Herbert Wertheim College of Engineering
UNIVERSITY of FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

Department of Electrical & Computer Engineering

Digital Logic And Computing Systems

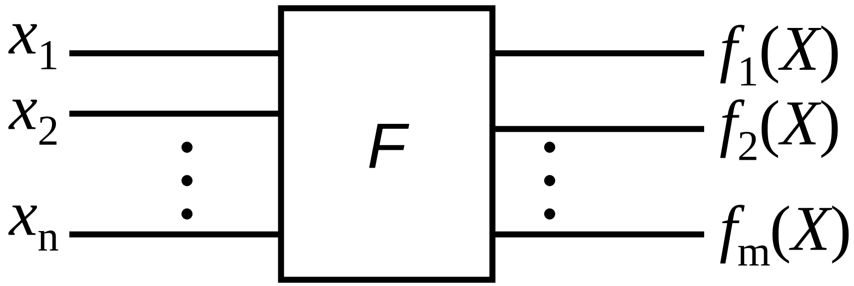
Chapter 03 – Combinational Circuit Design

Dr. Christophe Bobda

Agenda

- Combinational Circuits
- Synthesis of Combinational Circuits
- Logic-Optimization
  - Simplification Theorem
  - Canonical- and Normal Forms
  - Karnaugh-Method
  - Quine-McCluskey Algorithm

Combinational Circuits

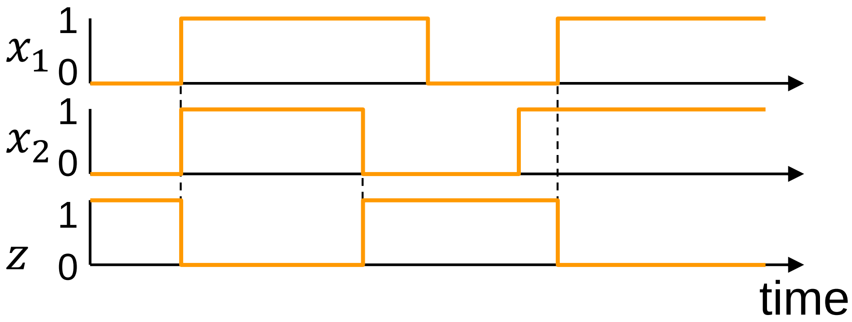


- Definition: A logic circuit with *n* inputs $X=(x_1,x_2,\ldots,x_n)$ and *m* outputs, $m \geq 1$, whereas $F(X)=(f_1(X), f_2(X), \ldots ,f_m(X))$, is called combinational circuit, if and if $F(X)$ is solely defined at a given time by the inputs X value at that time.
  - Every change in the inputs instantaneously impacts the outputs.
  - A combinational circuit has no memory.
    - Previous input values cannot be recalled.

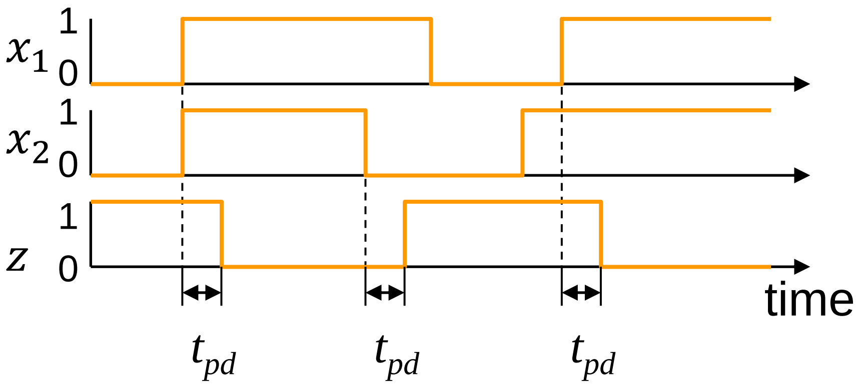
Delays

- Gates and signals have delays. Information is saved for a very short period of time

- Timing diagram: ideal gate



- In reality



t_{pd} … propagation delay

Synthesis of Digital Circuits

- Design and implementation of circuits to perform a given function
  - Given: Specification; Library: AND, NOR, NOT, etc....
  - Result: A circuit that implements the specification with the library elements

- Synthesis steps
  1. Define the inputs and outputs of the circuit
  2. Draw a truth table for every output as a function of all inputs
     a) Compute the logic expression for every output
     b) Minimize the expression if needed
  3. Use available library elements to assemble your circuit
     - Possible transformation needed

Synthesis of Digital Circuits

- Example: Circuit to control an elevator
  - Goal: Prevent riding under dangerous conditions (door open)
    - Ride only possible if the door is closed and the load is under limit
- Step 1: Define Inputs and Outputs
  - A → Door (A = 0 → door open, A = 1 → door closed)
  - B → Weight (B = 1 → above limit, B = 0 → below limit)
  - C → Button (C = 1 → button pressed, B = 0 → button not pressed)
  - Z → Output (Z = 1 → engine on, Z = 0 → engine off)

