Hardware Design I Chap. 3
Minimization of two level logic

Computing Architecture Lab.
Hajime Shimada
E-mail: shimada@is.naist.jp

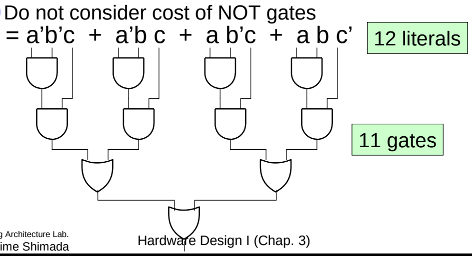
Outline

- Why we have to minimize logic size
  - Relationship between logical expression size and hardware
  - Delay and hardware cost on FET
- Minimizing logical expression method
  - Minimize on Boolean algebra
  - Karnaugh map
  - Quine-McCluskey algorithm

Relationship between logical expression and logical circuit

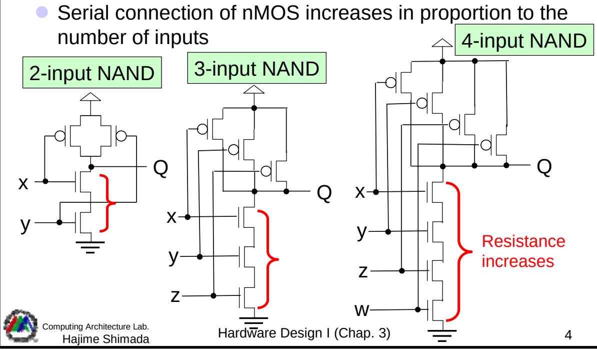
- The number of gates increase in proportion to the number of literals
  - Assuming 2-input gates
  - Do not consider cost of NOT gates

h = a'b'c + a'b c + a b'c + a b c'



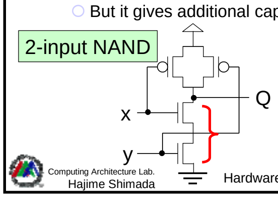
Multiple input CMOS NAND gates (1/2)

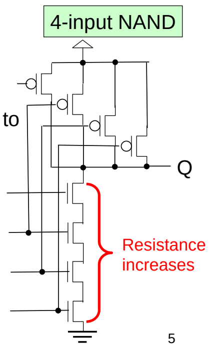
- Serial connection of nMOS increases in proportion to the number of inputs



Multiple input CMOS NAND gates (2/2)

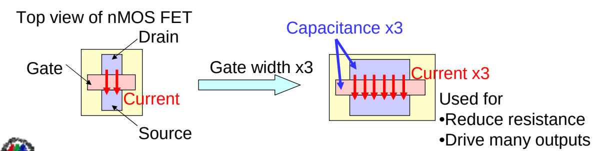
- Discharge speed of 4-input NAND is half of that of 2-input NAND
  - Resistance: x2
  - Current: x1/2
- Usually, we extend gate width of FET to reduce resistance
  - But it gives additional capacitance





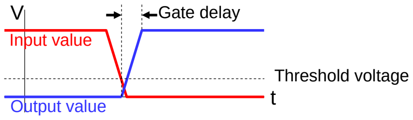
Gate width of FET

- We can choose several gate width of FET
  - We can choose channel width of FET
  - Enlarge channel width = Enlarge current conducting capacity
- But it gives additional capacitance of gate and diffusion area



Operation delay of logic gates

- Voltage-time graph of NOT logic gate



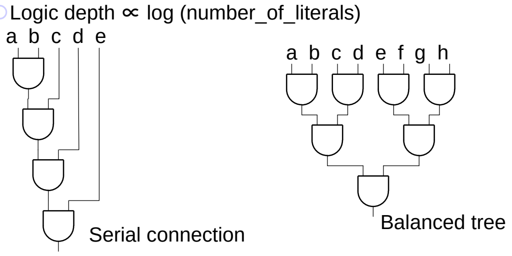
- The gate delay slightly differs between semiconductor process technology
  - Shrinked technology is faster
  - Some techniques to improve transistor: strained silicon, SOI, etc...

Normalized delay of logic gates

- Gate delay differs with several parameters
  - Semiconductor process technology
  - Number of outputs (≒ gate width)
- FO4 inverter delay
  - The delay of the NOT gate which can drive 4 same NOT gate
  - An technology independent delay notation
  - e.g.
    - NOT gate which can drive 8 outputs: 1.2 FO4
    - NAND gate which can drive 4 outputs: 1.5 FO4
    - NAND gate which can drive 8 outputs: 1.7 FO4

Logic depth reduction with balanced tree

- We can reduce logic depth by balanced tree
  - Number of logic gates = (number_of_literals – 1)
  - Logic depth ∝ log (number_of_literals)



Why we have to create minimized circuit?

