CMPE 212
Principles of Digital Design

Lecture 11

Karnaugh Maps

February 29, 2016

www.csee.umbc.edu/~younis/CMPE212/CMPE212.htm

Lecture's Overview

- ***Previous Lecture:***
  - ➔ Reduction of combinational logic
    (Types, Goals, Methodology, Fundamental Concept)
  - ➔ The Karnaugh map method
    (Basic concept, Implicants, prime implicants)
- ***This Lecture***
  - ➔ Extended K-map procedure
  - ➔ Optimization of incompletely specified functions
  - ➔ Scalability limitation of the K-map method
  - ➔ Other circuit performance considerations

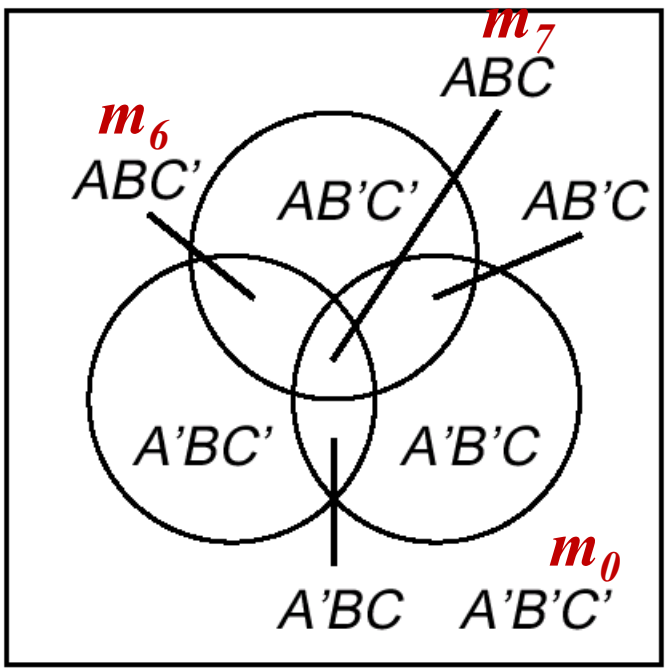
Design Optimization

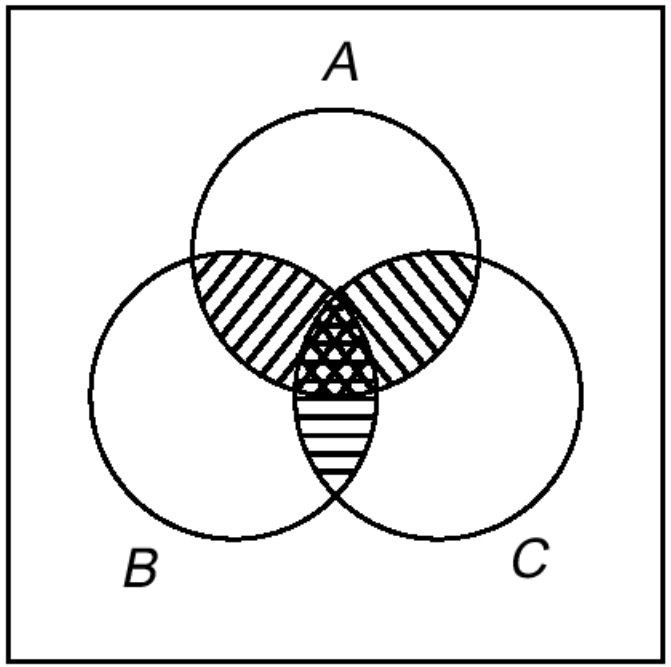
- Quality of combinational circuit design is measured using following metrics:
  - *Gate counts*: fewer gates require smaller area and cost less
  - *Propagation delay*: time for the output to become available after applying input. This time depends on transistor-level gate implementation
  - *Gate fan-in*: large gate fan-in can lead to increased gate counts and propagation delay (by using multi-level of gates)
  - *Gate fan-out*: large gate fan-out may mandate logic replication
- In many cases the canonical sum-of-products or product-of-sums forms are not minimal in terms in their number and size
- Since a smaller Boolean equation translates to a lower gate input count in the target circuit, reduction of the equation is an important consideration when circuit complexity is an issue
- Three methods for reducing Boolean equations are considered:
  - Algebraic reduction
  - Karnaugh map (K-map) reduction
  - Tabular reduction (Quine-McCluskey)

Karnaugh Maps

- Karnaugh maps (K-map) is a graphical technique to visualize minterms along with variables that are common to them
- Variables common to multiple minterms are candidates for elimination
- The basis of K-maps is the Venn diagram representation for visualizing concepts in set theory
- Each distinct region in the "Universe" represents a minterm
- *Example:* Majority Function

F = BC + AC + AB





Forming K-Map

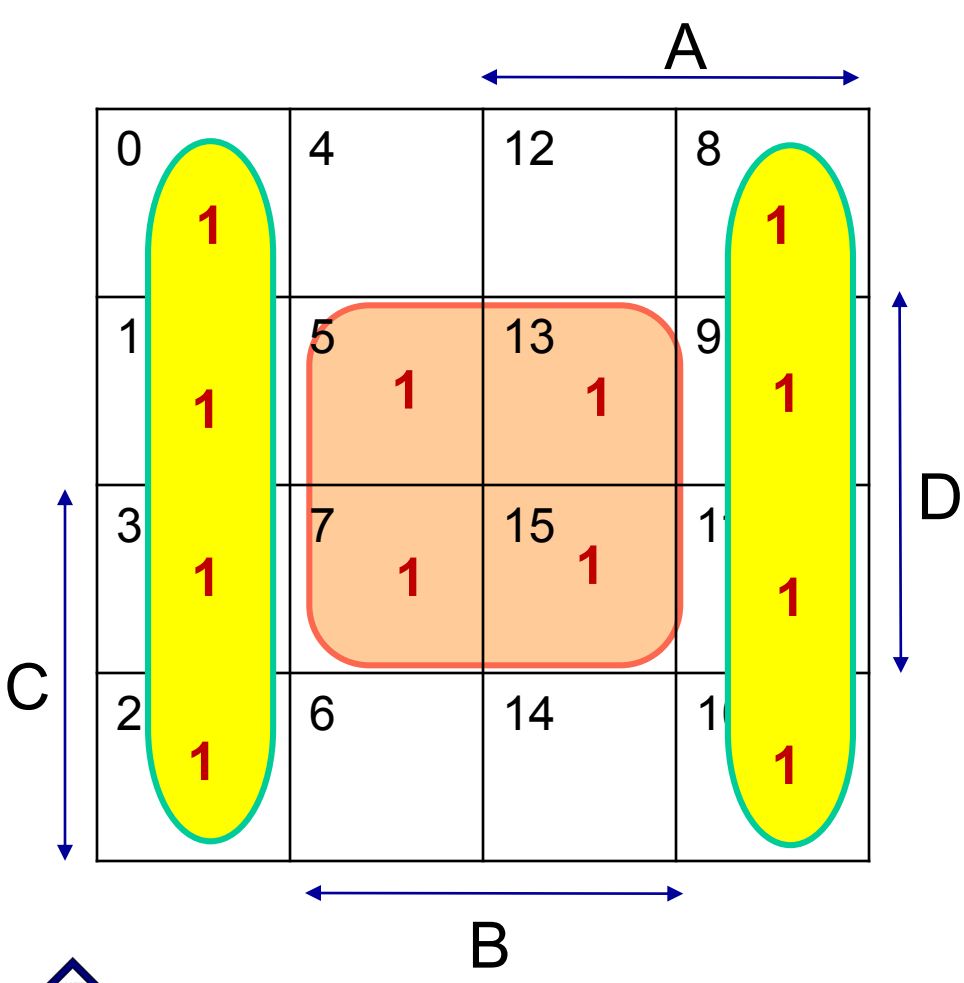
- Labeling along top and left side is arranged in a "Gray Code", in which exactly one variable changes between adjacent cells along each dimension
- Place a "1" in each cell that corresponds to that minterm (remaining cells should have a zero but omitted for clarity)
- Adjacent 1's satisfy the condition needed to apply complement property of Boolean algebra
- Grouping of adjacent cells are made into rectangles in sizes that corresponds to powers of 2 (called prime implicants)
- The larger the prime implicants gets, the bigger the number of eliminated variables becomes
- Cells on the outer edge of the map "wrap around"
- All ones has to be covered in at least one prime implicant (multiple coverage is actually encouraged if it leads to further simplification)

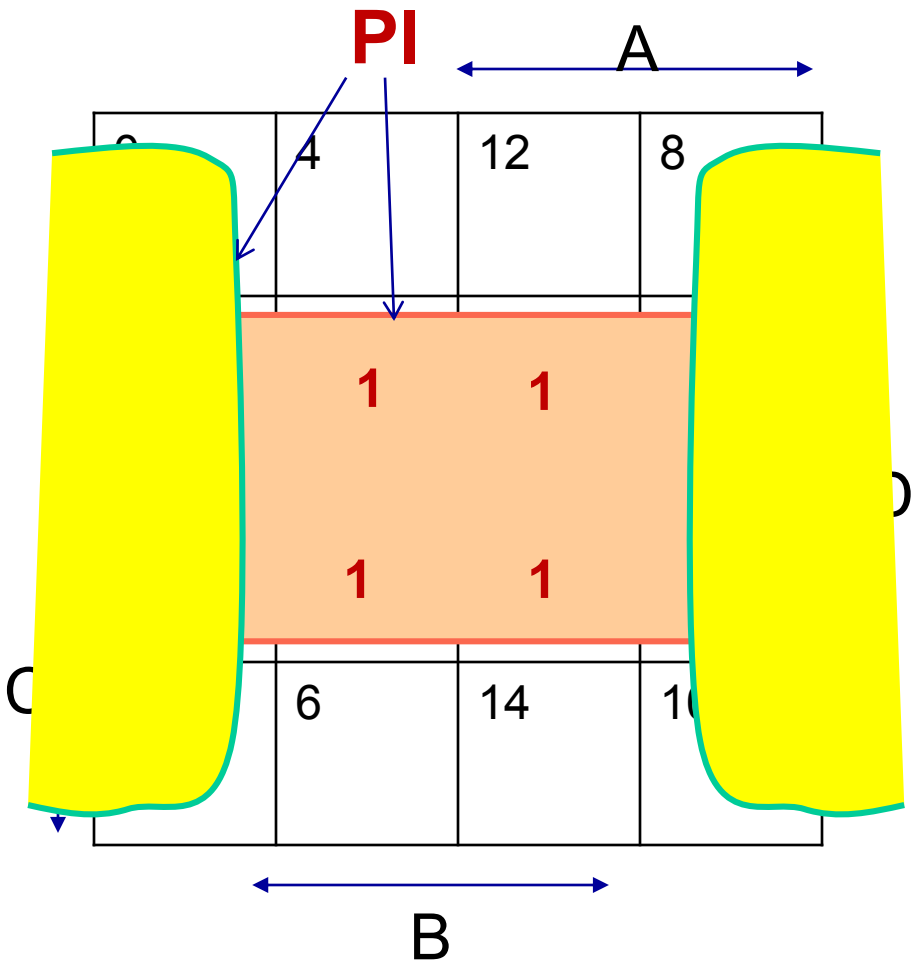
Minterm Index	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

C \ AB	00	01	11	10
0			1	
1		1	1	1

Prime Implicant (PI)

- A cube or implicant of a function that cannot grow larger by expanding into other cubes