Synthesis of Digital Circuits

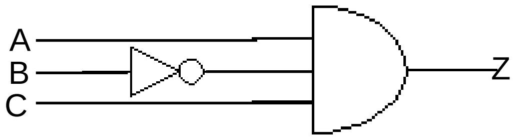
- Step 2:
  - Truth table

A	B	C	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

  - Logic expression

$$Z = A \wedge \overline{B} \wedge C$$

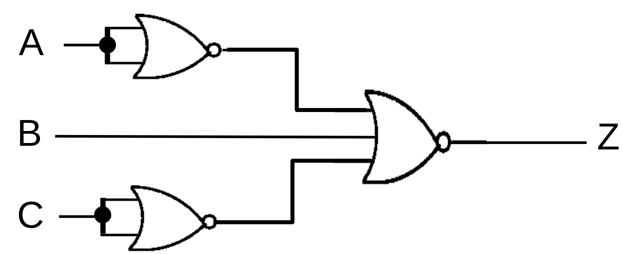
    - No more reduction needed



Synthesis of Digital Circuits

- Step 3:
  - Assuming only NOR-gates available

$$Z = A \wedge \overline{B} \wedge C = \overline{\overline{A \wedge \overline{B} \wedge C}} = \overline{\overline{A} \vee B \vee \overline{C}}$$



- Function Minimization
  - Boolean Algebra rules
  - Karnaugh Method
  - Quine McCluskey

Logic Optimization

- Minimization: Step 2 b) Simplification → design the optimal circuit
  - ⇒ Optimization problem with various cost functions

- The cost functions depends on design goals and the underlying technology
  - Common design goals
    - Monetary cost
      - Low amount of gates and connections, low number of input per gate
    - High speed
      - Parallel processing, low number of stages
    - Low power

Case Study

- Combinational Adder: Design a circuit to perform the addition of 2 2-bit numbers

Case Study

- Simplification
  - $(A+B)(A+C) = ?$
  - $A'BC + AB'C + A'BC + ABC' + ABC = ?$

  - Consensus Theorem: $AB + A'C + BC = AB + A'C$

Digital Circuit Design Simplification

- Theorem: Absorption

a) $x + (x \cdot y) = x$

Proof

$$x + (x \cdot y) = x \cdot 1 + x \cdot y$$
$$= x \cdot (1 + y)$$
$$= x \cdot 1$$
$$= x$$

b) $x \cdot (x + y) = x$

Proof

$$x \cdot (x + y) = x \cdot x + x \cdot y$$
$$= x + x \cdot y$$
$$= x$$

AB +A(B+C) + B(B+C)

Digital Circuit Design Simplification

- Theorem:

$$a)\, x + (\bar{x} \cdot y) = x + y$$

Proof

$$\begin{aligned} x + (\bar{x} \cdot y) &= (x + \bar{x}) \cdot (x + y) \\ &= 1 \cdot (x + y) \\ &= x + y \end{aligned}$$

$$b)\, x \cdot (\bar{x} + y) = x \cdot y$$

Proof

$$\begin{aligned} x \cdot (\bar{x} + y) &= (x \cdot \bar{x}) + (x \cdot y) \\ &= 0 + (x \cdot y) \\ &= x \cdot y \end{aligned}$$

Canonical Forms

- Definition (Literal): Boolean variable or complement of a Boolean variable
- Positive a Negative $\bar{b}$
- Definition (Product term): AND- Operation on $k \geq 1$ literals

  a abc̄ $\bar{b}d$ $\bar{a}b\bar{c}d$

- Definition (Sum term): OR-Operation on $k \geq 1$ literals

  $b + \bar{d}$ $a + \bar{b} + c$

- Definition (Minterm): Product term in which all variables appear exactly once (negated or non negated)

  $\bar{a}b\bar{c}d$ abcd $\bar{a}\bar{b}\bar{c}\bar{d}$

Canonical- AND Normal Form

- Definition (Maxterm): Sum term in which all variables appear exactly one, negated or non negated

  $a+\bar{b}+c+d \qquad a+b+c+d \qquad \bar{a}+b+\bar{c}+\bar{d}$

- Theorem (Canonical Disjunctive Normal Form (CDNF)): Any Boolean function can be represented uniquely by a sum (OR) of minterms
  - From truth table, the CDNF is built by summing up all minterms for which the function value is 1
  - From a CDNF expression, built the function's truth table by entering a 1 in each function line corresponding to a minterm of the CDNF

Canonical- AND Normal Form

- Theorem (Disjunctive Normal Form (DNF)), Sum of Products (SOP)): Every Boolean Function can be represented as sum (OR-operation) of product terms
  - From the truth table: the DNF is constructed by summing (OR-ing) all minterms that for which the value of the function is 1
  - By simplification one can determine DNFs
- Implementation as 2-level circuit
  - Very fast, higher amount of gates

