EECS150

Section 5
Simplification and State Minimization
Fall 2001

Simplification

- Finding a minimal sum of products or product of sums realization
  - Exploit don't care information in the process
- Algebraic simplification
  - Not an algorithmic/systematic procedure
  - How do you know when the minimum realization has been found?
- Computer-aided design tools
  - Precise solutions require very long computation times, especially for functions with many inputs (> 10)
  - Heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- Hand methods still relevant
  - To understand automatic tools and their strengths and weaknesses
  - Ability to check results (on small examples)

The Uniting Theorem

- Key tool to simplification: A (B' + B) = A
- Essence of simplification of two-level logic
  - Find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

F = A'B'+AB' = (A'+A)B' = B'

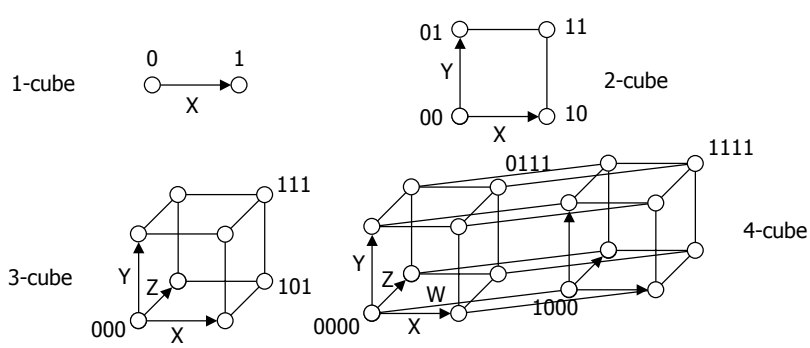
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

B has the same value in both on-set rows
– B remains

A has a different value in the two rows
– A is eliminated

Boolean cubes

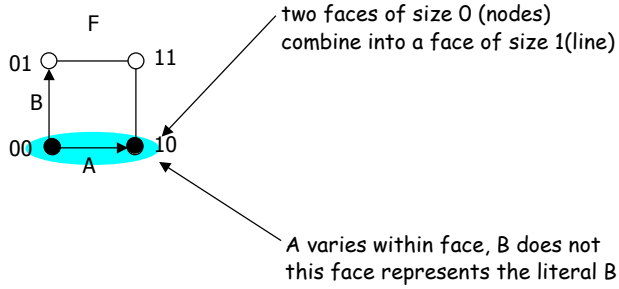
- Visual technique for indentifying when the uniting theorem can be applied
- n input variables = n-dimensional "cube"



Mapping truth tables onto cubes

- Uniting theorem combines two "faces" of a cube into a larger "face"
- Example:

A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

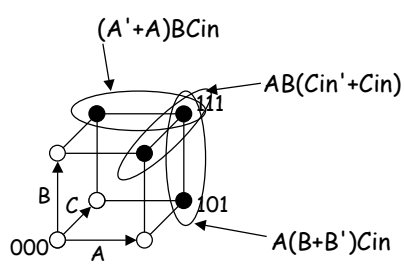


ON-set = solid nodes
OFF-set = empty nodes

Three variable example

- Binary full-adder carry-out logic

A	B	Cin	Cout
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

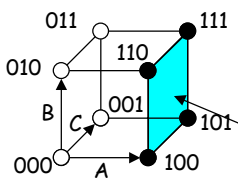


the on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that "111" is covered three times

Cout = BCin+AB+ACin

Higher dimensional cubes

- Sub-cubes of higher dimension than 2



F(A,B,C) = Σm(4,5,6,7)

on-set forms a square
i.e., a cube of dimension 2

represents an expression in one variable
i.e., 3 dimensions – 2 dimensions

A is asserted (true) and unchanged
B and C vary

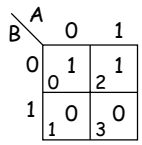
This subcube represents the literal A

m-dimensional cubes

- In a 3-cube (three variables):
  - 0-cube, i.e., a single node, yields a term in 3 literals
  - 1-cube, i.e., a line of two nodes, yields a term in 2 literals
  - 2-cube, i.e., a plane of four nodes, yields a term in 1 literal
  - 3-cube, i.e., a cube of eight nodes, yields a constant term "1"
- In general,
  - m-subcube within an n-cube (m < n) yields a term with n – m literals