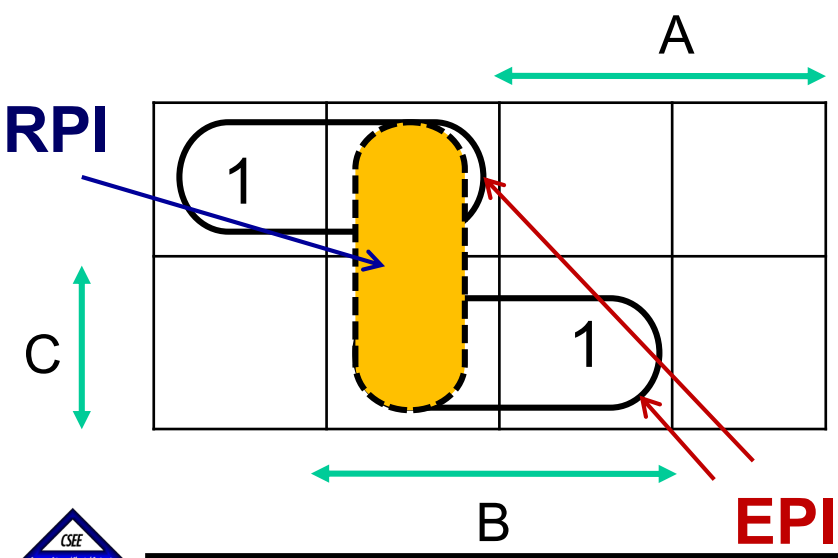


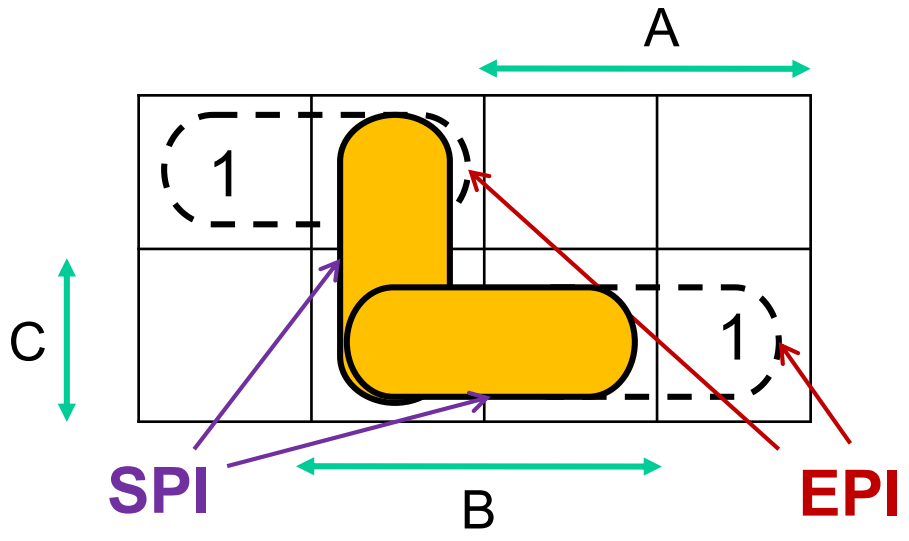


Slide contents are courtesy of Vishwani D. Agrawal

Type of Prime Implicants

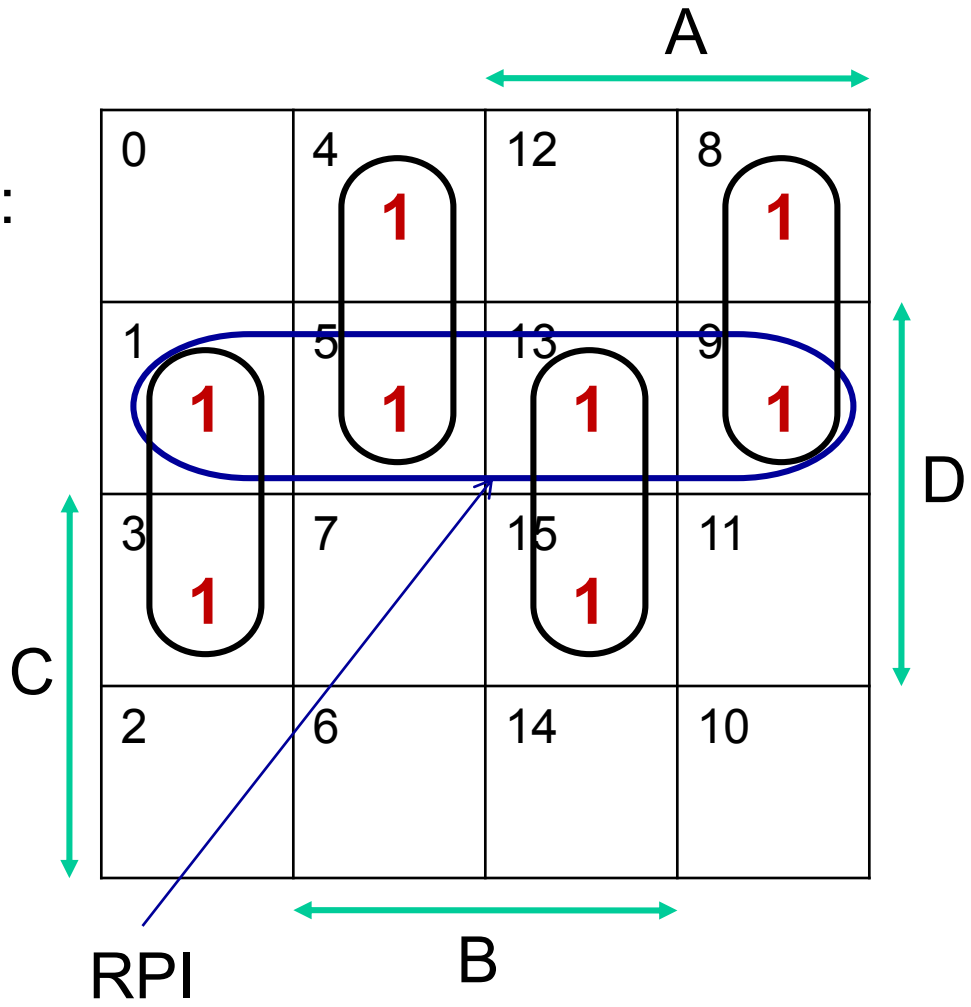
- If among the minterms subsuming a prime implicant (PI), there is at least one minterm that is covered by this and only this PI, then the PI is called an *essential* prime implicant (EPI)
- If each minterm subsuming a prime implicant (PI) is also covered by other essential prime implicants, then that PI is called a *redundant* prime implicant (RPI)
- A prime implicant (PI) that is neither EPI nor RPI is called a *selective* prime implicant (SPI); SPIs occur in pairs.





Minimum Sum of Products (MSOP)

- Identify all prime implicants (by letting implicants grow)
- Construct MSOP with PIs only:
  - Cover all minterms
  - Use only essential prime implicants (EPI)
  - Use no redundant prime implicant (RPI)
  - Use cheapest selective prime implicants (SPI), e.g. choose EPI in ascending order, starting from 0-implicant, then 1-implicant, 2-implicant, . . .

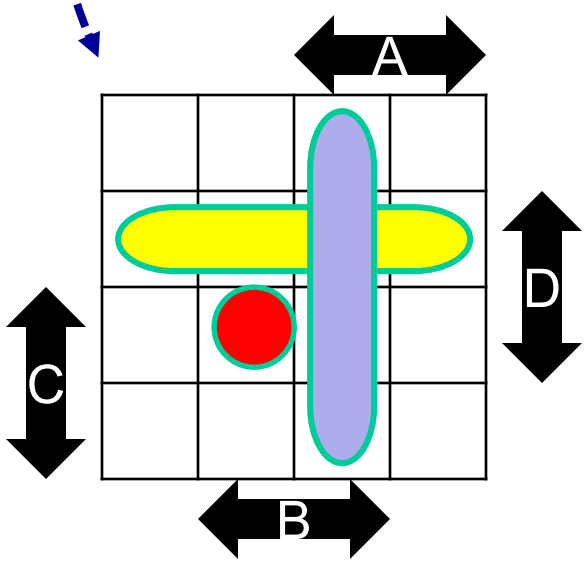


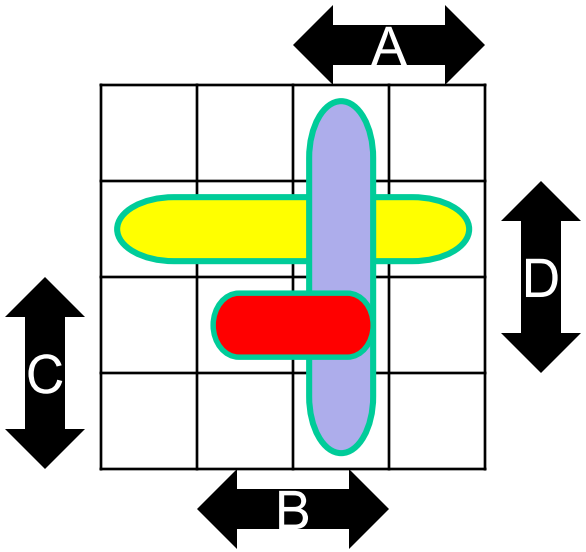
Example: $F=\sum m(1,3,4,5,8,9,13,15)$

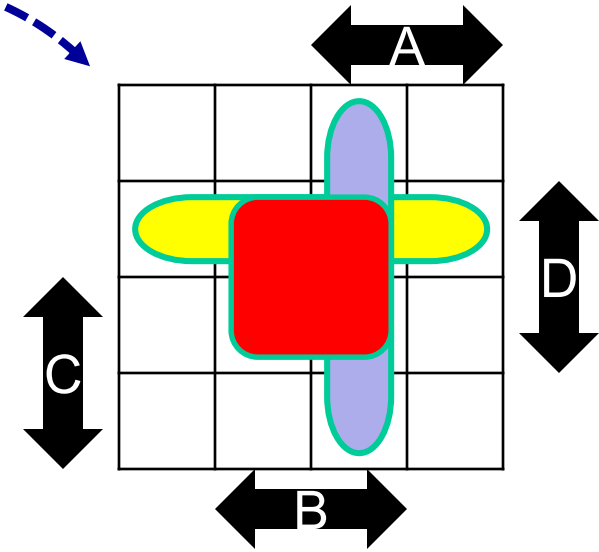
$$F = \bar{A}\,\bar{B}\,D + \bar{A}\,B\,\bar{C} + A\,B\,D + A\,\bar{B}\,\bar{C}$$

Growing Implicants to PI

- F = AB + $\bar{C}$D + $\bar{A}$BCD — *initial implicants*
  = AB + $\bar{A}$BCD + BCD + $\bar{C}$D — *consensus th.*
  = AB + BCD + $\bar{C}$D — *absorption th.*
  = AB + BCD + $\bar{C}$D + BD — *consensus th.*
  = AB + $\bar{C}$D + BD — *absorption th.*







Slide contents are courtesy of Vishwani D. Agrawal

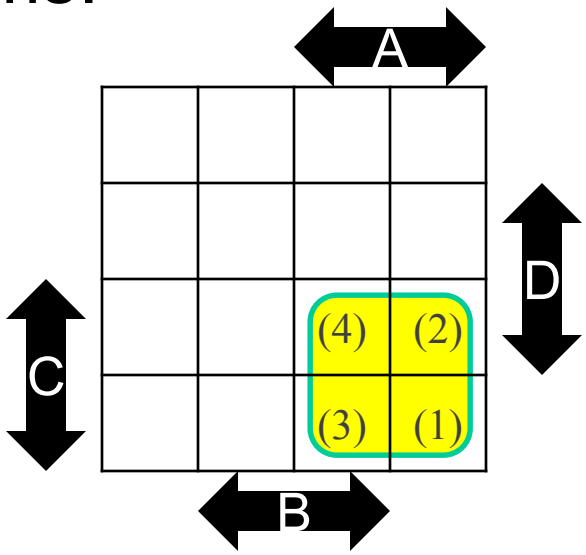
Minterms Covered by a Product

- A product from which k variables have been eliminated, covers 2^k minterms.
- Example: For four variables, A, B, C, D

  Product AC covers $2^2 = 4$ minterms:

  1) A $\bar{B}$ C $\bar{D}$
  2) A $\bar{B}$ C D
  3) A B C $\bar{D}$
  4) A B C D

Obtained by inserting the eliminated variables in all possible ways.