Canonical- AND Normal Form

- Compact representation of the Boolean function
  - Representation of minterms as code-word

$$m_5 = \bar{a}b\bar{c}d \qquad m_8 = a\bar{b}\bar{c}\bar{d}$$

  - Representation of a function as sum of all minterm-Code-words

$$\sum_{1}^{n}(8,7,3,1,11) = m_8 + m_7 + m_3 + m_1 + m_{11}$$
$$= a\bar{b}\bar{c}\bar{d} + \bar{a}\mathbf{bcd} + \bar{a}\bar{b}\mathbf{cd} + \bar{a}\bar{b}\bar{c}d + a\bar{b}\mathbf{cd}$$

Canonical- AND Normal Form

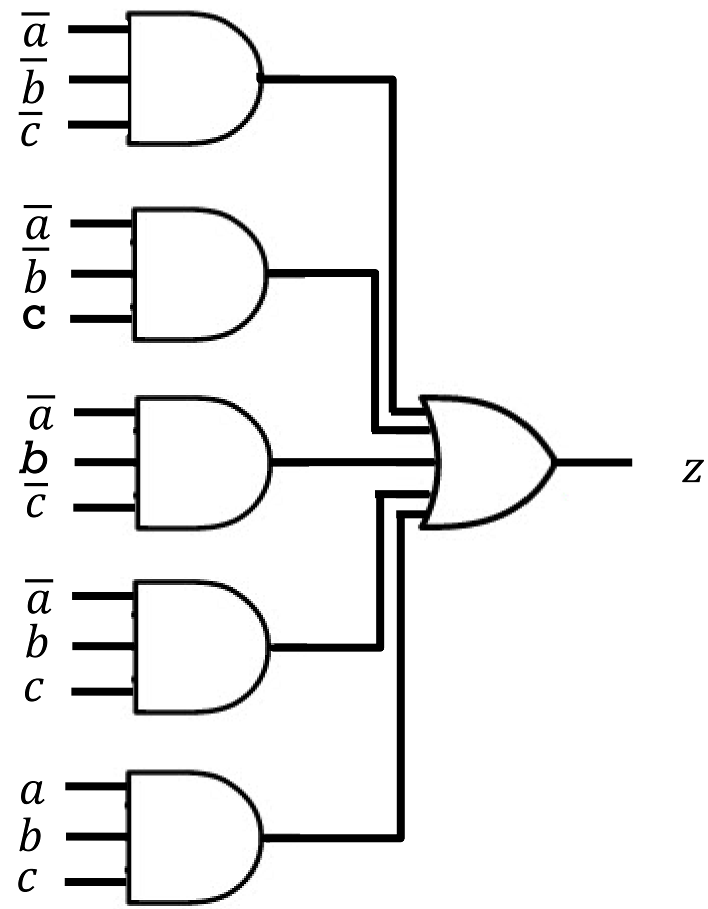
- Function $z(a,b,c)$

$(i)_{10}$	a	b	c	z
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

- Sum of minterms

$$z(a,b,c) = m_0 + m_1 + m_2 + m_3 + m_7 =$$
$$= \overline{a}\,\overline{b}\,\overline{c} + \overline{a}\,\overline{b}c + \overline{a}b\overline{c} + \overline{a}\mathbf{bc} + \mathbf{abc}$$

- Corresponding Circuit

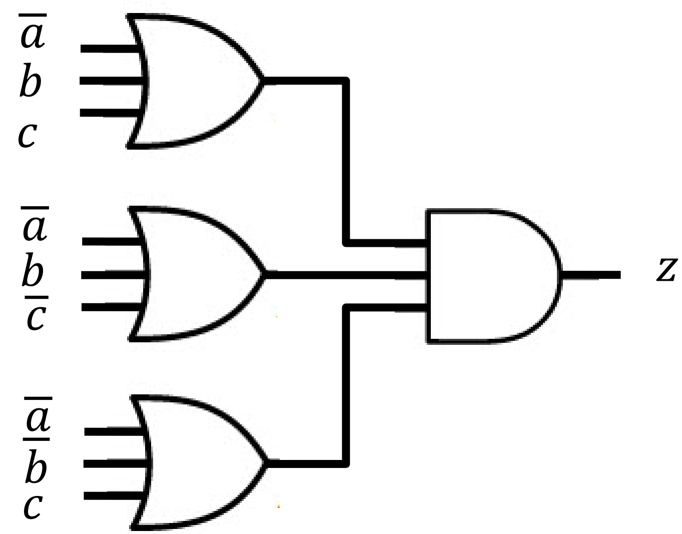


Canonical- AND Normal Form

- Function $z(a,b,c)$

$(i)_{10}$	a	b	c	z
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

- Corresponding Circuit



- Product of Maxterms

$$z(a,b,c) = M_4 \cdot M_5 \cdot M_6 =$$
$$= (\overline{a} + b + c) \cdot (\overline{a} + b + \overline{c}) \cdot (\overline{a} + \overline{b} + c)$$

