Gate-level Minimization

CS207 Lecture 3

James YU

Mar. 4, 2020

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Gate-level minimization

- The complexity of digital logic gates to implement a Boolean function is directly related to the complexity of algebraic expression.
- Gate-level minimization is the design task of finding an optimal gate-level implementation of Boolean functions describing a digital circuit.
  - Difficult by hand for more than few inputs.
  - Typically by computer, need to understand the underlying principle.

The map method

- The map method, first proposed by Veitch and slightly improvised by Karnaugh, provides a simple, straightforward procedure for the simplification of Boolean functions.
  - Called *Karnauph map*.
- The map is a diagram consisting of *square*s. For n variables on a Karnaugh map there are $2n$ numbers of squares.
  - Each square or cell represents one of the minterms.
  - Since any Boolean function can be expressed as a sum of minterms, it is possible to recognize a Boolean function graphically in the map from the area enclosed by those squares whose minterms appear in the function.

Two-variable K-map

- A two-variable system can form four minterms

		B = 0	B = 1
A	0	m_0	m_1
	1	m_2	m_3

		B = 0	B = 1
A	0	$A'B'$	$A'B$
	1	AB'	AB

		B = 0	B = 1
A	0	0	1
	1	1	1

- The two-variable Karnaugh map is a useful way to represent any of the 16 Boolean functions.
  - Example: $$A + B = A(B + B') + B(A + A')$$ $$= AB + AB' + AB + A'B = AB + AB' + A'B$$
  - So the squares corresponding to AB, AB', and $A'B$ are marked with 1.

Three-variable K-map

- Since there are eight minterms for three variables, the map consists of eight cells or squares.
  - Minterms are arranged, not according to the binary sequence, but according to the sequence similar to the gray code.
  - **Between two consecutive rows or columns, only one single variable changes its logic value from `0` to `1` or from `1` to `0`.**

BC

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Three-variable K-map

- To understand the usefulness of the map for simplifying the Boolean functions, we must observe the basic properties of the adjacent squares.
  - Any two adjacent squares in the Karnaugh map differ by only one variable, which is complemented in one square and uncomplemented in one of the adjacent squares.
  - The sum of two minterms can be simplified to a single AND term consisting of less number of literals.
  - $m_1 + m_5 = A'B'C + AB'C = (A' + A)B'C = B'C$

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Three-variable K-map

- Example: Simplify the Boolean function $F = A'BC + A'BC' + AB'C' + AB'C$.

BC

A \ BC	00	01	11	10
0	0	0	1	1
1	1	1	0	0

- The first row: $A'BC + A'BC' = A'B$.
- The second row: $AB'C' + AB'C = AB'$.
- $F = A'B + AB'$.

Three-variable K-map

- Example: Simplify the Boolean function $F = A'BC + AB'C' + ABC + ABC'$.

A \ BC	00	01	11	10
0	0	0	1	0
1	1	0	1	1

- The third column: $A'BC + ABC = BC$.
- The second row: $AB'C' + ABC' = AC'$.
- $F = BC + AC'$.

Three-variable K-map

- Example: Simplify the Boolean function $F = \sum(1, 2, 3, 5, 7)$.

A \ BC	00	01	11	10
0	0	1	1	1
1	0	1	1	0

- $F = C + A'B$.

Three-variable K-map

- Example: Simplify the Boolean function $F = \sum(0, 2, 4, 5, 6)$.

$A \backslash BC$	00	01	11	10
0	1	0	0	1
1	1	1	0	1

- $F = C' + AB'$.

Four-variable K-map

- Similar to the method used for two-variable and three-variable Karnaugh maps, four-variable Karnaugh maps may be constructed with 16 squares consisting of 16 minterms.

$ab \backslash cd$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Four-variable K-map

- Two, four, or eight adjacent squares can be combined to reduce the number of literals in a function.
- The squares of the top and bottom rows as well as leftmost and rightmost columns may be combined.
  - When two adjacent squares are combined, it is called a *pair* and represents a term with three literals.
  - Four adjacent squares, when combined, are called a *quad* and its number of literals is two.
  - If eight adjacent squares are combined, it is called an *octet* and represents a term with one literal.
  - If, in the case all sixteen squares can be combined, the function will be reduced to 1.