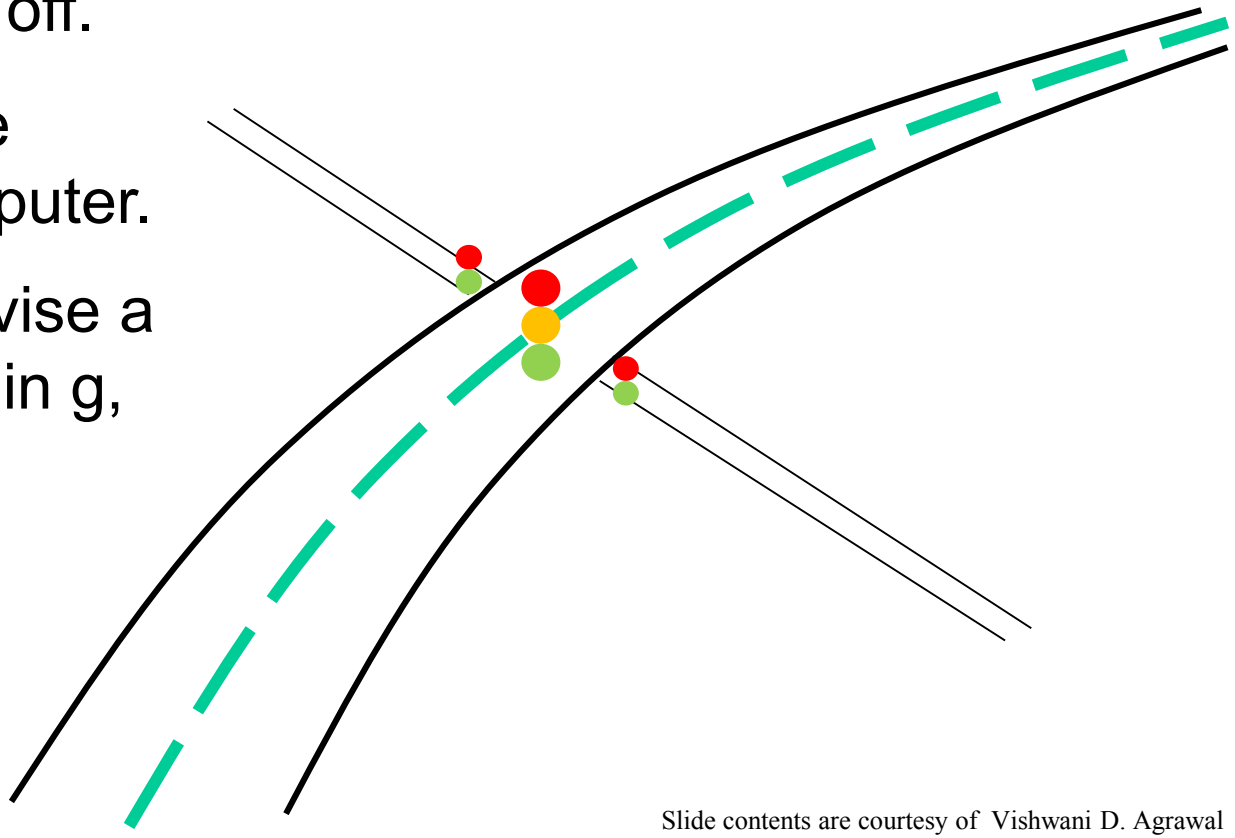
Truth Table with Don't Cares

- Don't cares entries represent combinations that are impossible to happen
- Example: X=1 indicates an elevator at top, and Y=1 indicates elevator at ground ➔ X and Y can never be one at the same time
- Don't care entries can be assumed any value and can thus leverage the simplification process
- There can be more than one minimal grouping, as a result of don't cares

A	*B*	*C*	*D*	*F*
0	0	0	0	*d*
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	*d*
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	*d*

Example: Traffic Signals

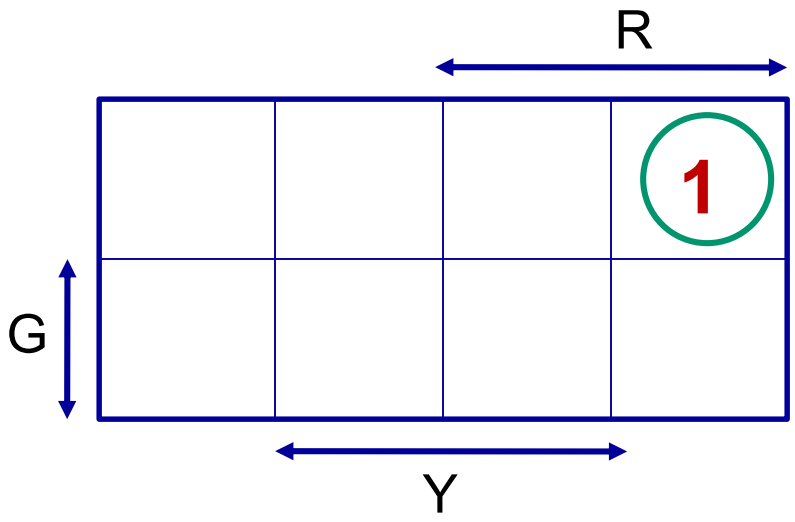
- Consider two roads crossing:
  - Highway with traffic signal, red (R), yellow (Y) or green (G).
  - Rural road with red (r) or green (g) signal.
- Here R, Y, G, r and g are Boolean variables; 1 implies light is on, 0 means light is off.
- Highway signals are controlled by a computer.
- We only need to devise a digital circuit to obtain g, because r = $\overline{g}$.



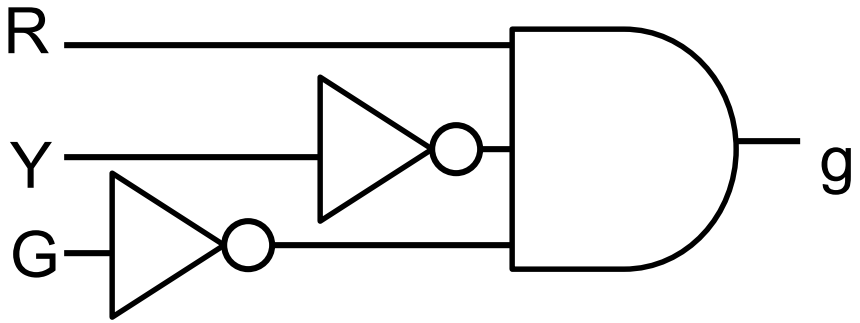
Slide contents are courtesy of Vishwani D. Agrawal

Completely Specified Function

Truth Table				
0	0	0	0	0
2	0	1	0	0
4	1	0	0	1
6	1	1	0	0

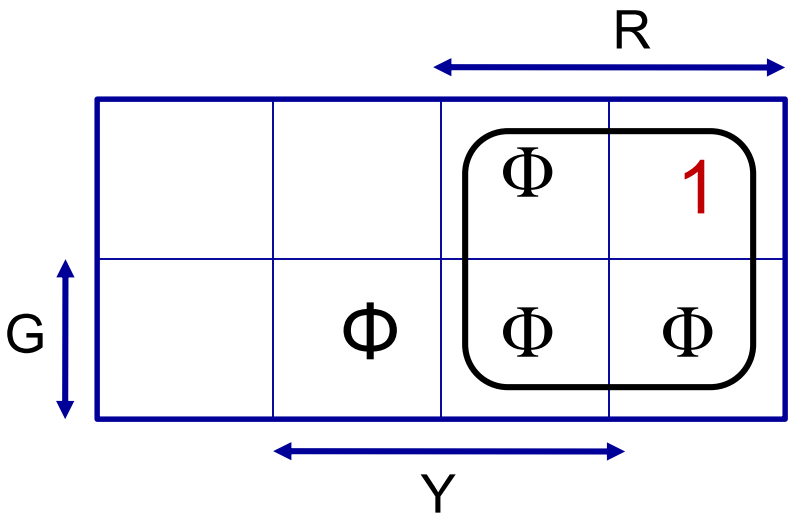


$$g = R\,\bar{Y}\,\bar{G}$$



Incompletely Specified Function

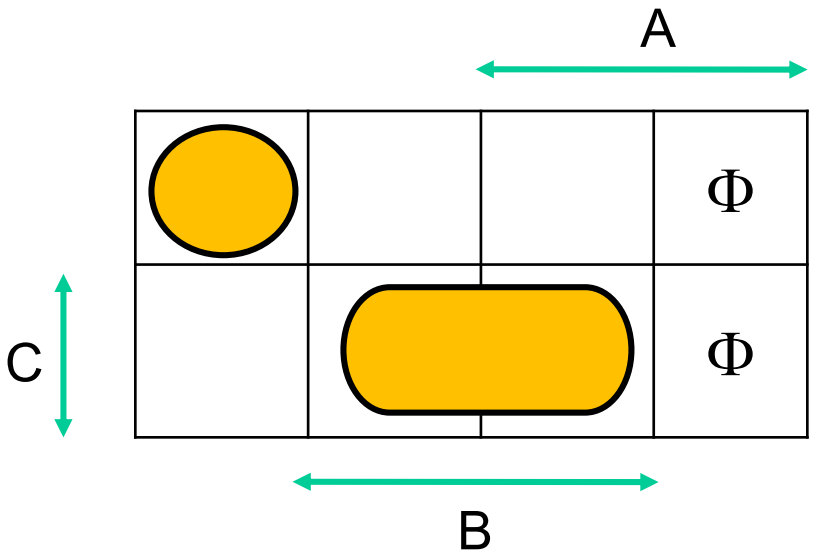
Truth Table				
0	0	0	0	0
2	0	1	0	0
4	1	0	0	1
6	1	1	0	Φ