Logic Optimization

- Theorem: for a Boolean expression *E* and a variable *x* we have

$$\boldsymbol{E}\mathbf{x} + E\overline{x} = E$$

- Proof: $E\mathrm{x} + E\overline{x} = E(\mathrm{x} + \overline{x}) = E$
  - Applying this theorem reduces a SOP-form by one product term
  - The theorem is the basis of most optimization methods for 2-level logic

- Definition (Irredundant Sum): A SOP-form, that cannot be reduced by any product term anymore is call irredundant sum

Logic Optimization
Minimization Problems

- Minimization Problem: Given a function $z(X)$, compute a SOP-form E for $z(X)$, so that:
  1. E has a minimum number of product terms

     Necessary (but not sufficient condition). E is a irredundant sum

  2. Any other SOP-form E' of $z(X)$ with a minimum number of product terms has at least the same number literals

- E is called mimimal SOP

Logic Optimization
Implicant, prime implicant, Primfactors

- Definition (Implicant): An Implicant p of a function z is a product term that fulfills the following :

$$\forall X: \quad p(X)=1 \Rightarrow z(X)=1$$

  - X is an assignment of the variables $x_i\,(0/1)$
  - Any minterm of p is also a minterm of z
  - An implicant can cover multiple 1s in the truth table of z

- Simplified definition: An implicant is a product term for which the value of the function is 1 in the truth table or in the Karnaugh-Map

Logic Optimization
Implicant, Prime implicant, Primfactors

- Definition(Prime implicant): An Implicant p of a function z is called Prime implicant, if the removal of any literal from p does not result in an implicant of z
  - Prime implicants are therefore no more reducible implicants of a function

Logic Optimization
Implicant, Prime implicant, Primfactors

- Example: The function $z(a,b,c) = ab + bc + a\bar{b}c$

  has the following implicants: $ab,\ bc,\ a\bar{b}c$

  but also: $\bar{a}bc,\ ab\bar{c},\ a\bar{b}c,\ abc$

  prime implicants are only: $ab,\ bc,\ ac$

a	b	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logic Optimization
Implicant, Prime implicant, Primfactors

Theorem: let *E* be a minimal SOP expression for a function *z*, then any product term of *E* is a prime implicant of *z*

Proof through contradiction (Exercise)

In other words, the mimimal SOP expression is a sum of prime implicants

However, the theorem does not state which prime implicants must be selected

Logic Optimization
Mimimization Approach

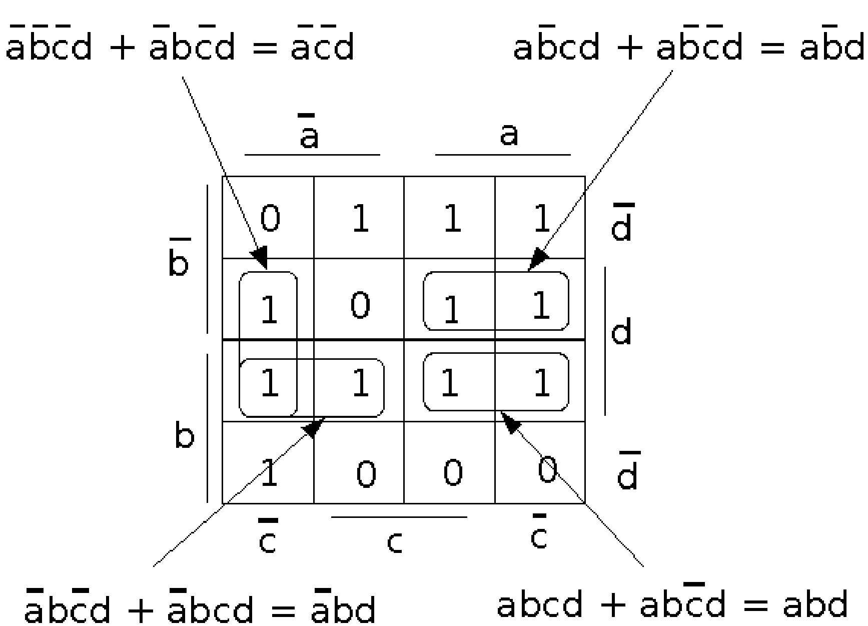
- Minimization of 2-level logic functions in 2 steps:
  - Step 1: find all prime implicants
  - Find the minimum number of prime implicants, which cover all minterms of the function (covering problem).
    - If there are many possibilities, then chose the ones that produces the minimum number of literals (cost function)
- Minimization approach
  - Karnaugh-Map: graphical method, for very small cases (up to 5 variables)
  - Quine-McCluskey: tabular method, used to compute exact solution of the minimization problem (efficient on small number of variables)