Karnaugh maps

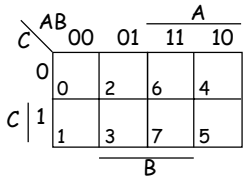
- Flat map of Boolean cube
  - Wrap–around at edges
  - Hard to draw and visualize for more than 4 dimensions
  - Virtually impossible for more than 6 dimensions
- Alternative to truth-tables to help visualize adjacencies
  - Guide to applying the uniting theorem
  - On-set elements with only one variable changing value are adjacent unlike the situation in a linear truth-table

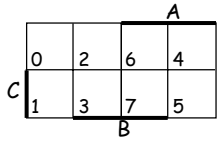


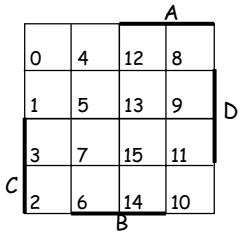
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

Karnaugh maps (cont'd)

- Numbering scheme based on Gray–code
  - e.g., 00, 01, 11, 10
  - Only a single bit changes in code for adjacent map cells



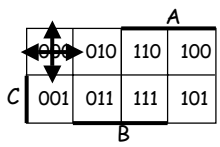


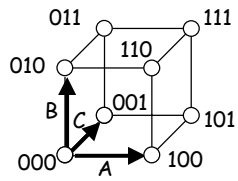


13 = 1101= ABC'D

Adjacencies in Karnaugh maps

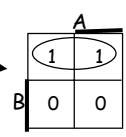
- Wrap from first to last column
- Wrap top row to bottom row



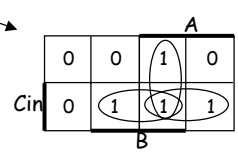


Karnaugh map examples

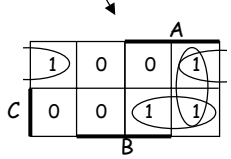
- F =
- Cout =
- f(A,B,C) = Σm(0,4,6,7)



B'



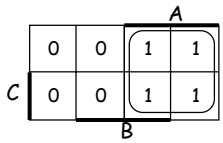
AB + ACin+ BCin



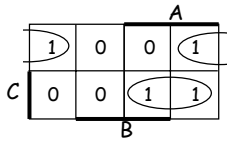
AC+ B'C' + ~~AB'~~

obtain the complement of the function by covering 0s with subcubes

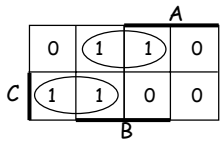
More Karnaugh map examples



$G(A,B,C) = A$



$F(A,B,C) = \Sigma m(0,4,5,7) = AC + B'C'$



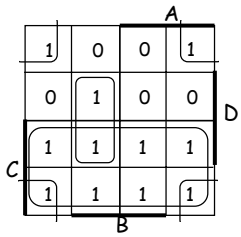
F' simply replace 1's with 0's and vice versa

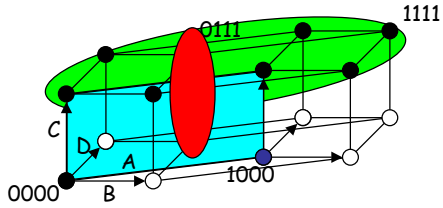
$F'(A,B,C) = \Sigma\, m(1,2,3,6) = BC' + A'C$

Karnaugh map: 4-variable example

- $F(A,B,C,D) = \Sigma m(0,2,3,5,6,7,8,10,11,14,15)$

  $F = C + A'BD + B'D'$

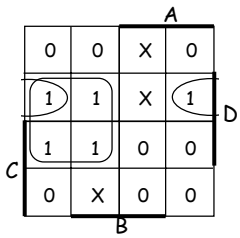




find the smallest number of the largest possible subcubes to cover the ON-set
(fewer terms with fewer inputs per term)

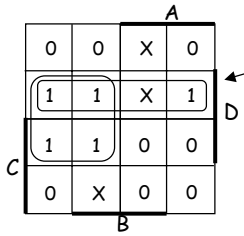
Karnaugh maps: don't cares

- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)
  - without don't cares
    - f = A'D + B'C'D