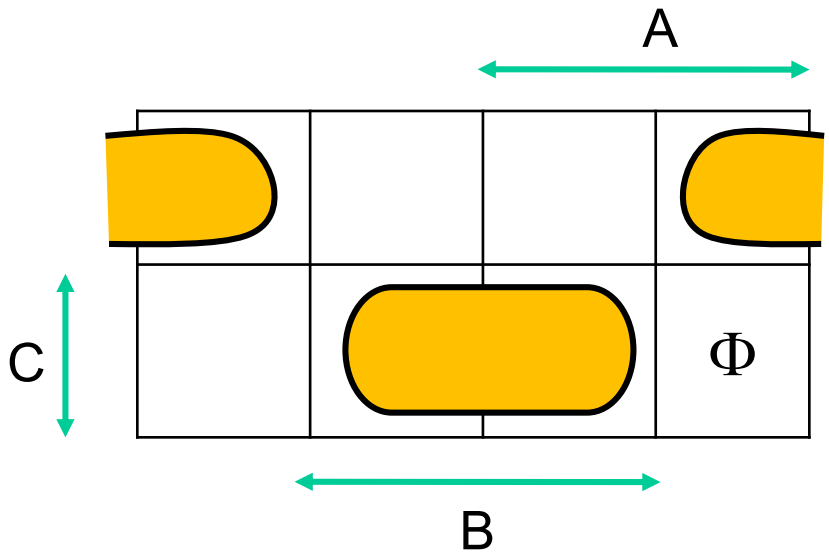
$g = R$

Functions with Don't Care Minterms

- F(A,B,C) = ∑m(0,3,7) + d(4,5)
- Include don't care minterms when beneficial



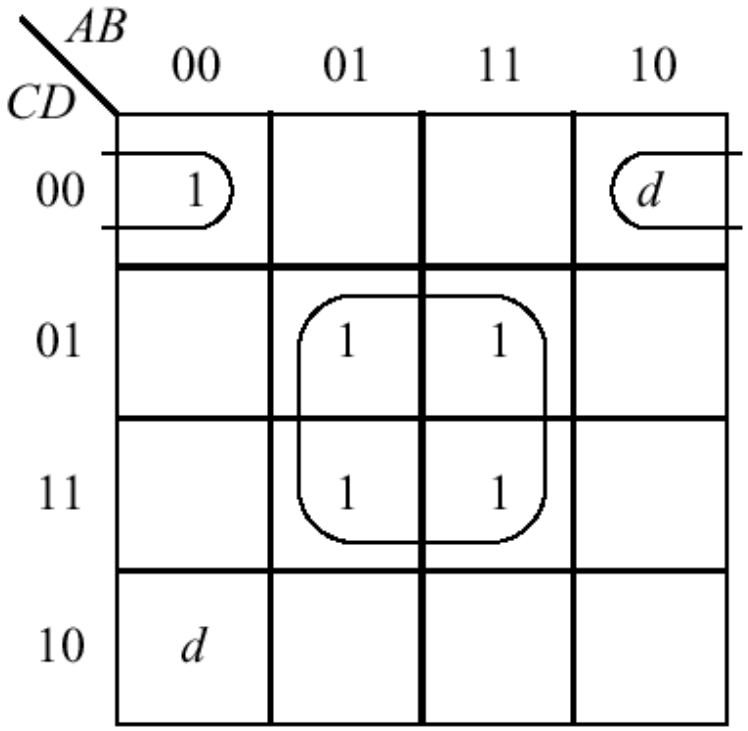
$F = B\,C + \bar{A}\,\bar{B}\,\bar{C}$



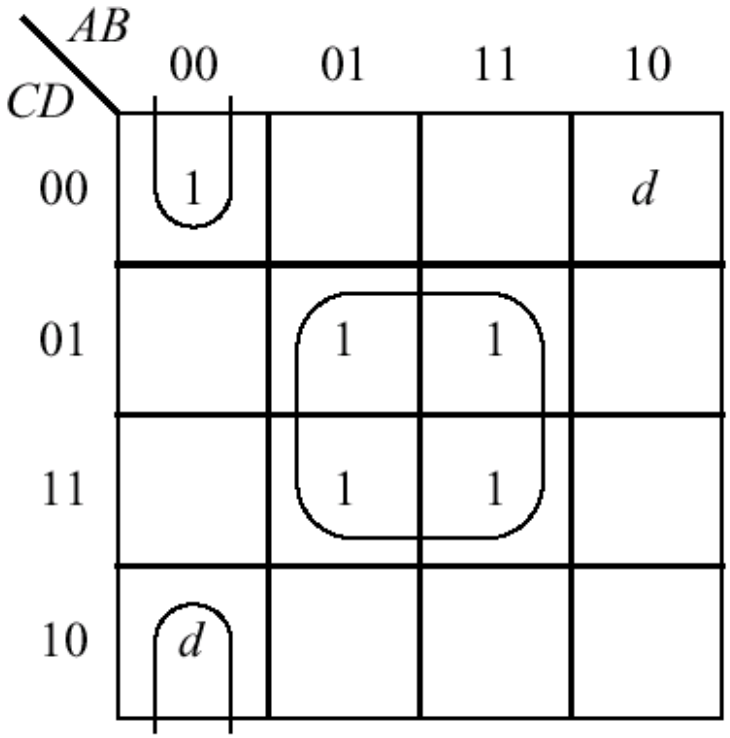
$F = B\,C + \bar{B}\,\bar{C}$

K-Maps and Don't Cares

- Don't care entries can be assumed any value and can thus leverage the simplification process
- There can be more than one minimal grouping, as a result of don't cares



$$F = \overline{B}\,\overline{C}\,\overline{D} + B\,D$$

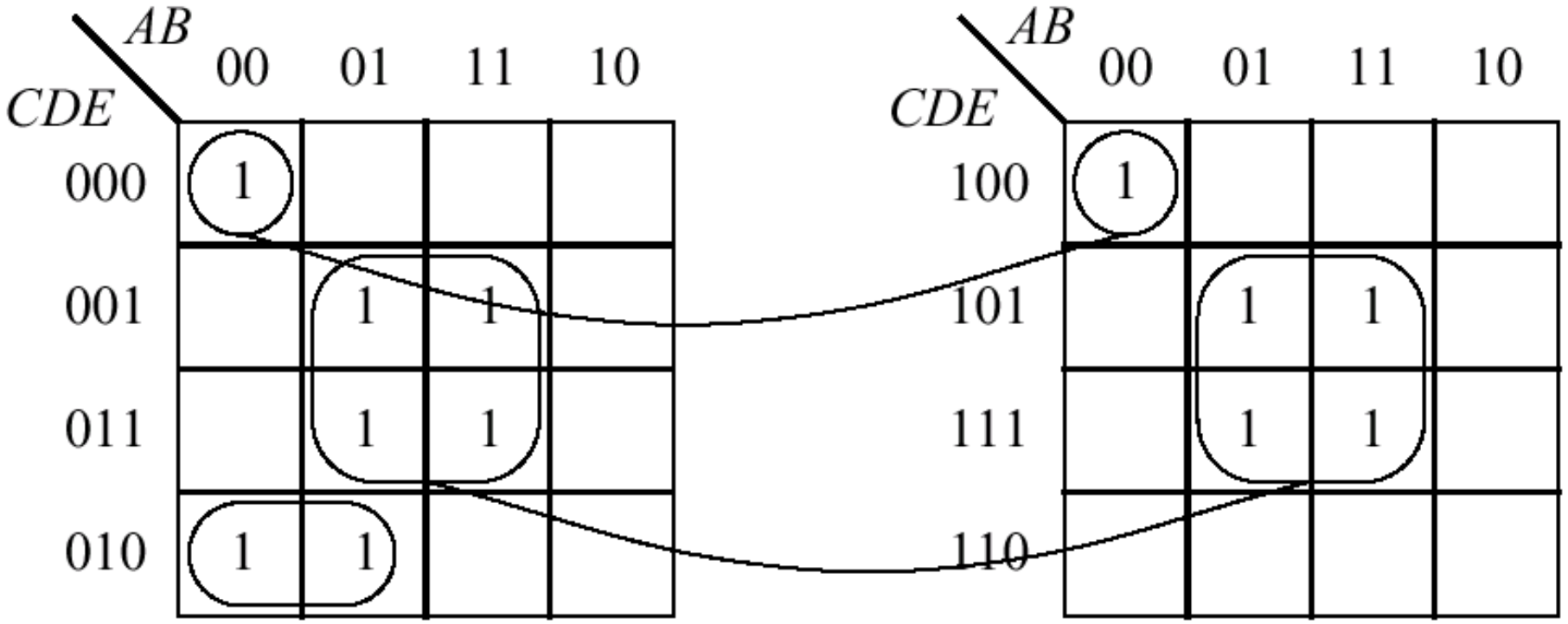


$$F = \overline{A}\,\overline{B}\,\overline{D} + B\,D$$

* Slide is courtesy of M. Murdocca and V. Heuring

Five-Variable K-Map

- Visualize two 4-variable K-maps stacked one on top of the other; groupings are made in three dimensional cubes.

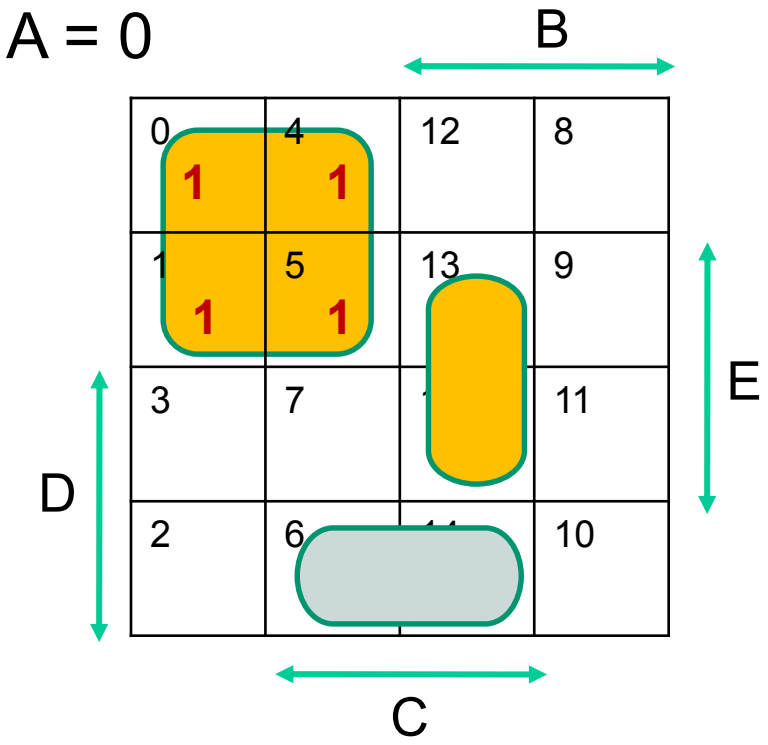


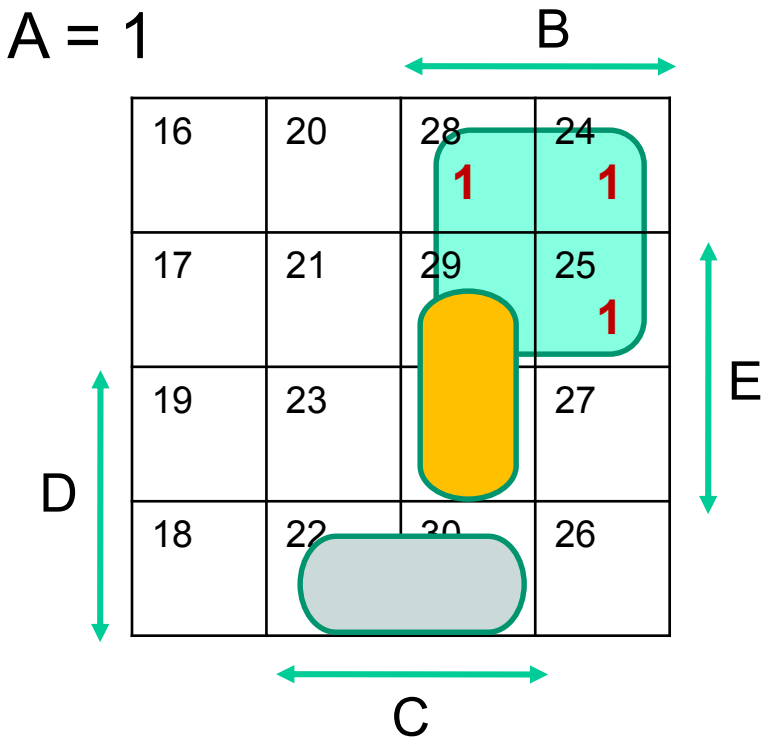
$$F = \overline{A}\,\overline{C}\,D\,\overline{E} + \overline{A}\,B\,D\,\overline{E} + B\,E$$