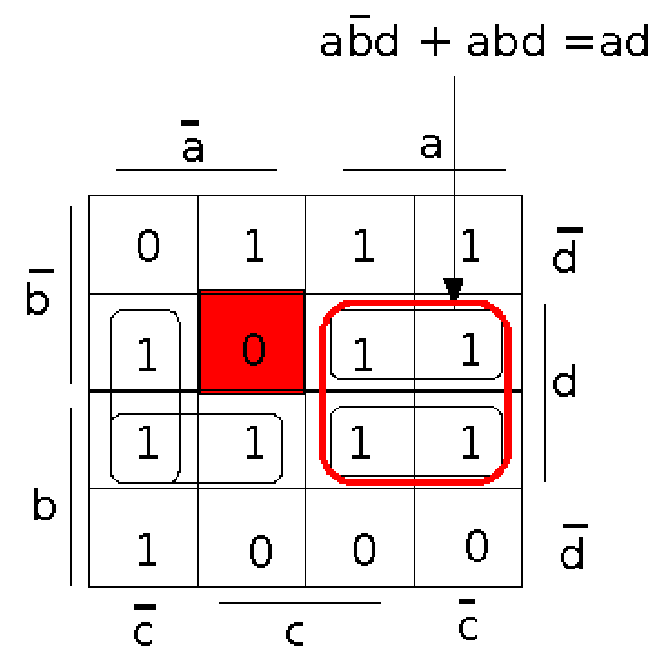
Logic Optimization
Karnaugh Method– K-Map

- A Karnaugh-Map (K-Map) for a n-variable function is a special graphical representation of that function's truth table
  - 2^n cells, each of which corresponds to a row of the truth table
  - Each literal is associated with a row or a column
  - A cell has a unique address (numbering)
    - which is defined through cell-column/cell-row or
    - through a binary number, which is defined by literals in the column and row
  - Neighbor cells differs only on 1 position (bit) in their numbering
  - A K-Map is built by entering the value 1 in the address defined by the corresponding minterm from the truth table

Logic Optimization
Karnaugh Method– K-Map

- Cell arrangement insures that product terms with k literals are represented through rectangular blocks of 2^{n-k} connected cells
  - $k=n \rightarrow$ Minterm $\rightarrow$ 1 cell
  - $k=0 \rightarrow$ Constant (0-Function, 1-Functon) $\rightarrow$ Block with all cells





Logic Optimization
Karnaugh Method– K-Map

- K-Map for n=2

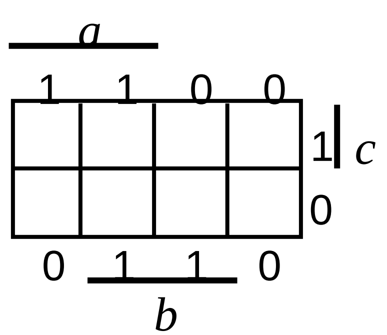
a	b	z
0	0	0
0	1	1
1	0	1
1	1	0

	$\bar{a}$	a
$\bar{b}$	0	1
b	1	0

- K-Map for n=3

a	b	c	z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

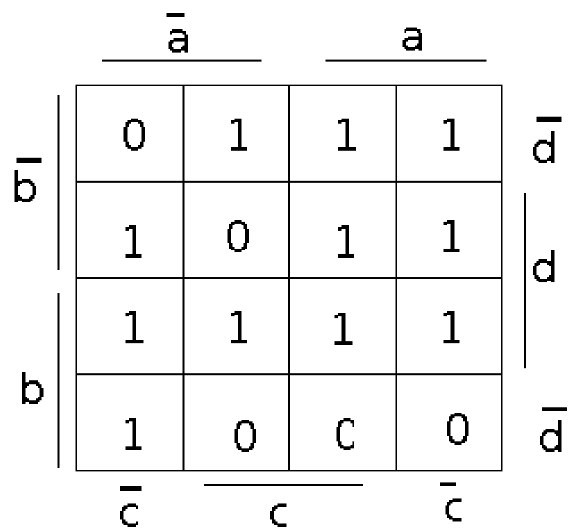
	$\bar{a}$	$\bar{a}$	a	a
$\bar{b}$	0	1	1	0
b	1	0	1	0
	$\bar{c}$	c	c	$\bar{c}$