- To reduce silicon die size
  - Large circuit requires large silicon to place gates
- To create faster circuit
  - Large circuit increases capacitance and resistance which gives long charge/discharge time

Want to create smaller logical circuit

Want to create smaller logical expression

Example of combinational logic design

1. (Write truth table) ← Specification
2. Write logical expression
3. Simplify logical expression

a b cin	cout	sum
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1

cout = a' b cin + a b' cin + a b cin' + a b cin

sum = a' b' cin + a' b cin' + a b' cin' + a b cin

Simplify (Including multi level circuit viewpoint -> see Chap. 8)

cout = (a' b + a b') cin + a b

sum = cin (a' b' + a b) + cin' (a' b + a b')

Outline

- Why we have to minimize logic size
  - Relationship between logical expression size and hardware
  - Delay and hardware cost on FET
- Minimizing logical expression method
  - Minimize on Boolean algebra
  - Karnaugh map
  - Quine-McCluskey algorithm

Minimize on Boolean algebra

- The rule used for simplification
  - Idempotent: a+a = a
  - Distributive: (a+b)・c = a・c+b・c
  - Complements: a+a' = 1
  - Identity: a・1 = a

e.g. f = a'bc + a'bc' + abc' + abc
= a'bc + abc + a'bc' + abc'
= (a'+a)bc + (a'+a)bc' = 1bc + 1bc' =
=1(Complements) Identity
= bc + bc' = b(c+c') = 1b =b

Problems in minimizing on Boolean algebra

- Comparatively hard to determine what rule simplifies the logical expression
- Hard to determine what rule must be applied for final goal

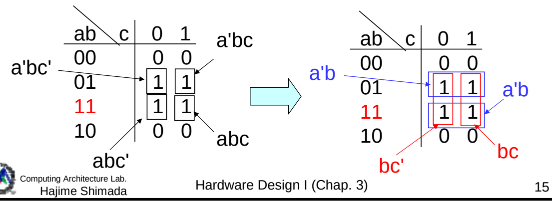
⇩

- Usually, following method is widely used
  - Hand optimizing -> Karnaugh map
  - On EDA -> Quine-McCluskey algorithm

Outline of Karnaugh map (1/3)

- 2-dimension notation of truth table
- Consecutive value can apply complements rule
  - With special order (gray code: 00, 01, 11, 10)
  - Group them with rectangle to apply complements rule

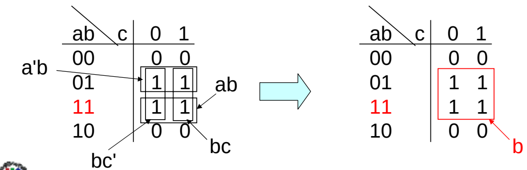
e.g. f = a'bc + a'bc' + abc' + abc = (a'b + a'b) or (bc + bc')



Outline of Karnaugh map (2/3)

- We can extend group size to apply complements rule further
  - The size of group must be 2^n

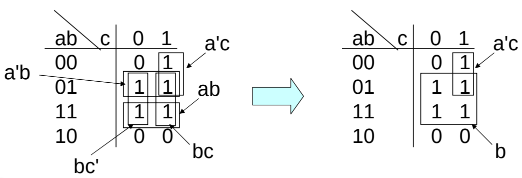
e.g. f = (a'b + ab) or (bc + bc') = b



Outline of Karnaugh map (3/3)

- We can share a minterm between groups

e.g. f = a'bc + a'bc' + abc' + abc + ab'c = b + a'c
  - If we input (a,b,c)=(0,1,1), f = b+ac' = 1+1 = 1
- You have to select least groups



Gray code

- A binary notation which only flips 1-bit in consecutive values
  - Usual binary

    0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000

    1-bit flip 2-bit 1-bit 3-bit 1-bit 2-bit 1-bit 4-bit
  - Gray code

    0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100

    1-bit flip 1-bit 1-bit 1-bit 1-bit 1-bit 1-bit 1-bit
    - Assume mirrored order for lower bits after carry

      0, 1, 11, 10, 110, 111, 101, 100, 1100

4-value Karnaugh map (1/3)

- Please assume following edges are connected
  - Top edge and bottom edge
  - Left edge and right edge

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

Consecutive through edge

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

b'c'd

ab'd'

4-value Karnaugh map (2/3)

- Example of 4-content group through edges
  - Watch out for a group which is consisted with 4 corners
- Consider 8-content group if it is possible

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

a'd'

b'c'

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

b'd'

4-value Karnaugh map (3/3)

- Considering least groups becomes more important

  e.g. f = a'c' + b'c' + cd' + abc

a'c' a'd'

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

abc

b'c' b'd' cd'

a'c' a'd'

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

abc

b'c' b'd' cd'

Prime implicant

- Prime implicant
  - A largest product term which covers "1" area on truth table
- Essential prime implicant
  - The prime implicant that covers some "1" area which has not covered by the other prime implicant
  - Must be chosen in first procedure when you are trying minimization of logical expression