* Slide is courtesy of M. Murdocca and V. Heuring

Minimizing Five-Variable Function

$$F(A,B,C,D,E) = \sum m(0,1,4,5,6,13,14,15,22,24,25,28,29,30,31)$$

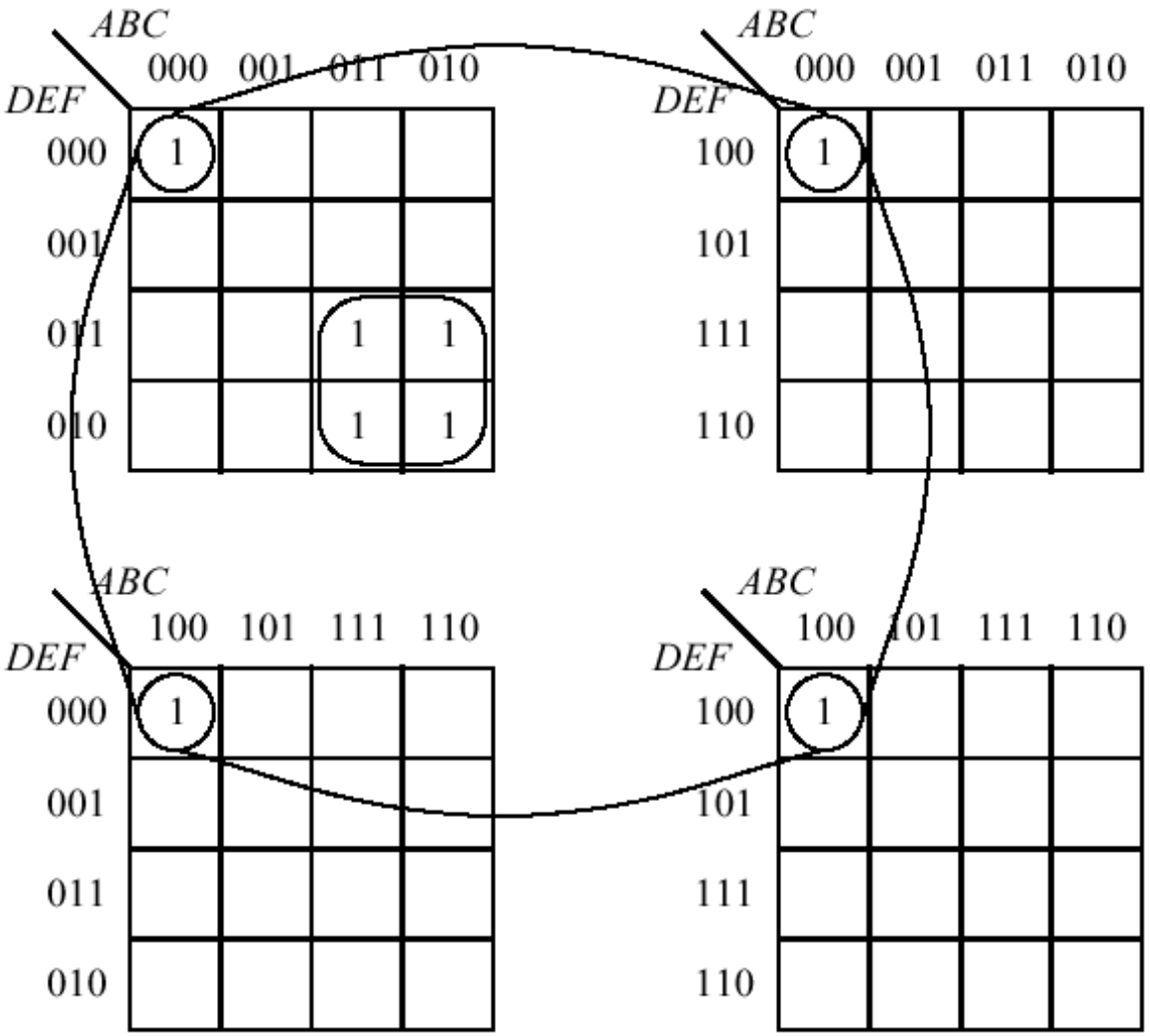




$$F = \bar{A}\,\bar{B}\,\bar{D} + A\,B\,\bar{D} + B\,C\,E + C\,D\,\bar{E}$$

Six-Variable K-Map

- Visualize four 4-variable K-maps stacked one on top of the other; groupings are made in three dimensional cubes.

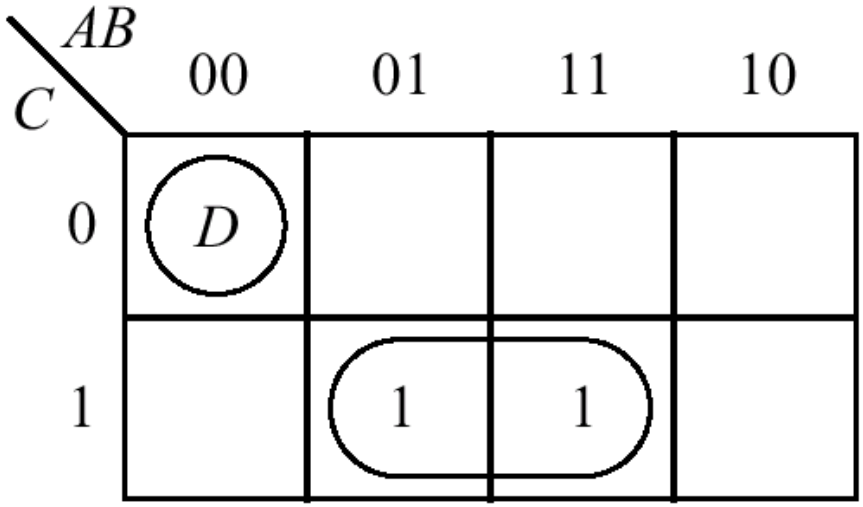


$$G = \overline{B}\,\overline{C}\,\overline{E}\,\overline{F} + \overline{A}\,B\,\overline{D}\,E$$

* Slide is courtesy of M. Murdocca and V. Heuring

Map-Entered Variables

- Allowing variables to be entered in the K-map simplifies the representation of some functions (less map size)



$$F = B\,C + \overline{A}\,\overline{B}\,\overline{C}\,D$$

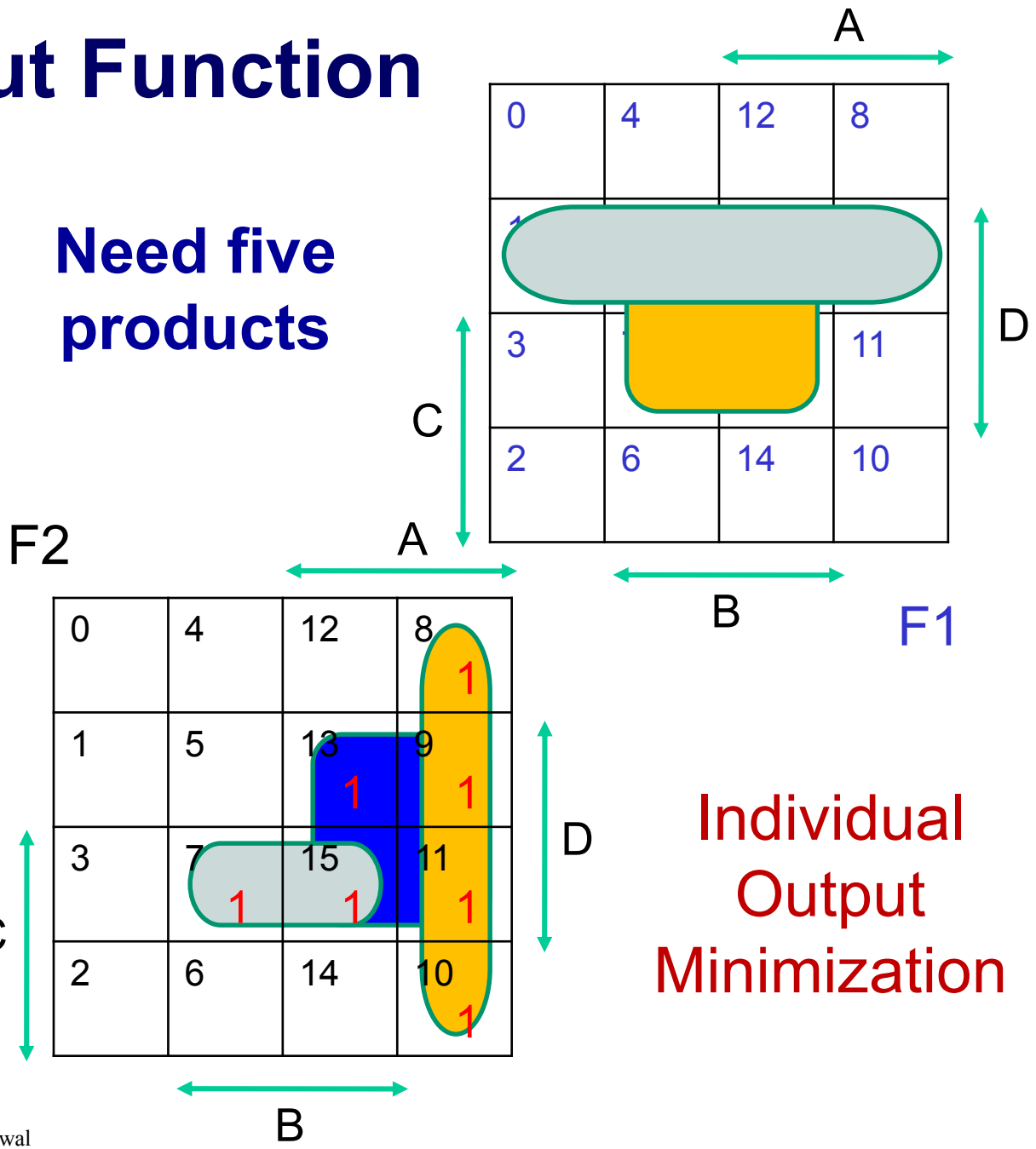
- The map-entered variable D is treated as a 1 for the purpose of grouping, which in this case results in a 1-group since there are no adjacent 1's to the D cell
- Notice that the variable D appears in the final expression

* Slide is courtesy of M. Murdocca and V. Heuring

Multiple-Output Function

Inputs				Outputs	
A	B	C	D	F1	F2
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	1

Need five products

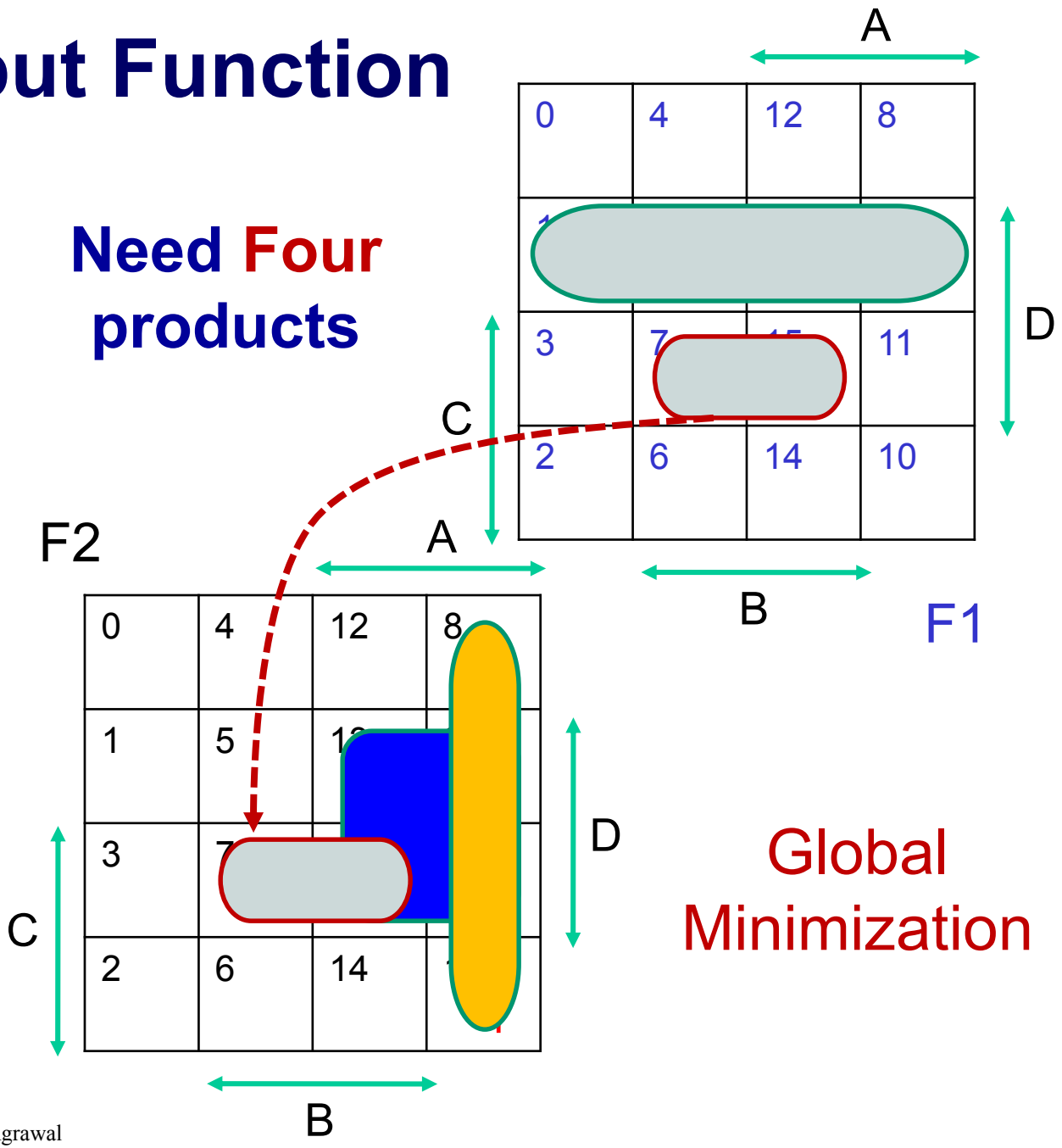


Individual Output Minimization

Multiple-Output Function

Inputs				Outputs	
A	B	C	D	F1	F2
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	1