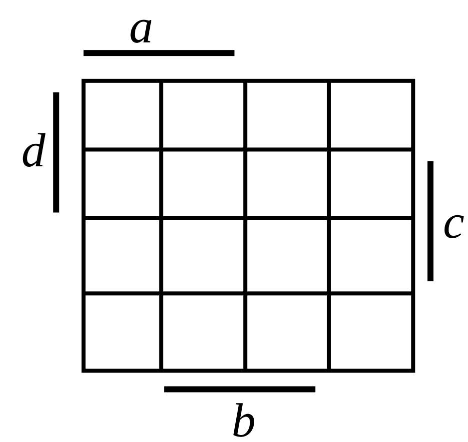


Logic Optimization
Karnaugh Method– K-Map

- K-Map for $n=4$

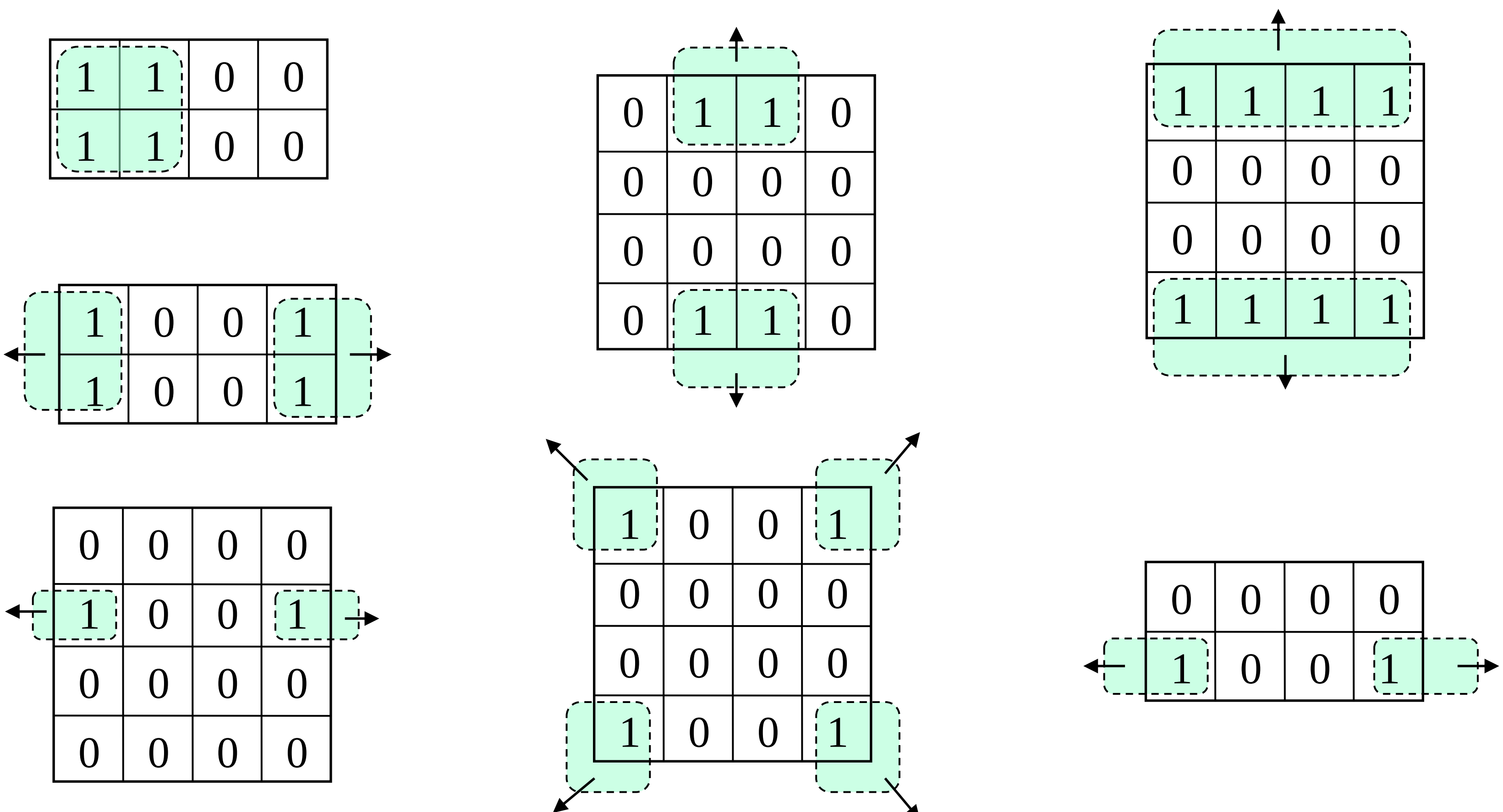
a	b	c	d	z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1





Logic Optimization
Karnaugh Method– K-Map

- Prime Implicant Examples



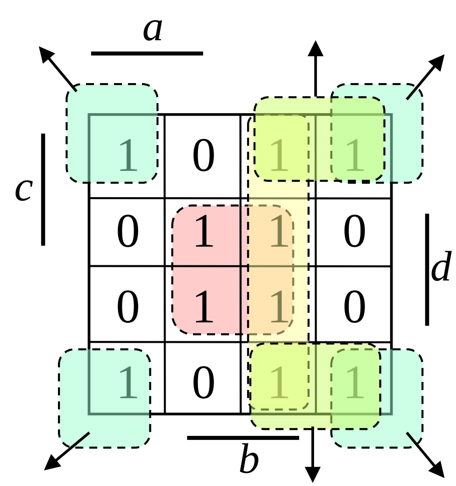
Logic Optimization
Karnaugh Method– K-Map

