Two-level Combinational Circuit Minimization

Z. Jerry Shi
Department of Computer Science and Engineering
University of Connecticut

CSE2300W: Digital Logic Design
Circuit minimization

• All the logic functions can be implemented with two level logic networks
  – Canonical sum and canonical product
  – Become intractable when there are a lot of inputs
• Minimize circuit: reduce the cost to build the circuit while satisfying all the constraints
  – Constraints: time (delay), area, power, etc.
  – Cost: area, time to market, etc.
• The minimization method we will study reduces the cost of two-level AND-OR, OR-AND, NAND-NAND, NOR-NOR circuit
  – Minimizing the number of first-level gates
  – Minimizing the number of inputs on each first-level gates
  – Minimizing the number of inputs on the second-level gate
• We assume the true and complemented forms of input signals are available
Karnaugh Map

Karnaugh map of function of one variable.

\[
\begin{array}{c|c|c}
 x & f(x) & x \\
 0 & f(0) & 0 \\
 1 & f(1) & 1 \\
\end{array}
\]

\[(a)\] 

\[(b)\] 

\[
\begin{array}{c|c|c}
 X & Y & Z \\
 00 & 01 & 11 & 10 \\
 0 & 2 & 6 & 4 \\
 1 & 3 & 7 & 5 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 W & X & Y \\
 00 & 01 & 11 & 10 \\
 0 & 4 & 12 & 8 \\
 1 & 5 & 13 & 9 \\
 3 & 7 & 15 & 11 \\
 2 & 6 & 14 & 10 \\
\end{array}
\]
Visualizing T10 -- Karnaugh maps (1)

T10: \( X \cdot Y + X \cdot Y' = X \)

Adjacent cells can be merged (reduced)

Minterm 5: \( W' \cdot X \cdot Y' \cdot Z \)
Minterm 13: \( W \cdot X \cdot Y' \cdot Z \)

\( X \cdot Y' \cdot Z \)
Visualizing T10 -- Karnaugh maps (2)

T10: \[ X \cdot Y + X \cdot Y' = X \]

Minterm 1: \[ W' \cdot X' \cdot Y' \cdot Z \]
Minterm 9: \[ W \cdot X' \cdot Y' \cdot Z \]

\[ X' \cdot Y' \cdot Z \]
Visualizing T10 -- Karnaugh maps (3)

Minterm 5 and Minterm 13:
\[ X \cdot Y' \cdot Z \]

Minterm 1 and Minterm 9:
\[ X' \cdot Y' \cdot Z \]
\[ Y' \cdot Z \]

Each mergence removes one literal
\[ 2^i \text{ cells} \rightarrow (n - i) \text{ literals} \]

Corresponding product terms:
- covers only 1: variables
- covers only 0: complement of variables
- covers both 0 and 1: not included
Merging cells
Example: $F = \Sigma(1,2,5,7)$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Karnaugh-map usage

- Plot 1’s corresponding to minterms of function
- Circle largest possible rectangular sets of 1’s
  - Number of 1’s in a set must be power of 2
  - OK to cross edges
- Read off product terms, one per circled set
  - Variable is 1 ==> include variable
  - Variable is 0 ==> include complement of variable
  - Variable is both 0 and 1 ==> variable not included
- Circled sets and corresponding product terms are called “implicants”
  - The largest circles are called “prime implicants”
- A minimal sum of a logic function F is a sum-of-products expression for F such that no sum-of-products expression for F has fewer product terms, and any sum-of-products expression with the same number of product terms has at least as many literals.
More concepts

- **Imply**: Logic expression $F_1$ implies $F_2$ if any assignment of values to the variables involved makes $F_1 = 1$, it also makes $F_2 = 1$.
  - $F_1 = X' \cdot Z + Y' \cdot Z$ implies $F_2 = X' \cdot Y + Y' \cdot Z$
- A term $T_1$ **subsumes** $T_2$ if any literal that occurs in $T_2$ also appears in $T_1$
  - $W \cdot X' \cdot Y \cdot Z$ subsumes $X' \cdot Z$
  - $X + Y$ subsumes $X$
- A product term is said to be an **implicant** of a logic function if it implies the function
  - In SOP, any product is an implicant
- An implicant is said to be a **prime implicant** if the implicant does not subsume any other implicant with fewer literals of that function
- Example: $F(X, Y, Z) = X' \cdot Y + Z$.
  - Both $X' \cdot Y$ and $X \cdot Z$ are implicant of $F$.
  - However, $X \cdot Z$ subsumes $Z$. So $X \cdot Z$ is not a prime implicant.
Prime Implicant Theorem

A minimal sum is a sum of prime implicants.