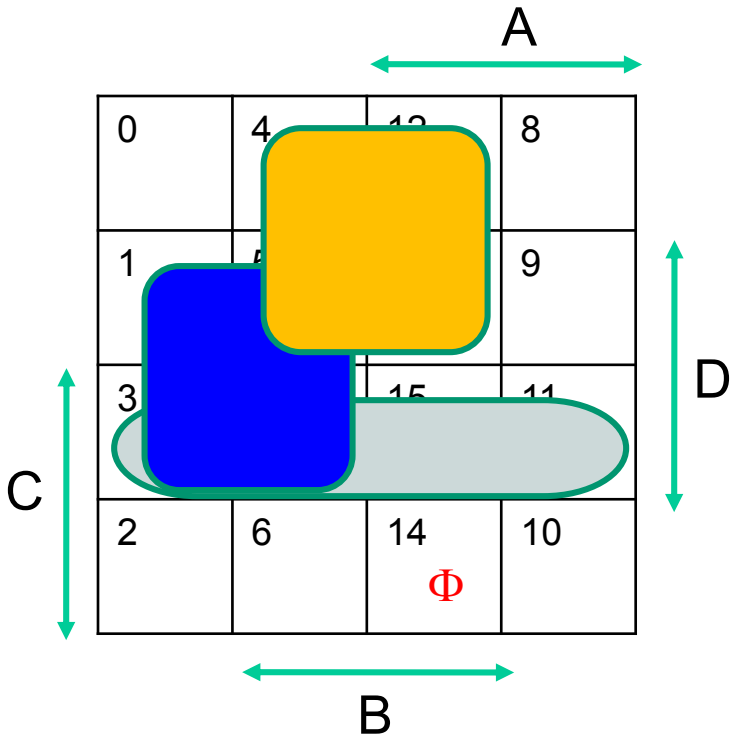
Need Four products



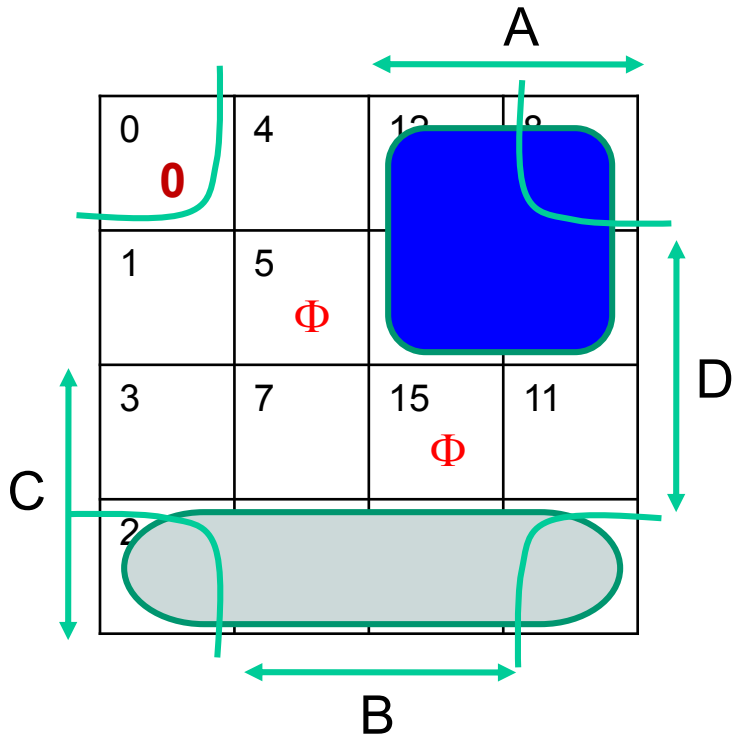
Global Minimization

Minimized SOP and POS

- $F(A,B,C,D) = \sum m(1,3,4,7,11) + d(5,12,13,14,15)$
  $= \prod M(0,2,6,8,9,10)\ D(5,12,13,14,15)$



$F = B\bar{C} + \bar{A}D + CD$



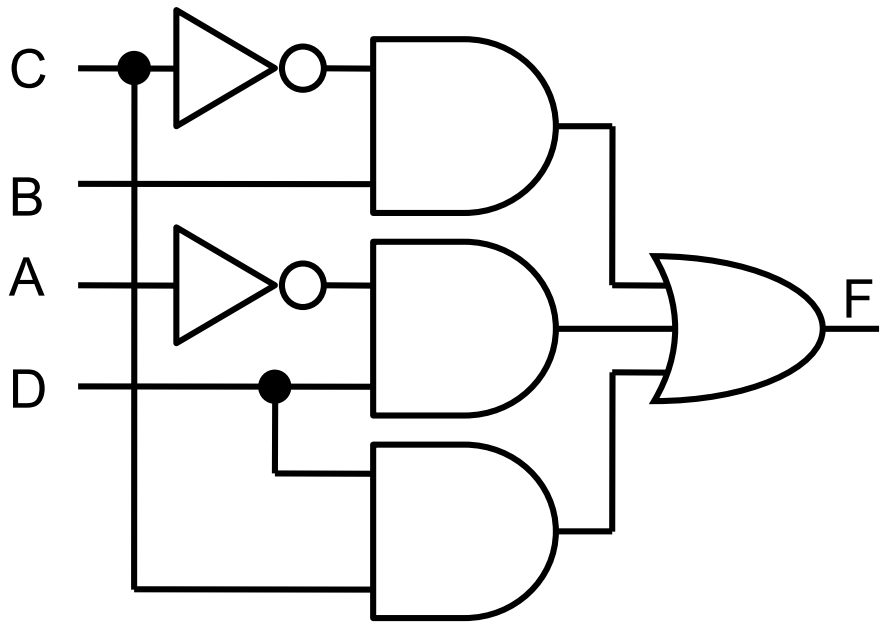
$\bar{F} = \bar{B}\bar{D} + C\bar{D} + A\bar{C}$

$F = (B + D)(\bar{C} + D)(\bar{A} + C)$

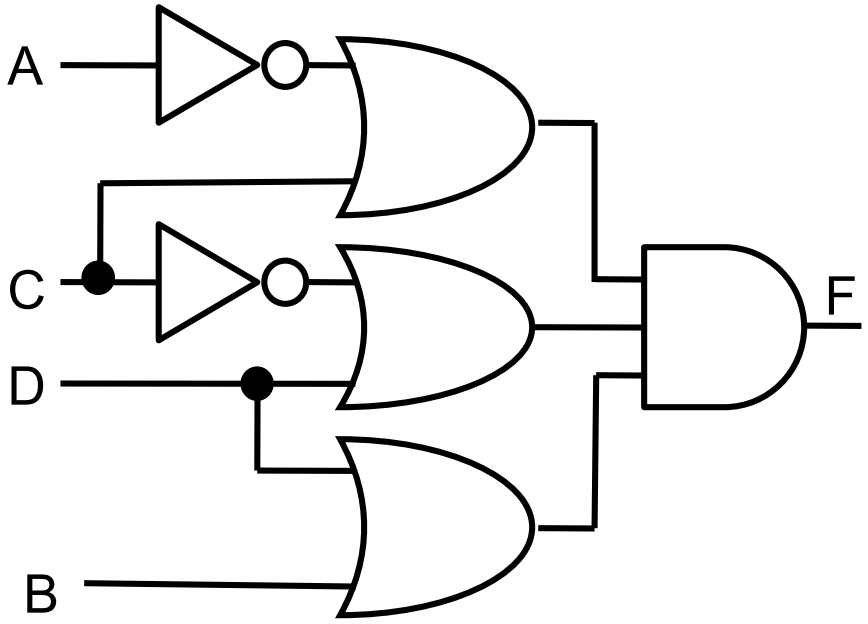
SOP and POS Circuits

- F(A,B,C,D) = $\sum$ m(1,3,4,7,11)+d(5,12,13,14,15)
  = $\prod$ M(0,2,6,8,9,10) D(5,12,13,14,15)

F = B $\bar{C}$ + $\bar{A}$D + C D



F = (B + D)($\bar{C}$ + D)($\bar{A}$ + C)



Are two circuits functionally Identical?

Speed and Performance

Design Area

- Gate counts and interconnects derive the required design area

Circuit Depth

- Circuit depth is the number of logic gates on the longest path between the input and output within the circuit
- Fan-in and Fan-out of logic gates affects the circuit depth
- Circuit depth can be reduced using function decomposition

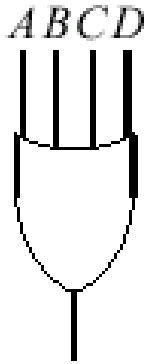
Circuit Speed

- The speed of a digital system is governed by:
  - the propagation delay through the logic gates
  - the propagation delay across interconnections

* Slide is courtesy of M. Murdocca and V. Heuring

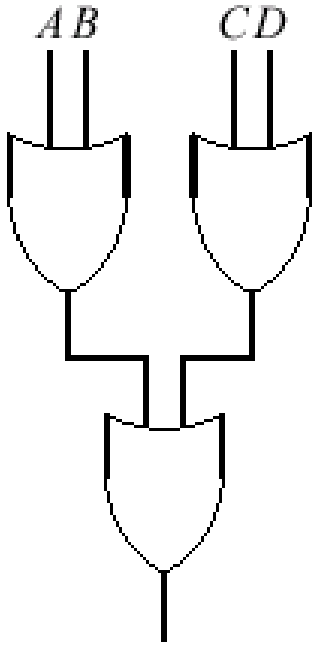
OR-Gate Decomposition

- Fan-in affects circuit depth.