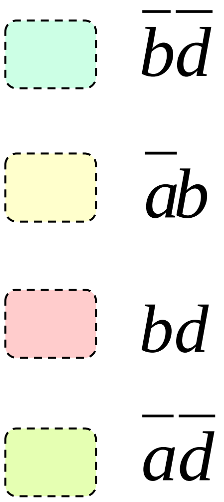
- Minimization with K-Map
  1. Construct K-Map
  2. Identify all prime implicants
     - prime implicants are rectangular blocks of 2^{n-k} 1's cells, $0 \leq k \leq n$, that are not contained in other blocks
     - prime implicants can overlap
  3. Select a minimal number of prime implicants that covers all K-Map 1's
     - In case there are many alternatives, select the one with the minimum number of literals

Logic Optimization
Karnaugh Method– K-Map

- Example

a	b	c	d	z
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1





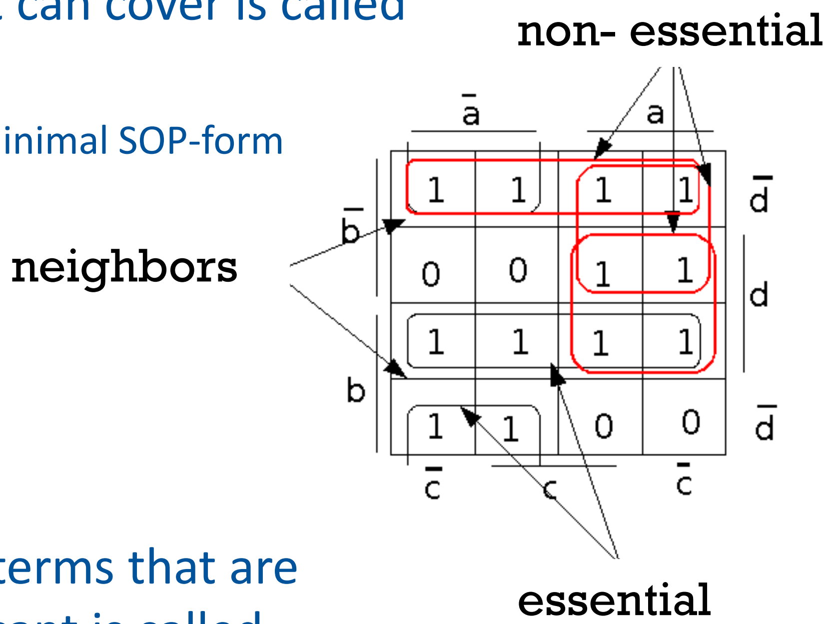
$$z = \bar{a}b + bd + \bar{b}\bar{d}$$

or

$$z = \bar{a}\bar{d} + bd + \bar{b}\bar{d}$$

Logic Optimization
Karnaugh Method– K-Map

- Definition (Essential prime implicant) A prime implicant that covers a minterm that no other implicant can cover is called essential prime implicant
  - Essential prime implicants must be part of any minimal SOP-form



- A prime implicant which only covers minterms that are already covered by essential prime implicant is called absolute non-essential

→ Non essential prime implicants should never be part of a minimal SOP

Logic Optimization
Karnaugh Method– K-Map

- How to select the non-essential prime implicants that will lead to the minimal SOP-Form ?
  - With K-Map, all possible combinations can be explored. The search space is usually not large
  - In realistic cases with more variables, computer-aided methods must be used. Efficient algorithms must de provided for efficient search

Examples

a

	1	1	0	0
b	1	1	0	1
d	1	1	0	1
	1	1	0	1

c

a	b	c	d	z
0	0	0	0	0
0	0	0	1	0
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	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

10-Min Quiz

c

b	1	1	0	0
	1	0	0	1
a	1	0	0	1
	1	1	0	0

d

- Given the K-Map above for a function F
  - List 2 minterms of F
  - List all prime implicants of F
  - Does F has non-essential prime implicants ?
  - Compute the minimal Boolean expression of F
  - Did you purchase the DAD Board?
    - If yes, from which vendor ?

Logic Optimization
Quine-McCluskey Algorithm

- Phase I: identification of Prime implicants
  1. Make a list L_1 of all minterms, sorted in groups G_h, $0 \le h \le n$, where G_h consists of all minterms with h '1's in their code-word
  2. Using list L_i compare all elements E' in G_h with E'' in G_{h+1}. if $E'=Ex$ and $E''=E\overline{x}$, apply the minimization theorem, mark E' and E'', and insert E in the group G_h of a new List L_{i+1}, if not yet included
  3. If L_{i+1} is not empty, set $i = i+1$ and go to step 2. if L_{i+1} is empty, then the non-marked inputs $L_1,\ldots,L_i$ i are prime implicants