Four-variable K-map

- Example: Simplify the Boolean function $F = m_1 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} + m_{15}$.
  - $A'B'C'D + A'BC'D = A'C'D$,
  - $ABC'D' + ABC'D = ABC'$,
- $F = A'C'D + ABC' + ACD + AB'C$.
- This reduced expression is not a unique one.
  - If pairs are formed in different ways, the simplified expression will be different.

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

Four-variable K-map

- Example: Simplify the Boolean function $F = m_1 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} + m_{15}$.
- $F = A'C'D + ABC' + ABD + AB'C$.

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

Four-variable K-map

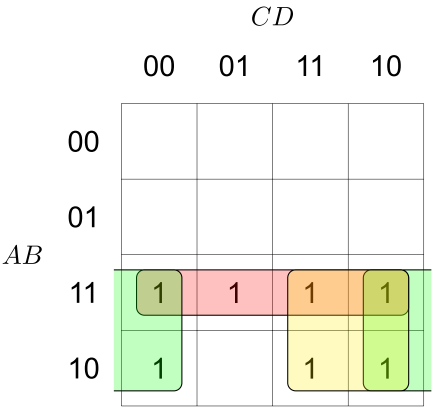
- Example: Simplify the Boolean function $F = \sum(7, 9, 10, 11, 12, 13, 14, 15)$.
- $F = AB + AC + AD + BCD$.

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

Four-variable K-map

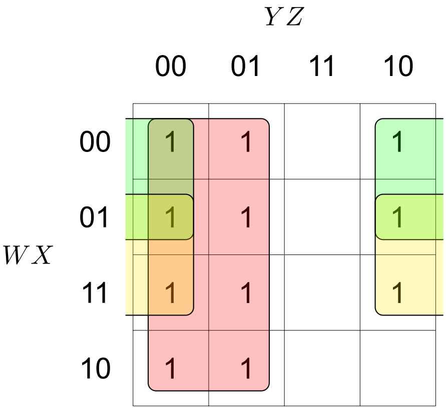
- Example: Plot the logical expression $F(A,B,C,D) = ABCD + AB'C'D' + AB'C + AB$ on a four-variable Karnaugh map.

$$
\begin{aligned}
& F(A,B,C,D) \\
&= ABCD + AB'C'D' + AB'C + AB \\
&= ABCD + AB'C'D' + AB'C(D+D') \\
&\quad + AB(C+C')(D+D') \\
&= \dots \\
&= \sum(8,10,11,12,13,14,15) \\
&= AB + AC + AD'
\end{aligned}
$$



Four-variable K-map

- Simplify the expression $F(W, X, Y, Z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$.



Four-variable K-map

- Simplify the expression $F(W,X,Y,Z) = W'X'Y' + X'YZ' + W'XYZ' + WX'Y'$.

$$
\begin{aligned}
&F(W,X,Y,Z)\\
&= W'X'Y'(Z+Z') + X'YZ'(W+W')\\
&\quad + W'XYZ' + WX'Y'(Z+Z')\\
&= W'X'Y'Z + W'X'Y'Z' + WX'YZ'\\
&\quad + W'X'YZ' + W'XYZ' + WX'Y'Z\\
&\quad + WX'Y'Z'\\
&= \sum(0,1,2,6,8,9,10)\\
&= X'Y' + X'Z' + W'YZ'
\end{aligned}
$$

$WX \backslash YZ$	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

Four-variable K-map

- Simplify the expression $F(W,X,Y,Z) = \sum(3,4,5,7,9,13,14,15)$.
  - It may be noted that one quad can also be formed, but it is redundant as the squares contained by the quad are already covered by the pairs which are essential.
- $F = W'XY' + W'YZ + WY'Z + WXY$.