Example of prime implicant

- a'c', b'c' and abc are selected as prime implicant
  - a'bc'd is only covered by a'c'
  - ab'c'd is only covered by b'c'
  - abcd is only covered by abc
- Repeat similar operation
  - But sometimes it becomes set cover problem which is NP-complete problem

a'c' a'd'

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	1	1
10	1	1	0	1

abc

b'c' b'd' cd'

5-value Karnaugh map

- The order of variables is as follows
  - 000, 001, 011, 010, 110, 111, 101, 100
- Assume that values which exist mirrored position from center are consecutive

ab \ cde	000	001	011	010	110	111	101	100
00								
01		1				1		
11		1				1		
10								

Consecutive

a'bc'd'e + a'bcd'e = a'bd'e

bd'e

6-value Karnaugh map

- Assume mirror to the center of vertical position
- The end of Karnaugh map
  - It requires another dimension if we increase values

abc \ def	000	001	011	010	110	111	101	100
000								
001		1					1	
011								
010								
110								
111								
101		1					1	
100								

Consecutive
a'b'cde'f
+ ab'cde'f
= b'cde'f

b'ce'f

Short Exercise

- Show simplified logical expression of following logical function with Karnaugh map

x y z	f(x, y, z)
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

Answer

yz \ x	0	1
00	0	0
01	1	1
11	1	1
10	1	0

z

x'y

f(x, y, z) = x'y + z

xy \ z	0	1
00	0	1
01	1	1
11	0	1
10	0	1

x'y

z

zy \ x	0	1
00	0	0
01	1	0
11	1	1
10	1	1

x'y

z

Outline

- Why we have to minimize logic size
- Karnaugh map
- Quine-McCluskey algorithm

Quine-McCluskey algorithm

- An algorithm to get all prime implicant
  - Nearly brute force method
- Algorithm
  1. Translate minterm which outputs 1 to binary notation
  2. Sort with number of "1" in notation
  3. Create level *(i+1)* from level *i*
     - Select term pair which has only one different bit
     - Move it to level *i+1* with translating differ point to *
     - Add flag to used term
     - Finish if there's no level *i+1*
  4. (Select essential prime implicant)

Example of Quine-McCluskey algorithm (1/3)

- Translate minterm which outputs 1 to binary notation
  - a'b'c' : 000
  - a'bc : 011
  - ab'c : 101
  - abc' : 110
  - abc : 111
- Treat above list as level 1

a b c	f
000	1
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Example of Quine-McCluskey algorithm (2/3)

- Sort with number of "1" in notation
- Create level 2
  - Selected from different number of "1" notation group
  - We can use term redundantly

	Level 1	Level 2
# of "1" is 0	000	
	011 x	*11
# of "1" is 2	101	
	110	
# of "1" is 3	111 x	

Level 1	Level 2
000	
011 x	*11
101 x	1*1
110 x	11*
111 x	

Example of Quine-McCluskey algorithm (3/3)

- Not flagged values are prime implicants
- Check essential prime implicants with table
  - If minterm has only one "1" notation, the prime implicant is essential

Level 1	Level 2
000	
011 x	*11
101 x	1*1
110 x	11*
111 x	

Prime implicants

Minterms \ Prime implicants	*11	1*1	11*	000
000				1
011	1			
101		1		
110			1	
111	1	1	1	

f = bc + ac + ab + a'b'c'

Example of prime implicant cannot be defined

- Sometimes it becomes set cover problem
  - e.g. How to choose prime implicant to cover 0010, 0110, and 1010?
  - See those topic in other lecture

	0*0*	*00*	0**0	*0*0	**10	111*
0000	1	1	1	1		
0001	1	1				
0010			1	1	1	
0100	1		1			
0101	1					
0110			1		1	
1111						1
1110					1	1
1000		1		1		
1001		1				
1010				1	1	

Covered with essential

essential

Short exercise

- Show simplified logical expression of following logical function with Quine-McCluskey algorithm

x y z	f(x, y, z)
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

Answer

	Level 1		Level 2	Level 3
# of "1" is 1	001 x	# of "1" is 1	0*1 x	**1
	010 x		*01 x	
# of "1" is 2	011 x		01*	
	101 x	# of "1" is 2	*11 x	
# of "1" is 3	111 x		1*1 x	

2 prime implicants

	**1	01*
001	1	
010		1
011	1	1
101	1	
111	1	

Both of prime implicants are required

f(x, y, z) = x'y + z

How to minimize large hardware?

- Simplification ability of Karnaugh map and Quine-McCluskey algorithm are limited
  - Karnaugh map can treat until 6 variables
  - Quine-McCluskey algorithm can treat until around 30 values
    - Until around 10 variables in 1970's computer
- How to design large hardware with them?
  - Create module which has smaller inputs -> Chap. 4
  - Create large hardware by combining modules
- In practical use, we use MINI or Espresso -> Chap. 7