Logic Optimization
Quine-McCluskey Algorithm

$$z(a,b,c,d)=\sum(1,3,5,10,11,12,13,14,15)$$

L_1

Group 1	m_1 = 0001	✓
Group 2	m_3 = 0011	✓
	m_5 = 0101	✓
	m_{10} = 1010	✓
	m_{12} = 1100	✓
Group 3	m_{11} = 1011	✓
	m_{13} = 1101	✓
	m_{14} = 1110	✓
Group 4	m_{15} = 1111	✓

L_2

Group 1	00-1	
	0-01	
Group 2	-011	
	-101	
	101-	✓
	11-0	✓
	110-	✓
	1-10	✓
Group 3	1-11	✓
	11-1	✓
	111-	✓

L_3

Group 2	1-1-
	11--

Prime implicants:

$\{P_A, P_B, P_C, P_D, P_E, P_F\}$={00-1, 0-01, -011, -101, 1-1-, 11--}

Logic Optimization
Quine-McCluskey Algorithm

- Phase II: Compute the minimum sum of all prime implicants
    1. Make a coverage table T in which rows represent prime implicants and columns minterms. Mark all cells $T_{i,j}$ in the table with X, if the prime implicant in row i covers minterm of column j
    2. Identify all essential rows of T and insert the corresponding prime implicants to the solution. Reduce T by erasing the essential row and all columns that are covered by the row. Reduce T further by erasing dominant column and dominant rows
    3. Apply step 2 until T cannot be further reduced. If T is empty, then solution is S. If not, go to step 4
    4. Apply distributive law to compute a minimal SOP-form for the remaining rows of T

Logic Optimization
Quine-McCluskey Algorithm

- Coverage table example

	m_1 0001	m_3 0011	m_5 0101	m_{10} 1010	m_{11} 1011	m_{12} 1100	m_{13} 1101	m_{14} 1110	m_{15} 1111	
				✓	✓	✓	✓	✓	✓	
P_A= 00-1	X	X								
P_B= 0-01	X		X							
P_C= -011		X			X					
P_D= -101			X				X			
P_E= 1-1-				X	X			X	X	essential
P_F= 11--						X	X	X	X	essential

$$S = \{P_E, P_F\}$$

Logic Optimization
Quine-McCluskey Algorithm

- Row dominance: Row P_i dominates row P_j, if P_i has X in any column in which P_j also has X
  - Either P_i has more X than P_j , or P_i=P_j
  - A dominant row covers the same or more minterms than the dominated row
  - A dominated row can be removed from the coverage table, if it contains more literals than the dominant row
    - Random choice if P_i=P_j and the two rows have the same number of literals

	X	X	X	X	p_1	c_1
		X		X	p_2	c_2
			...			
	X		X	X	p_k	c_k

	X	X	X	X	p_1	c_1
		~~X~~		~~X~~	~~p_2~~	~~c_2~~
			...			
	X		X	X	p_k	c_k

Logic Optimization
Quine-McCluskey Algorithm

- Colum dominance: Column m_i dominates m_j, if m_i has X in any row in which m_j also has X
  - A dominant column can be erased, since every coverage of m_j is also a coverage of m_i
    chose any to erase, if the two columns are identical

	m_i	m_j	
p1	X	X	
p2			
p3	X	X	
p4	X		
p5	X		

	m_i	m_j	
p1	X	X	
p2			
p3	X	X	
p4	X		
p5	X		

  - p_1 and p_3 cover m_i and m_j : p4 and p5 are redundant in regard to m_i and m_j

Logic Optimization
Quine-McCluskey Algorithm

	m_1 0001	m_3 0011	m_5 0101
P_A= 00−1	X	X	
P_B= 0−01	X		X
P_C= −011		X	
P_D= −101			X

P_A dominates P_C $\rightarrow$ erase P_C
P_B dominates P_D $\rightarrow$ erase P_D

	m_1 0001	m_3 0011	m_5 0101
P_A= 00−1	X	X	
P_B= 0−01	X		X

m_1 dominates m_3, m_5
$\rightarrow$ erase m_1

	m_3 0011	m_5 0101
P_A= 00−1	X	
P_B= 0−01		X

P_A, P_B are essential

$$S = \{P_E, P_F, P_A, P_B\} =$$
$$\{00{-}1, 0{-}01, 1{-}1{-}, 11{-}{-}\} =$$
$$\bar{a}\bar{b}d + \bar{a}\bar{c}d + ac + ab$$

Logic Optimization
Quine-McCluskey Algorithm

- Branching
  - If the coverage table T still has k rows and cannot be further reduced, randomly chose a row P_i and „tentatively“ insert the corresponding prime implicant to the solution S. Erase P_i and all columns covered by P_i. Further reduce T by erasing new resulting essential rows, dominant columns and dominated rows
  - If the coverage table becomes empty, note the tentative solution S, and repeat the process for all remaining k-1 row. The final solution is the one with the lowest cost among the tentative k solutions
  - If the coverage table is not empty, select another P_j and proceed as described above. In worst-case, the algorithm will result in an exponential run-time

UF Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE