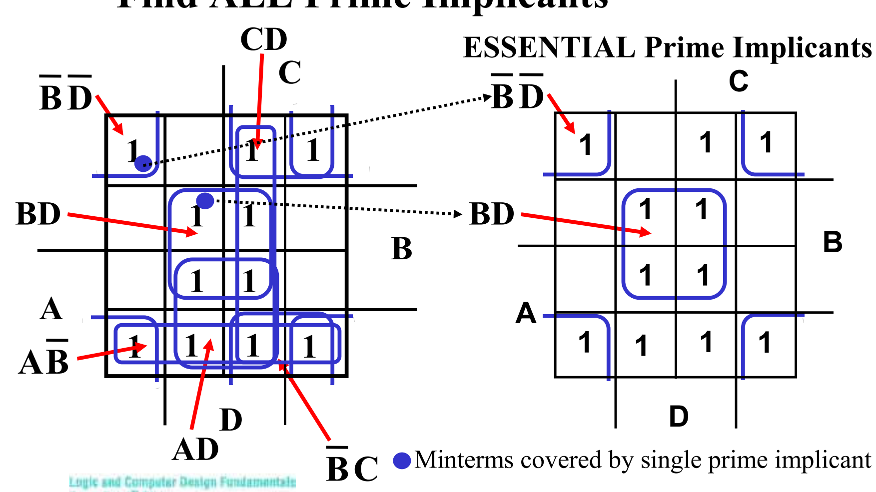


Systematic Simplification

- A *Prime Implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2.
- A prime implicant is called an *Essential Prime Implicant* if it is the only prime implicant that covers (includes) one or more minterms.
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map.
- A set of prime implicants *"covers all minterms"* if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm.

Example of Prime Implicants

- Find ALL Prime Implicants



Optimization Algorithm

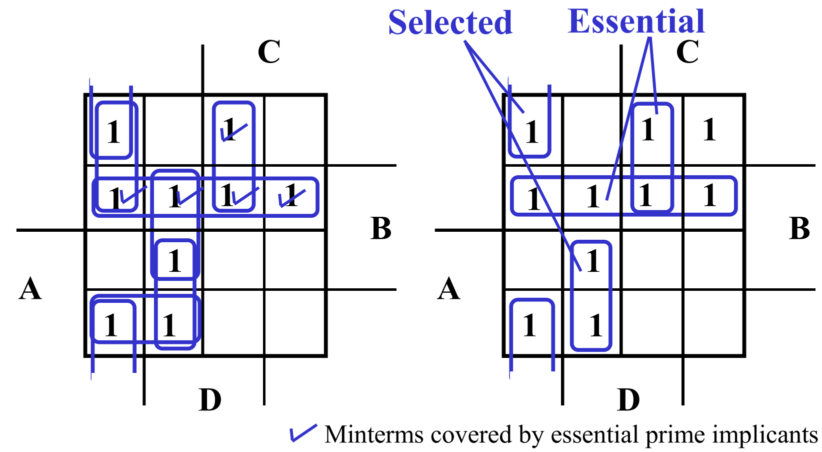
1. Find all prime implicants.
2. Include all essential prime implicants in the solution
3. Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered.

Notes regarding Step 3:

- Always prefer:
  - Small number of implicants (=rectangles)
  - Implicants with small number of literals (=large rectangles)
- This step is not well-defined, however, for the small examples we consider, you should be able to find an optimal or near-optimal solution.

Example

- Simplify F(A, B, C, D) given on the K-map.



Product of Sums Minimization

- We can minimize the POS form of a function f using K-Maps:
  - Cover the maxterms (O's) in the K-Map.

  Or:
  - Find minimized SOP for the complement of f, $\overline{f}$
  - Get a POS representation of f by taking the dual of the SOP representation of $\overline{f}$ and complementing each literal.

- Example: Simplify $F(A,B,C,D)=\sum_m(0,1,2,5,8,9,10)$ in POS form.

Don't Cares in K-Maps

- Sometimes a function table or map contains entries for which it is known:
  - the input values for the minterm will never occur, or
  - The output value for the minterm is not used
- In these cases, the output value need not be defined
- Instead, the output value is defined as a "don't care"
- By placing "don't cares" (an "x" entry) in the function table or map, the cost of the logic circuit may be lowered.

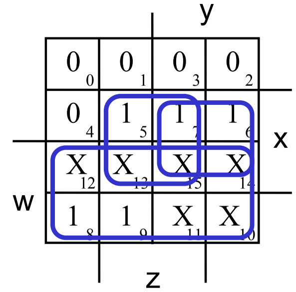
Don't Cares in K-Maps

- Example 2: A circuit that represents a very common situation that occurs in computer design has two distinct sets of input variables:
  - A, B, and C which take on all possible combinations, and
  - Y which takes on values 0 or 1.

  and a single output Z. The circuit that receives the output Z observes it only for (A,B,C) = (1,1,1) and otherwise ignores it. Thus, Z is specified only for the combinations (A,B,C,Y) = 1110 and 1111. For these two combinations, Z = Y. For all of the 14 remaining input combinations, Z is a don't care.
- Ultimately, each "x" entry may take on either a 0 or 1 value in resulting solutions
- For example, an "x" may take on value "0" in an SOP solution and value "1" in a POS solution, or vice-versa.
- Any minterm with value "x" need not be covered by a prime implicant.

Example: BCD "5 or More"

- The map below gives a function F1(w,x,y,z) which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:



F1 (w,x,y,z) = w + x z + x y G = 7

- This is much lower in cost than F2 where the "don't cares" were treated as "0s."

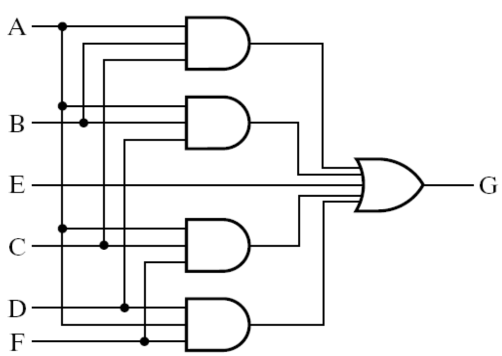
$F_2(w,x,y,z) = \overline{w}\,x\,z + \overline{w}\,x\,y + w\overline{x}\,\overline{y}$ G = 12

- For this particular function, cost G for the POS solution for F_1(w,x,y,z) is not changed by using the don't cares.

2-Levels Optimization is not Always Optimal

- Consider the function $G = ABC + ABD + E + ACF + ADF$

  Implemented according to this expression its gate input cost is 17



2-Levels Optimization is not Always Optimal (cont')

- Using Boolean Algebra we can simplify G:

$$G = ABC + ABD + E + ACF + ADF$$
$$= AB(C+D) + E + AF(C+D)$$
$$= (AB + AF)(C+D) + E$$
$$= A(B+F)(C+D) + E$$

➔ Results in a 3-levels circuit with gate input cost 8 !