Karnaugh maps: don't cares

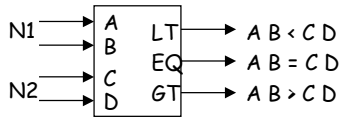
- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)
  - f = A'D + B'C'D — without don't cares
  - f = A'D + C'D — with don't cares



by using don't care as a "1" a 2-cube can be formed rather than a 1-cube to cover this node

<u>don't cares</u> can be treated as 1s or 0s depending on which is more advantageous

Example: two-bit comparator

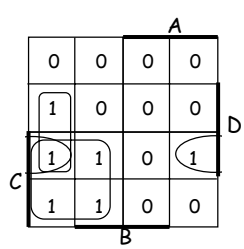


block diagram
and
truth table

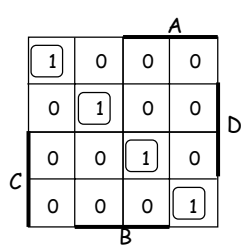
A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

we'll need a 4-variable Karnaugh map
for each of the 3 output functions

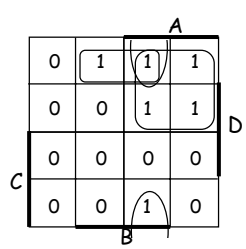
Example: two-bit comparator



K-map for LT



K-map for EQ



K-map for GT

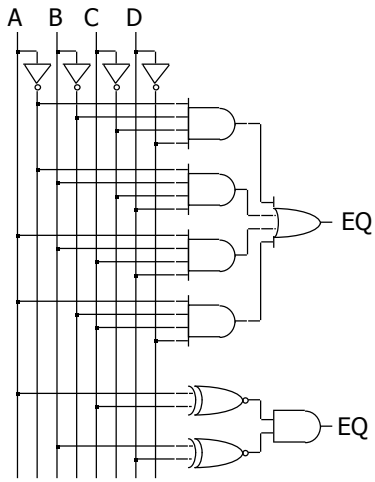
LT = A' B' D + A' C + B' C D

EQ = A'B'C'D' + A'BC'D + ABCD + AB'CD' = (A xnor C) • (B xnor D)

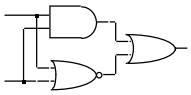
GT = B C' D' + A C' + A B D'

LT and GT are similar (flip A/C and B/D)

Example: two-bit comparator

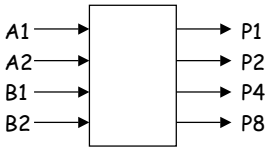


two alternative implementations of EQ with and without XOR



XNOR is implemented with at least 3 simple gates

Example: 2x2-bit multiplier

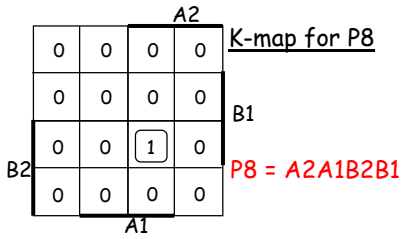


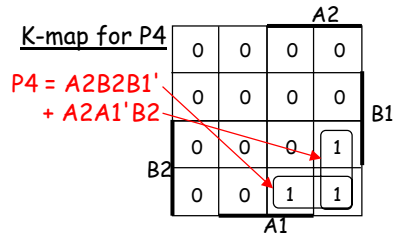
block diagram and truth table

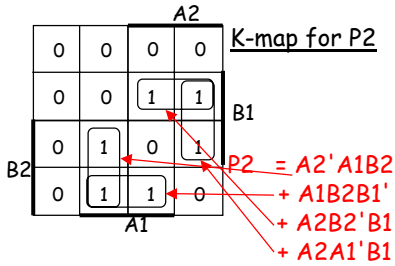
A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

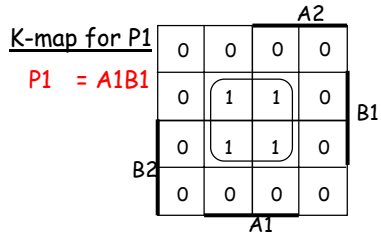
4-variable K-map for each of the 4 output functions