Proof: (By contradiction). Think about this like a logician. Suppose a product term \( P \) in a minimal sum is not a prime implicant. According to the definition of prime implicant, if \( P \) is not a prime implicant, it is possible to remove some literals from \( P \) to obtain a new product term \( P^* \) that still implies \( F \). If we replace \( P \) with \( P^* \) in the minimal sum, the resulting sum still equals to \( F \) but has fewer literals. Therefore, the presumed minimal sum is not minimal.

- The sum of all the prime implicants of a logic function is called the **complete sum**
  - The complete sum is not necessarily a minimal sum.
Which prime implicants to choose?

- **Distinguished 1-cell** of a logic function is a 1 cell that is covered by *only one prime implicant*

- **An essential prime implicant** of a logic function is a prime implicant that covers one or more distinguished 1-cells

- Choose all the essential prime implicants first

- What if some 1’s are not covered?
Example:

\[
\begin{array}{c|cccc}
AB & 00 & 01 & 11 & 10 \\
\hline
CD & \textcolor{red}{00} & \textcolor{green}{01} & \textcolor{blue}{11} & \textcolor{yellow}{10} \\
\end{array}
\]

\[
F = A' \cdot C' + A' \cdot D + A' \cdot B' + B \cdot C \cdot D + A \cdot B \cdot C
\]
Secondary essential prime implicants

- Removed the 1-cells covered by essential prime implicants
- Remove some prime implicants
  - Given two prime implicants P and Q, P is said to eclipse Q (written \( P \sqsubset Q \)) if P covers at least all 1-cells covered by Q
  - A prime implicant Q is removed if there exists P
    - P costs no more than Q
    - \( P \sqsubset Q \) (i.e., P covers all 1 cells covered by Q)
- After removing some prime implicants, some 1-cells are only covered by one implicant. Those implicants are called secondary essential prime implicants.
  - Secondary essential prime implicants must be included in the minimal sum (assuming we can find such prime implicants)
Example: Remove essential prime implicants

\[ F = A' \cdot C' + A' \cdot D + A' \cdot B' + B \cdot C \cdot D + A \cdot B \cdot C \]
Example: Remove more prime implicants

\[ F = A' \cdot D + B \cdot C \cdot D \]

\( (A' \cdot D) \ldots (B \cdot C \cdot D) \)

(B \cdot C \cdot D) is more expensive

Remove (B \cdot C \cdot D) !!!
Example: Include secondary essential prime implicants

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td><em>1</em></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
F = \overline{A} \cdot \overline{C} + \overline{A} \cdot \overline{D} + \overline{A} \cdot \overline{B} + B \cdot C \cdot D + A \cdot B \cdot C
\]
Example 2:

Include essential prime implicants (blue circles)

Remove 1 cells covered by essential prime implicant
Example 2 (continued)

Remove red prime implicants

Can you remove the green prime implicant?
What if no essential prime implicants can be found?

- Trial and error
- Branching method
Two minimal sums

Both equations have the same cost.

Not always true.
Prime-number detector (again)

\[ F = \sum_{N3, N2, N1, N0}(1, 2, 3, 5, 7, 11, 13) \]

\[ F = N_{3}' \cdot N_0 + N_{3}' \cdot N_{2}' \cdot N_1 + N_{2}' \cdot N_1 \cdot N_0 + N_2 \cdot N_{1}' \cdot N_0 \]
Resulting circuit

- When we solved algebraically, we missed one simplification -- the circuit below has three less gate inputs.
Another example

\[ F = \Sigma_{W,X,Y,Z}(5,7,12,13,14,15) \]

\[ F = X \cdot Z + W \cdot X \]
Yet another example

F = Σ_{W,X,Y,Z}(1,3,4,5,9,11,12,13,14,15)

F = X \cdot Y' + X' \cdot Z + W \cdot X
Yet another example

\[ F = \Sigma_{W,X,Y,Z}(2,3,4,5,6,7,11,13,15) \]

\[ F = W' \cdot Y + W' \cdot X + X \cdot Z + Y \cdot Z \]
Steps in K-map reduction

1. Merge 1 cells to get the largest circles (identify prime implicants)
2. Identify distinguished 1 cells
   • Find essential prime implicants
     • Secondary, tertiary, …
     • If no essential prime implicants can be found → Trial and error
3. Include essential prime implicants in the minimal sum
4. Are all 1 cells covered?
   • If yes, you are done.
5. Remove all 1 cells already covered
6. Remove some prime implicants
   • Remove Q if
     • P covers all the 1 cells covered by Q, i.e., (P … Q)
     • Cost(P) ≤ Cost (Q)
7. Goto step 2
K-map with 5 variables

\[ \begin{array}{cccc}
0 & 4 & 12 & 8 \\
1 & 5 & 13 & 9 \\
3 & 7 & 15 & 11 \\
2 & 6 & 14 & 10 \\
\end{array} \]

\[ \begin{array}{cccc}
16 & 20 & 28 & 24 \\
17 & 21 & 29 & 25 \\
19 & 23 & 31 & 27 \\
18 & 22 & 30 & 26 \\
\end{array} \]
Example of 5-variable K-map

\[ D1 = Q1 + Q2' \cdot Q3' \]
\[ D2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A + Q1 \cdot Q2 \cdot B \]
\[ D3 = Q1 \cdot A + Q2' \cdot Q3' \cdot A \]
K-map with 6 variables
Simplifying Products of Sum