WX \ YZ	00	01	11	10
00			1	
01	1	1	1	
11		1	1	1
10		1		

Four-variable K-map

- Simplify the expression $F(W, X, Y, Z) = \prod(0, 1, 4, 5, 6, 8, 9, 12, 13, 14)$.
  - The above expression is given in respect to the maxterms.
  - `0`'s are to placed instead of `1`'s at the corresponding maxterm squares.
- $F' = Y' + XZ' \rightarrow F = Y(X' + Z)$.

WX \ YZ	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	0	0	1	0
10	0	0	1	1

Four-variable K-map

- Simplify the expression $F(W, X, Y, Z) = \prod(0, 1, 4, 5, 6, 8, 9, 12, 13, 14)$.
  - The other way to achieve the minimized expression is to consider the 1's of the Karnaugh map.
- $F = YZ + X'Y = Y(X' + Z)$.

WX \ YZ	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	0	0	1	0
10	0	0	1	1

Five-variable K-map

- Karnaugh maps with more than four variables are not simple to use.
  - The number of cells or squares becomes excessively large and combining the adjacent squares becomes complex.
  - A five-variable Karnaugh map contains 2^5 or 32 cells.

Prime Implicants

- A *prime implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map.
- The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares.
  - If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be *essential*.
- Gate-level minimization:
  - Determine all essential prime implicants.
  - Find other prime implicants that cover remaining minterms.
  - Logical sum all prime implicants.

Don’t care conditions

- In practice, Boolean function is not specified for certain combinations of input variables.
  - Input combinations never occur during the process of a normal operation.
  - Those input conditions are guaranteed never to occur.
- Such input combinations are called *don’t-care condition*s.
- These input combinations can be plotted on the Karnaugh map for further simplification.
  - The don’t care conditions are represented by d or X in a K-map.
  - They can be either `1` or `0` upon needed.

Don’t care conditions

- Simplify the expression $F(A, B, C, D) = \sum(1, 3, 7, 11, 15), d = \sum(0, 2, 5)$.
- $F = A'B' + CD$.

$AB \backslash CD$	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

Don’t care conditions

- Simplify the expression $F(A,B,C,D) = \sum(1,3,7,11,15), d = \sum(0,2,5)$.
- $F = A'D + CD$.

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

Verilog user defined primitives

- Verilog has built-in primitives like gates, transmission gates, and switches.
  - This is a rather small number of primitives.
  - If we need more complex primitives, Verilog provides *User Defined Primitives* (UDP).
- UDP begins with reserve word `primitive` and ends with `endprimitive`.
  - Similar to what we do for module definition, ports/terminals of primitive should follow.
- **UDPs should be defined outside `module` and `endmodule`.**

```
primitive udp_syntax (a, b, c, d);
output a;
input b,c,d;

// UDP function code here
endprimitive
```

UDP ports rules

- An UDP can contain only one output and up to 10 inputs.
- Output port should be the first port followed by one or more input ports.
- All UDP ports are scalar, i.e. Vector ports are not allowed.
- UDPs can not have bidirectional ports.
- It is illegal to declare a `reg` for the output terminal of a combinational UDP.
- The output terminal of a sequential UDP requires an additional declaration as type `reg`.

UDP body

- Functionality of primitive is described inside a `table`, and it ends with reserved word `endtable` as shown in the code below.

```
primitive udp_syntax (a, b, c);
output a;
input b,c;

// UDP function code here
// A = B | C;
table
//   B   C     : A
     ?   1     : 1;
     1   ?     : 1;
     0   0     : 0;
endtable
endprimitive
```

UDP body

- Table is used for describing the function of UDP.
- Each line inside a table is one condition.
  - When an input changes, the input condition is matched and the output is evaluated to reflect the new change in input.
  - An UDP cannot use z in the input table.
- UDP uses special symbols to describe functions like rising edge, don't care and so on.

UDP body

Symbol	Interpretation	Explanation
`?`	`0` or `1` or `x`	
`b`	`0` or `1`	
`f`	`(10)`	Falling edge on an input
`r`	`(01)`	Rising edge on an input
`p`	`(01)` or `(0x)` or `(x1)` or `(1z)` or `(z1)`	
`n`	`(10)` or `(1x)` or `(x0)` or `(0z)` or `(z0)`	
`*`	`(??)`	All transitions
`-`	-	No change