Example: 2x2-bit multiplier

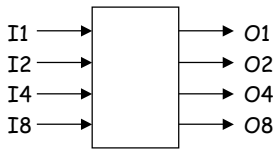








Example: BCD increment by 1

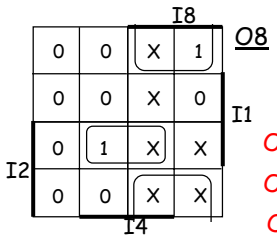


block diagram
and
truth table

I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

4-variable K-map for each of
the 4 output functions

Example: BCD increment by 1

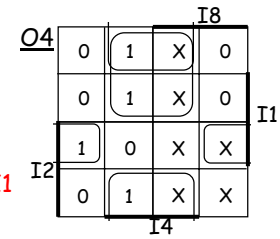


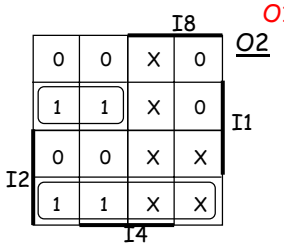
$O8 = I4\ I2\ I1 + I8\ I1'$

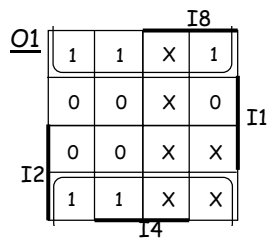
$O4 = I4\ I2' + I4\ I1' + I4'\ I2\ I1$

$O2 = I8'\ I2'\ I1 + I2\ I1'$

$O1 = I1'$



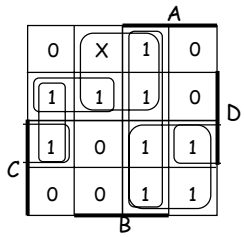




Definition of terms

- Implicant
  - Single element of ON-set or any group of these elements that can be combined to form a subcube
- Prime implicant
  - Implicant that can't be combined with another to form a larger subcube
- Essential prime implicant
  - Prime implicant is essential if it alone covers an element of ON-set
  - Will participate in ALL possible covers of the ON-set
- Objective:
  - Grow implicant into prime implicants (minimize literals per term)
  - Cover the ON-set with as few prime implicants as possible (minimize number of product terms)

Examples to illustrate terms



6 prime implicants:

A'B'D, BC', AC, A'C'D, AB, B'CD

essential

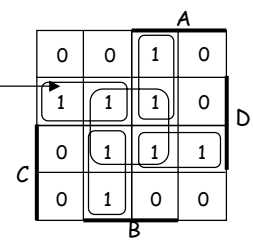
minimum cover: AC + BC' + A'B'D

5 prime implicants:

BD, ABC', ACD, A'BC, A'C'D

essential

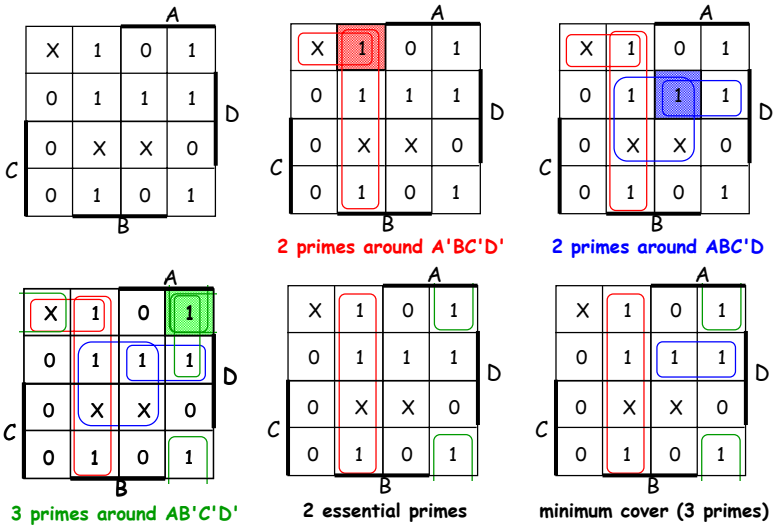
minimum cover: 4 essential implicants



Two-level simplification algorithm

- Algorithm: minimum sum-of-products expression from a Karnaugh map
  - Step 1: choose an element of the ON-set
  - Step 2: find "maximal" groupings of 1s and Xs adjacent to that element
    - consider top/bottom row, left/right column, and corner adjacencies
    - this forms prime implicants (number of elements are power of 2)
  - Repeat Steps 1 and 2 to find all prime implicants
  - Step 3: revisit the 1s in the K-map
    - if covered by single prime implicant, it is essential, and participates in final cover
    - 1s covered by essential prime implicant do not need to be revisited
  - Step 4: if there remain 1s not covered by essential prime implicants
    - select the smallest number of prime implicants that cover the remaining 1s