F = (A' + B') \cdot (C' + D') \cdot (B + D')

F' = A \cdot B + C \cdot D + B' \cdot D

F = B' \cdot D' + A' \cdot D' + A' \cdot B \cdot C'
Don’t care items

- **Don’t cares**: the output does not matter for these input combinations, or they never appear as valid input

\[ F = \Sigma(4, 12, 13, 14, 15) + d(0, 5, 8) \]

**Minimal Sum:**

\[ F = A \cdot B + C' \cdot D' \]

or

\[ F = A \cdot B + C' \cdot B \]

**Minimal Product:**

\[ F = B \cdot (A + C') \]
Differences with don’t care items

• When identifying prime implicants, consider x as 1 if doing so gives larger circles
  – Each prime implicant covers at least one 1

• When identifying essential prime implicants …
  – The definition of distinguished 1 cell is the same
  – X’s do not make a prime implicant essential
Another example: BCD Prime

\[ F = \Sigma(2, 3, 5, 7) + d(10, 11, 12, 13, 14, 15) = B \cdot D + B' \cdot C \]
Real-World Logic Design

• Lots have more than 6 inputs → can’t use Karnaugh maps
• Design correctness more important than gate minimization
  – Use “higher-level language” to specify logic operations
• Use programs to manipulate logic expressions and minimize logic
• PALASM, ABEL, CUPL -- developed for PLDs
• VHDL, Verilog -- developed for ASICs
Quine-McCluskey algorithm

• This process can be made into a program, using appropriate algorithms and data structures.
  – Guaranteed to find “minimal” solution
• Required computation has exponential complexity (run time and storage)-- works well for functions with up to 8-12 variables, but quickly blows up for larger problems.
  – Min-cover problem is NP-hard
• Heuristic programs (e.g., Espresso) used for larger problems, usually give minimal results.
Multiple-level circuit

- Additional cost savings can be obtained by using multiple-level circuit optimization.
  - $G = A \cdot B \cdot C + A \cdot B \cdot D + E + A \cdot C \cdot F + A \cdot D \cdot F$ (17)
  - $G = A \cdot B \cdot (C + D) + E + A \cdot (C + D) \cdot F$ (13)
  - $G = (A \cdot B + A \cdot F) \cdot (C + D) + E$ (10)
  - $G = A \cdot (B + F) \cdot (C + D) + E$ (9)
- An algorithm corresponding to K-Map for two-level circuit optimization that gives optimum circuit cost does not exist
- Common techniques:
  - Factoring Find the factored form
  - Decomposition Express a function as a set of new functions
  - Extraction Express a set of functions as a set of new functions
  - Substitution Substituting $G$ into a function $F$: express $F$ as a function of $G$ and some or all variables in $F$.
  - Elimination Inverse of substitution. Replace $G$ with its expression in function $F$. Also called flattening or collapsing.
- Algorithms: Boolean networks, factored form
Timing hazards

• We have studied steady-state behavior
  – Assuming inputs have been stable for a long time

• The transient behavior of circuits may differ
  – A circuit’s output may produce a pulse, called a glitch

• A hazard is said to exist when a circuit has a possibility of producing a glitch
Static and dynamic hazards

- **Static-1**: circuit may produce a 0 glitch when we expect a steady 1 at the output
  
  A pair of inputs combinations
  
  both give a 1 output
  
  differ in only one input variable
  
  May produce a momentary 0 output when the transition takes place

- **Static-0**: circuit may produce a 1 glitch when we expect a steady 0 at the output
  
  – Properly designed AND-OR circuits do not have static-0 hazards

- **A dynamic hazard** is the possibility of an output changing more than once as the result of a single input transition
A circuit with a static-1 hazard

Inputs X, Y, Z change from 111 to 110
Finding static hazards using K-map

Suppose X and Y are 1 and Z change from 1 to 0.

\[ F = X \cdot Z' + Y \cdot Z \]

<table>
<thead>
<tr>
<th>Y \cdot Z</th>
<th>X \cdot Z'</th>
<th>X \cdot Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

On K-map, any adjacent 1’s need to be covered by a circle.
Circuit with static-1 hazard eliminated