CS225 Switching Theory

S. Tripathy
IIT Patna

Previous Class

Minimization/ Simplification **of Switching Functions**

K-map (SOP)

This Class

Minimization/ Simplification **of Switching Functions**

K-map

Implicant, Prime implicant and Essential PI

Quine-McCluskey (Tabular) Minimization

Example:

Simplification of Two bit adder

Simplification of Full Adder

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Minimization using K-map

Minimal expression: covers all the 1 cells with the smallest number of cubes

such that each cube is as large as possible

- A cube contained in a larger cube must never be selected
- If there is more than one way of covering the map with a minimal number of cubes, select the cover with larger cubes
- A cube contained in any combination of other cubes already selected in the cover is redundant by virtue of the consensus theorem

Rules for minimization:

1. First, cover those 1 cells by cubes that cannot be combined with other 1 cells; continue to 1 cells that have a single adjacent 1 cell (thus can form cubes of only two cells)
2. Next, combine 1 cells that yield cubes of four cells, but are not part of any cube of eight cells, and so on
3. Minimal expression: collection of cubes that are as large and as few in number as possible, such that each 1 cell is covered by at least one cube

Don't-care Combinations

Don't-care combination ϕ : combination for which the value of the function is not specified. Either

- input combinations may be invalid
- precise output value is of no importance

Since each don't-care can be specified as either 0 or 1, a function with k don't-cares corresponds to a class of 2^k distinct functions. Our aim is to choose the function with the minimal representation

- Assign 1 to some don't-cares and 0 to others in order to increase the size of the selected cubes whenever possible
- No cube containing only don't-care cells may be formed, since it is not required that the function equal 1 for these combinations

Code Converter

Example: code converter from BCD to excess-3 code

- Combinations 10 through 15 are don't-cares

Decimal	BCD Inputs				Excess-3 Outputs			
	w	x	y	z	f_4	f_3	f_2	f_1
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$$f_1 = \sum(0,2,4,6,8) + \sum_{\phi}(10,11,12,13,14,15)$$

$$f_2 = \sum(0,3,4,7,8) + \sum_{\phi}(10,11,12,13,14,15)$$

$$f_3 = \sum(1,2,3,4,9) + \sum_{\phi}(10,11,12,13,14,15)$$

$$f_4 = \sum(5,6,7,8,9) + \sum_{\phi}(10,11,12,13,14,15)$$

Code Converter (Contd.)

f_1 Map

yz \ wx	00	01	11	10
00	1	1	ϕ	1
01			ϕ	
11			ϕ	ϕ
10	1	1	ϕ	ϕ

f_2 Map

yz \ wx	00	01	11	10
00	1	1	ϕ	1
01			ϕ	
11	1	1	ϕ	ϕ
10			ϕ	ϕ

f_3 Map

yz \ wx	00	01	11	10
00		1	ϕ	
01	1		ϕ	1
11	1		ϕ	ϕ
10	1		ϕ	ϕ

f_4 Map

yz \ wx	00	01	11	10
00			ϕ	1
01		1	ϕ	1
11		1	ϕ	ϕ
10		1	ϕ	ϕ

$f_1 = z'$

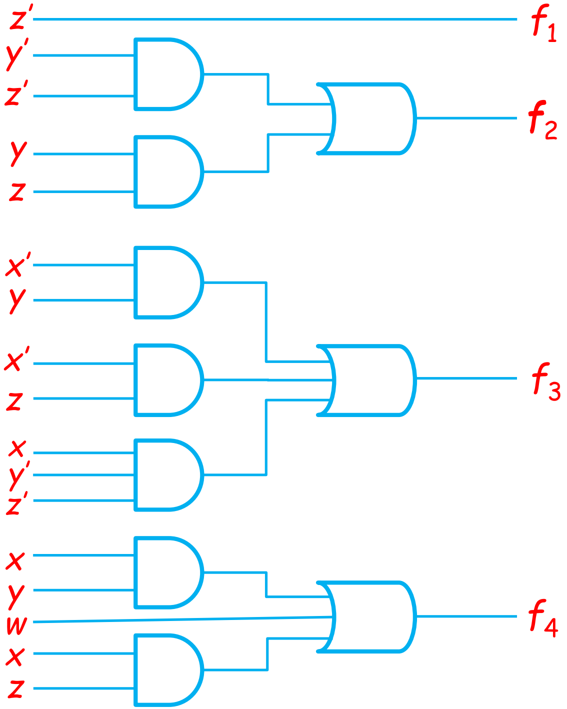
$f_2 = y'z' + yz$

$f_3 = x'y + x'z + xy'z'$

$f_4 = w + xy + xz$

Logic Network for Code Converter

Two-level AND-OR realization:



Five-variable Map

General five-variable map:

vwx yz	000	001	011	010	110	111	101	100
00	0	4	12	8	24	28	20	16
01	1	5	13	9	25	29	21	17
11	3	7	15	11	27	31	23	19
10	2	6	14	10	26	30	22	18

Example: Minimize $f(v, w, x, y, z) = \sum(1, 2, 6, 7, 9, 13, 14, 15, 17, 22, 23, 25, 29, 30, 31)$

vwx yz	000	001	011	010	110	111	101	100
00								
01	1		1	1	1	1		1
11		1	1			1	1	
10	1	1	1			1	1	

$f(v,w,x,y,z) = x'y'z + wxz + xy + v'w'yz'$

Minimal Functions and Their Properties

Implicants: function f covers function g with the same input variables if f has a 1 in every row of the truth table in which g has a 1

- If f covers g and g covers f, then f and g are equivalent
- Let h be a product of literals. If f covers h, then h is said to imply f or h is said to be an implicant of f, denoted as $h \rightarrow f$

Example: If $f = wx + yz$ and $h = wxy'$, then f covers h and h implies f

Prime implicant p of function f: product term covered by f such that the deletion of any literal from p results in a new product not covered by f

- p is a prime implicant if and only if p implies f, but does not imply any product with fewer literals which in turn also implies f

Example: $x'y$ is a prime implicant of $f = x'y + xz + y'z'$ since it is covered by f and neither x' nor y alone implies f

Theorem: Every irredundant sum-of-products equivalent to f is a union of prime implicants of f

Procedure for finding the minimal function via K-maps (layman terms)

1. Convert truth table to K-map
2. Group adjacent ones: In doing so include the largest number of adjacent ones (Prime Implicants)
3. Create new groups to cover all ones in the map: create a new group only to include at least one cell (of value 1) that is not covered by any other group (Essential Prime Impliants)
4. Select the groups that result in the minimal sum of products (we will formalize this because its not straightforward)

Y

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

Reading the reduced K-map

Y

CD \ AB	00	01	11	10
00	0: 1	4: 0	12: 0	8: 1
01	1: 0	5: 1	13: 0	9: 1
11	3: 1	7: 1	15: 0	11: 0
10	2: 1	6: 1	14: 0	10: 1

$\sum m(2,3,6,7)$
$\sum m(5,7)$
$\sum m(8,9)$
$\sum m(0,2,8,10)$

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