Example



Finite State Machine Optimization

- State Minimization
  - Fewer states require fewer state bits
  - Fewer bits require fewer logic equations
- Encodings: State, Inputs, Outputs
  - State encoding with fewer bits has fewer equations to implement
    - However, each may be more complex
  - State encoding with more bits (e.g., one-hot) has simpler equations
    - Complexity directly related to complexity of state diagram
  - Input/output encoding may or may not be under designer control

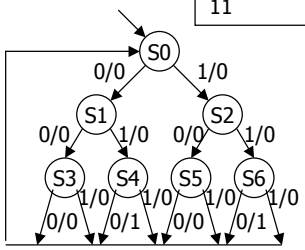
Algorithmic Approach

- Goal – identify and combine states that have equivalent behavior
- Equivalent States:
  - Same output
  - For all input combinations, states transition to same or equivalent states
- Algorithm Sketch
  - 1. Place all states in one set
  - 2. Initially partition set based on output behavior
  - 3. Successively partition resulting subsets based on next state transitions
  - 4. Repeat (3) until no further partitioning is required
    - states left in the same set are equivalent
  - Polynomial time procedure

State Minimization Example

- Sequence Detector for 010 or 110

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0



Method of Successive Partitions

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6)

(S0 S3 S5) (S1 S2) (S4 S6)

(S0) (S3 S5) (S1 S2) (S4 S6)

S1 is equivalent to S2

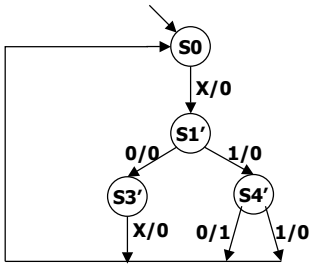
S3 is equivalent to S5

S4 is equivalent to S6

Minimized FSM

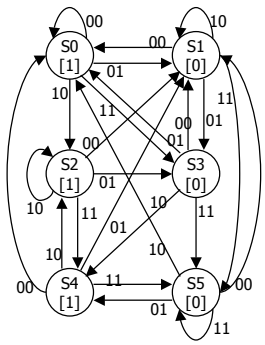
- State minimized sequence detector for 010 or 110

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	**S0**	**S1'**	**S1'**	**0**	**0**
0 + 1	**S1'**	**S3'**	**S4'**	**0**	**0**
X0	**S3'**	**S0**	**S0**	**0**	**0**
X1	**S4'**	**S0**	**S0**	**1**	**0**



More Complex State Minimization

- Multiple input example



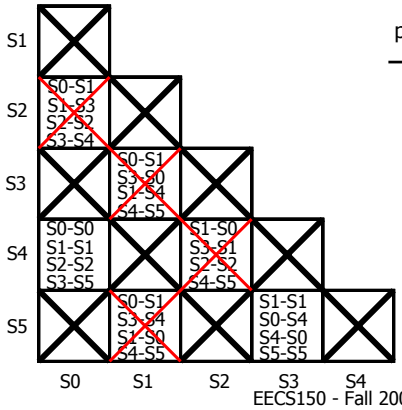
inputs here

present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

symbolic state transition table

Minimized FSM

- Implication Chart Method
  - Cross out incompatible states based on outputs
  - Then cross out more cells if indexed chart entries are already crossed out



present state	next state 00	01	10	11	output
S0'	S0'	S1	S2	S3'	1
S1	S0'	S3'	S1	S3'	0
S2	S1	S3'	S2	S0'	1
S3'	S1	S0'	S0'	S3'	0

minimized state table
(S0==S4) (S3==S5)

Incompletely Specified FSMs

- Equivalence of states is transitive when machine is fully specified
- But its not transitive when don't cares are present

e.g.,	state	output	
	S0	– 0	S1 is compatible with both S0 and S2
	S1	1 –	but S0 and S2 are incompatible
	S2	– 1	

- No polynomial time algorithm exists for determining best grouping of states into equivalent sets that will yield the smallest number of final states