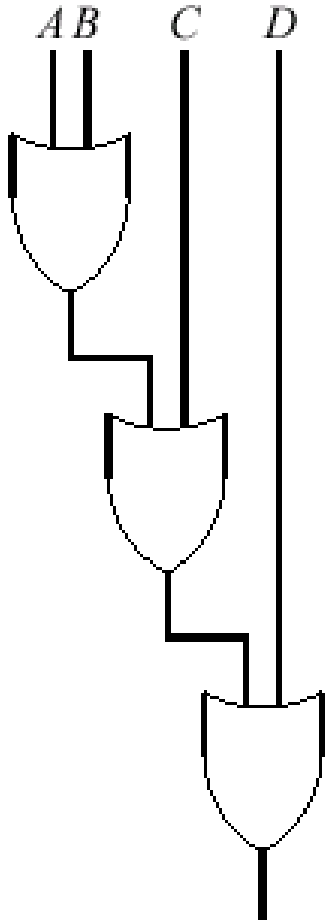
$A + B + C + D$

Initial high fan-in gate



$(A + B) + (C + D)$

Balanced tree



Degenerate trees increase circuit depth but facilitate circuit splitting on multiple stages

Associative law of Boolean algebra:

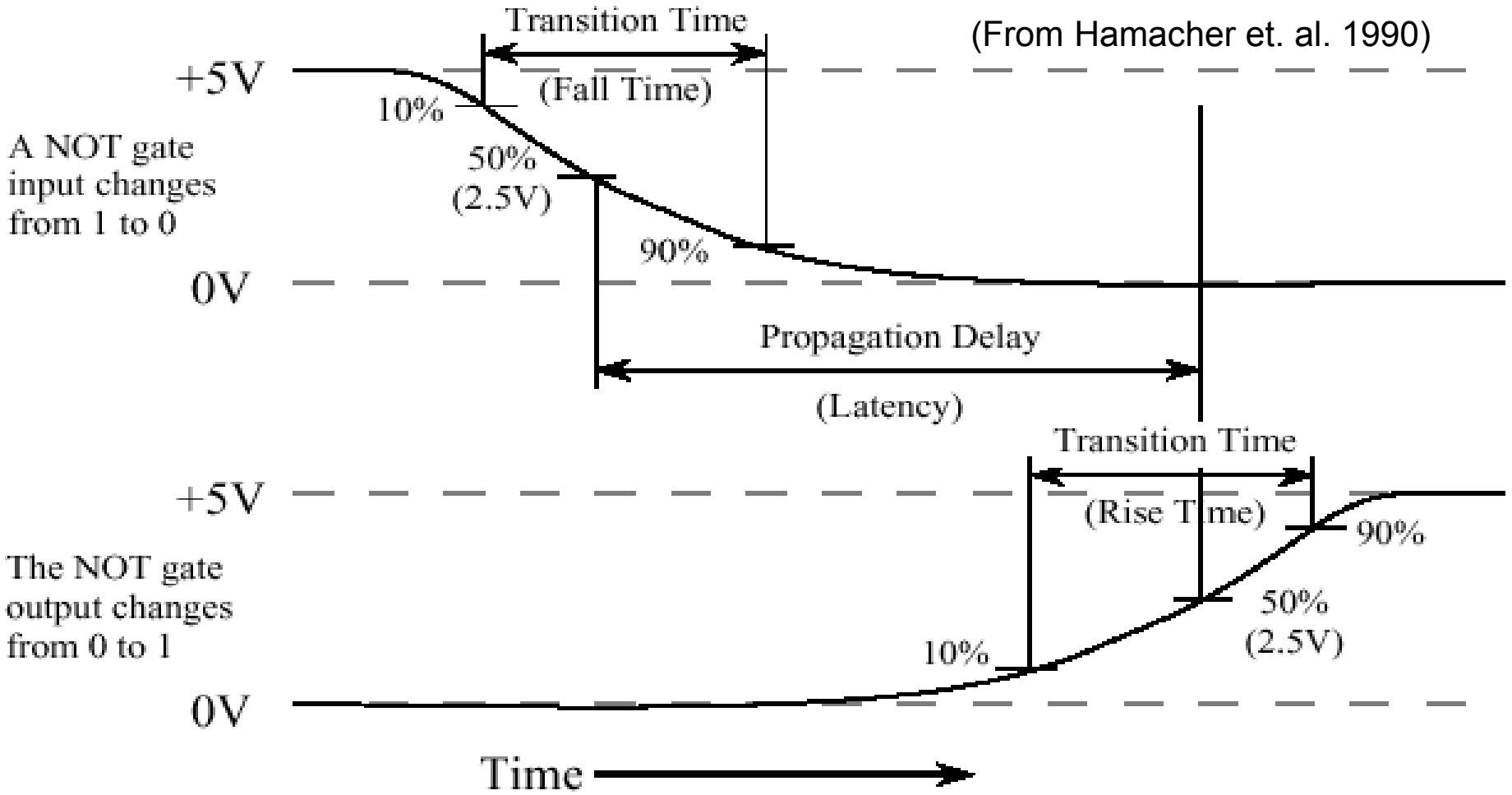
$A + B + C + D = (A + B) + (C + D)$

$((A + B) + C) + D$

Degenerate tree

* Slide is courtesy of M. Murdocca and V. Heuring

Propagation Delay

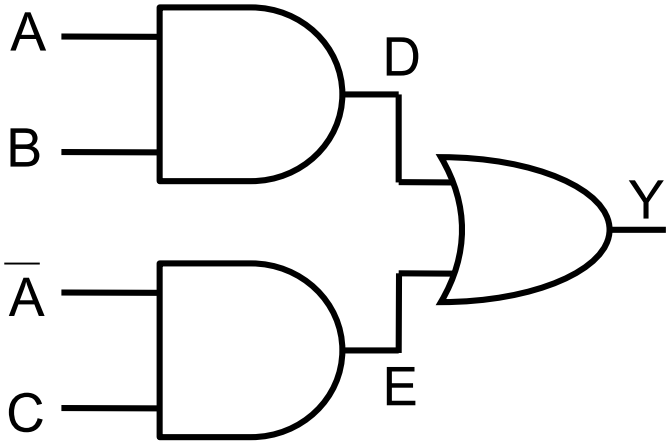


- Propagation delay depends on the technology used in the transistor's circuit
- As power consumption increases, propagation delay decreases
- High power consumption increases heat dissipation and requires circuit cooling

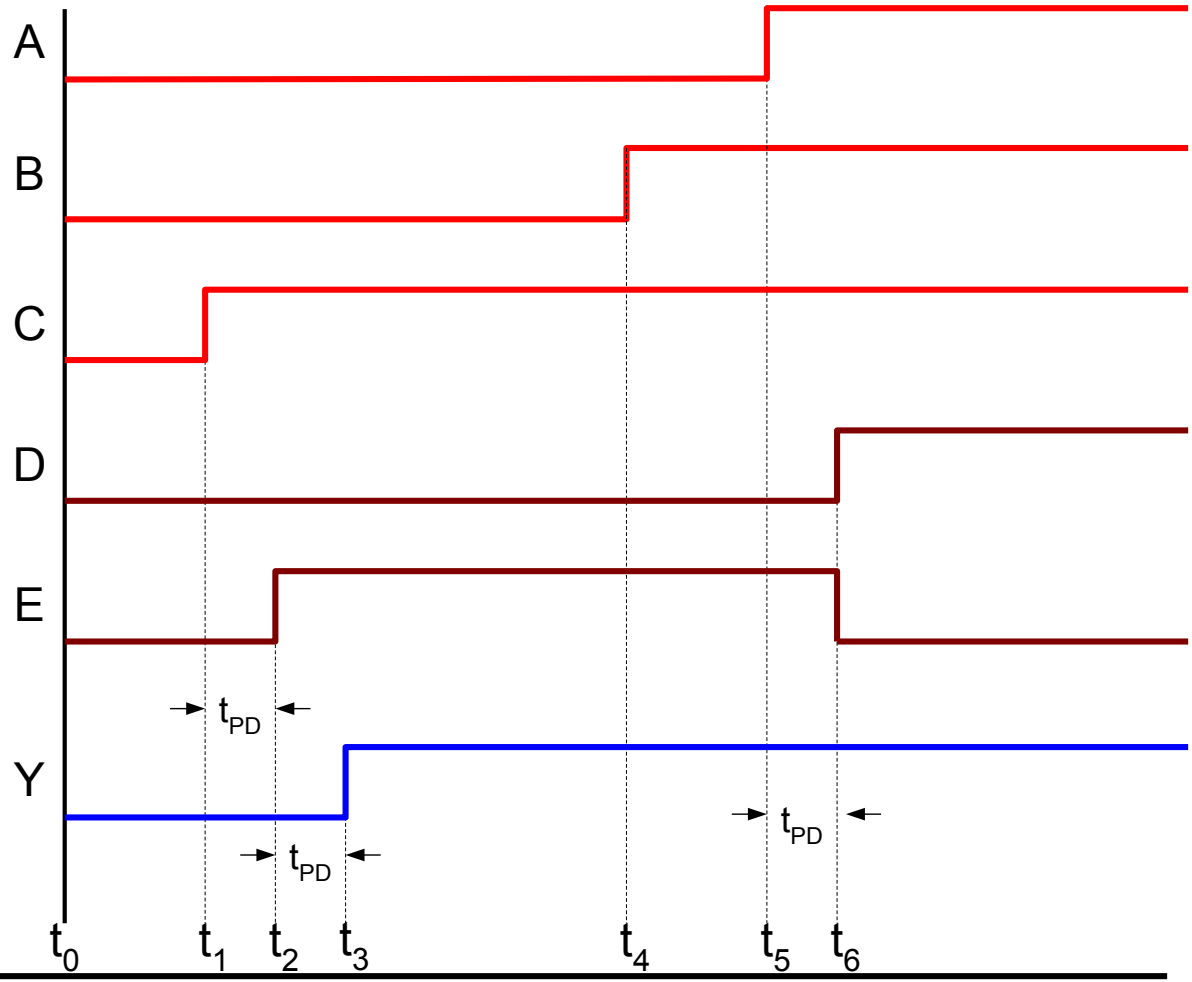
* Slide is courtesy of M. Murdocca and V. Heuring

Timing Hazard

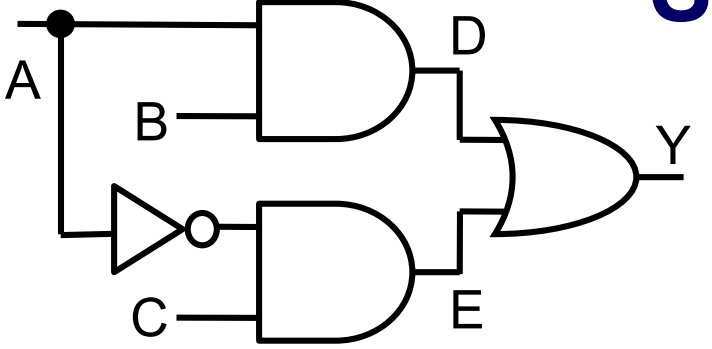
- Gate delay reflects the response time for an output of a gate to be reflect the result of a new input
- The gate delay is in the order of nanosecond, yet may not be the same for all devices



When all gates have the same response time, race conditions are avoided, and output “D” and “E” change simultaneously



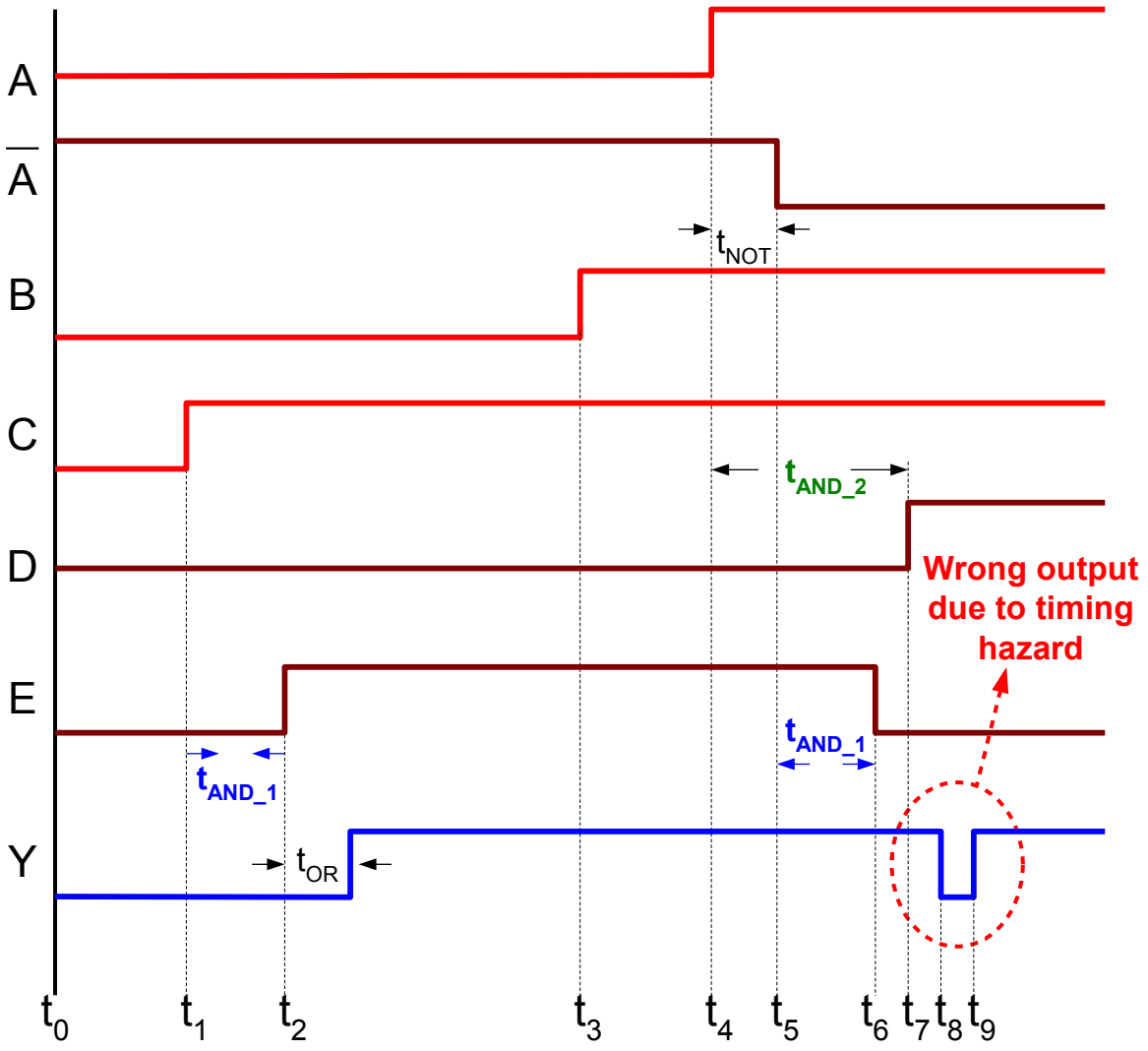
Static Hazard



What if $\bar{A}$ was not readily available?

- In some periods, the output will reflect the incorrect values and may cause problems to other circuits or the application
- Can be serious if the output is connected to an important device
- Called dynamic if it happens more than once until output settles

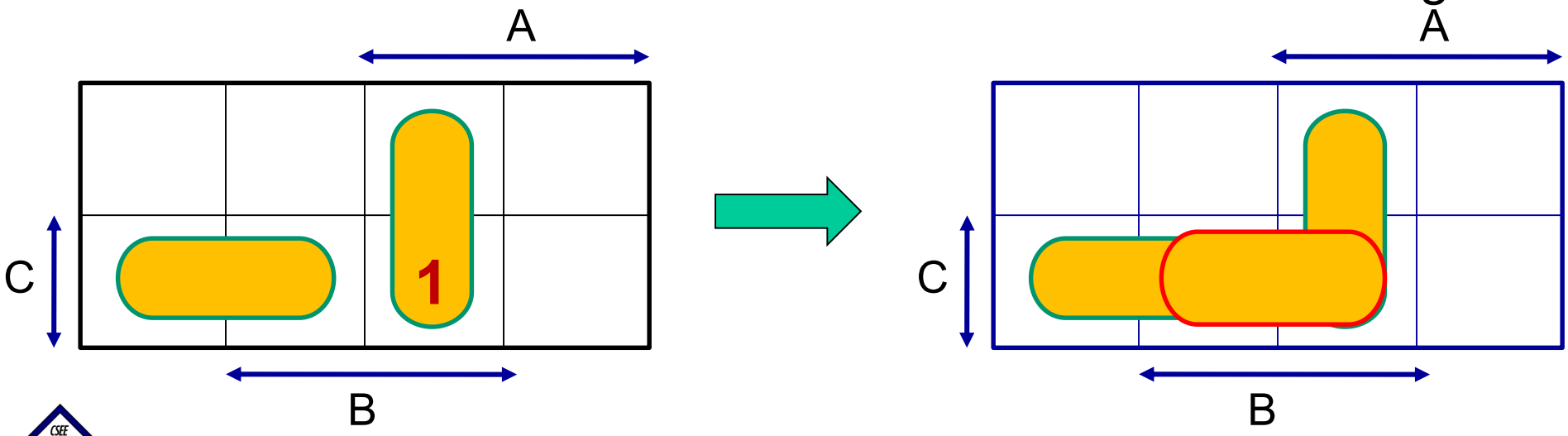
Gets more complicated if gate delay is not equal



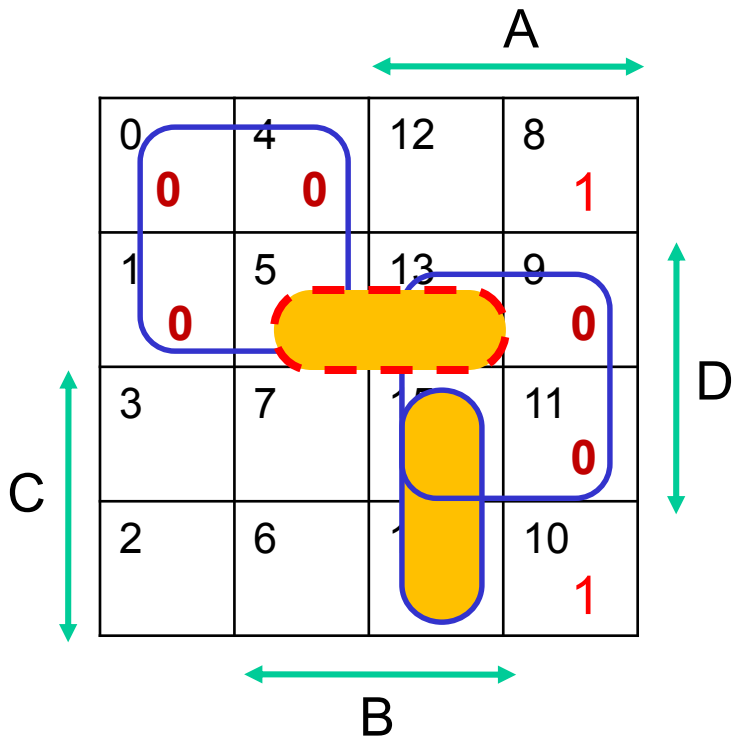
1 and 0 Glitches

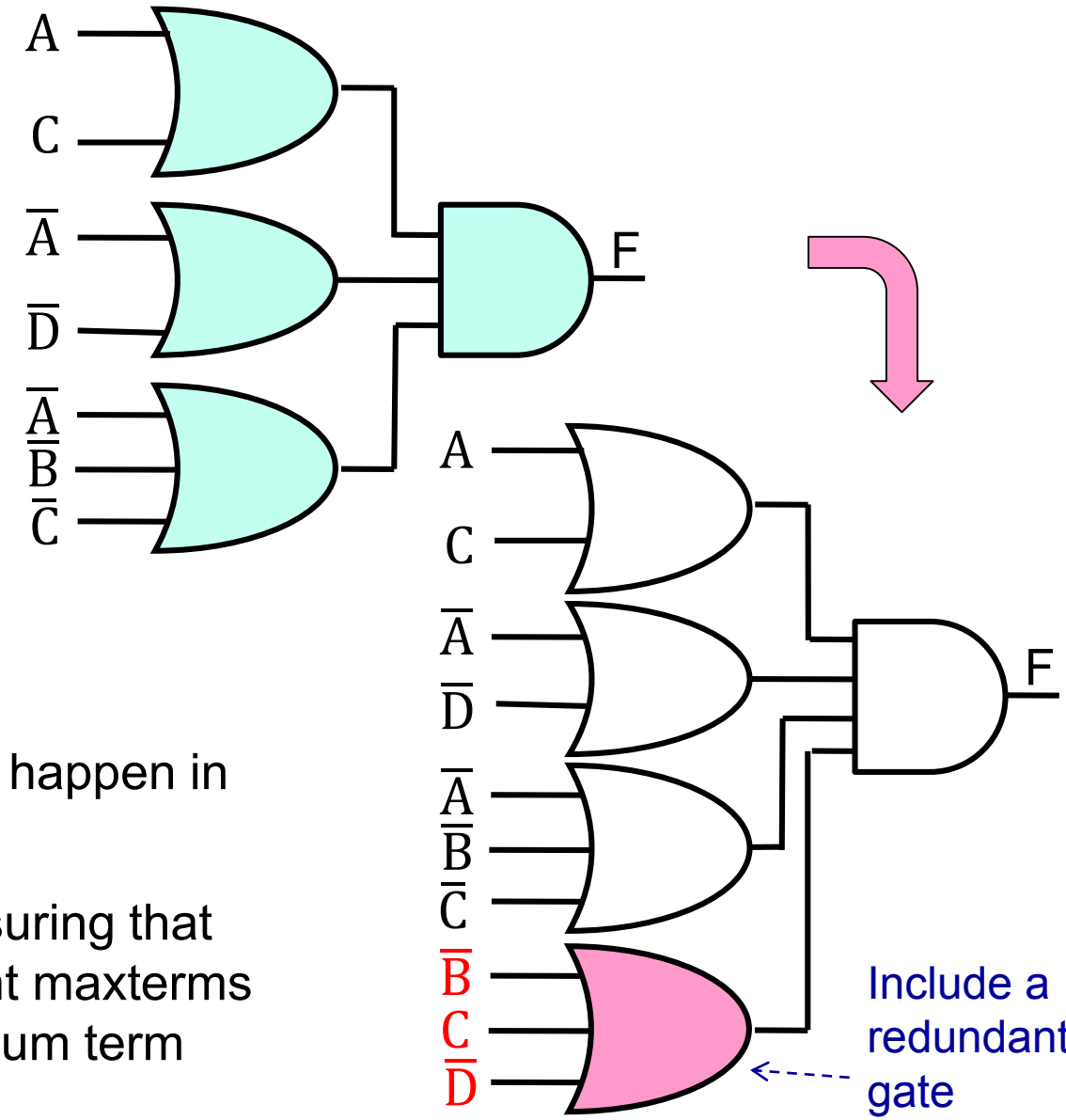
- Glitches refer to the case when momentary changes in the output takes place when no change should occur
- Static 1 hazard (1 glitch) occurs primarily in AND-OR circuits, while static 0 hazard (0 glitch) may happen in OR-AND circuits
- Static hazards can be prevented by ensuring that every pair of logically adjacent minterms (maxterms) are covered with a common product (sum) term

➔ Hazard can be eliminated with the inclusion of redundant gates



Example: 0 Glitch Avoidance





- Static 0 hazard (0 glitch) may happen in OR-AND circuits
- Prevent static hazards by ensuring that every pair of logically adjacent maxterms are covered with a common sum term

Conclusion

❑ *Summary*

➔Extended K-map procedure

(Minterm covering, Multi-output optimization, map-entered variable)

➔ Optimization of incompletely specified functions

(Exploiting don't care entries)

➔ Scalability limitation of the K-map method

(5 and 6 variables k-maps)

➔ Other circuit performance considerations

(timing hazards issues and countermeasures)

❑ *Next Lecture*

➔ The Quine-McCluskey algorithm

Reading assignment: Sections 3.6-3.8 in the textbook