Synthesis of Two Level Circuits

Sungho Kang

Yonsei University
Outline

- Sums of Products and Products of Sums
- Implicants and Prime Implicants
- Iterated Consensus
- Recursive Computation of Prime Implicants
- Selecting a Subset of Primes
- Unate Covering Problem
- Branch-and-Bound Algorithm
- Multiple Output Functions
Two Level Logic

- Implementation in two levels
  - Speed
  - Simplicity

- The number of gates and the number of gate inputs are the measure of the cost of a two level implementation
Sums of Products

• A letter is a constant or a variable
• A literal is a letter or its complement
• A product term is a formula of one of the following forms
  - 1
  - a non-constant literal
  - a conjunction of non-constant literals where no letter appears more than once
• A sum term is a formula of one of the following forms
  - 0
  - a non-constant literal
  - a disjunction of non-constant literals where no letter appears more than once
An SOP (sum of products) is a join of meets, henceforth called OR of ANDs:

\[ f = xy'z + z'y + wxyz \]

Conjunctions (ANDs) of literals are functions. They are also called cubes, terms, and implicants.

An SOP is optimum, if no other SOP representing the same function has fewer cubes or literals.

Disjunctive Normal Form (DNF) - SOP
Conjunctive Normal Form (CNF) - POS
Implicants with Don’t Cares

Discriminant 1 (ON-set)
• Discriminant don’t care
  Discriminant 0 (OFF-set)

\[ f = x'y'z + x'y + wxyz, \quad d = w'xyz \]

Cubes \( x'y \) and \( wyz \) are **implicants** of \( f \) :

\[ (c = x'y = 1) \Rightarrow (x = 0, y = 1) \]

\[ \Rightarrow f = 0 + 1 + 0 + 0 = 1 \]
Implicants with Don’t Cares

- Discriminant 1 (ON-set)
- Discriminant don’t care
- Discriminant 0 (OFF-set)

Cube $c = xyz$ is an implicant of $f + d$
K-maps and Boolean 4-cubes

Note these minterms are “Distance-2”

\[ B_4 = \{0,1\}^4 \]

Note: Each vertex is a minterm of \( F_4(\{0,1\}) \)

\[ f = wx(y'z + yz') = wxy'z + wxyz' \]
Prime Implicants

- An implicant $p$ of $f$ is a **prime implicant** if and only if no other implicant of $f$ contains $p$.

- If so, the cube formed from $p$ by deleting any literal of $f$ must intersect the OFF-set.

- An **optimal SOP** can be formed from only prime implicants.
Cube $xy'$ is a prime implicant

Cube $xy'$ is not an implicant. It contains offset minterm $wx'y'z'$, and similarly for cube $y$.

$F_{\text{min}} = xz + xy'$
Prime Implicants Can Be Redundant

\[ f = xy'z + x'y + wxyz, \quad d = w'xyz \]

- \( xz, xy' yz \) are all **prime implicants**
- \( yz \) is **redundant**
Another canonical representation of a function $f$ is the sum of all its prime implicants, called the complete sum.

For $f = ab + bc'$, the complete sum is $f = ab + bc' + ac$.

Note that the consensus term $ac$ is also a prime implicant of $f$.

The complete sum is unique, but usually sub-optimal.
The Complete Sum

- The complete sum is $O(3^n/n)$, where $n$ is the number of input variables.
- The minterm canonical form is only $O(2^n)$.
- Nevertheless, the complete sum is important because of the role it plays in logic minimization.
Complete Sum with Don’t Cares

$$f = xy'z + x'y + wxyz, \quad d = w'xyz$$

$$F = CS(f + d) = x'y + xz + yz$$

$CS(f+d)$ denotes the complete sum of $f+d$
Complete Sum: 4-Cube Vs. K-Map

Prime Implicants

1 = xy
2 = xz
3 = yz

Note Cube 1 (xy') has x=0, and y=1
Complete Sum: 4-Cube Vs. K-Map

K-Map

4-Cube

\[ F = xy'z + x'y + wxyz + w'xyz \]

\[ C = yz \]
Cube $p$ is an essential prime implicant of $f$ if it contains a minterm **not contained** by any other prime implicant $p'$ of $f$.

All essential prime implicants are present in **every** optimal SOP.
Quine’s Prime Implicant Theorem

A minimal SOP must always consist of a sum of prime implicants if any definition of cost is used in which the addition of a single literal to any formula increases the cost of the formula.
• Only $xz, xy'$ are essential ( $xz$ exclusively contains minterms $wx'yz'$ and $wxyz'$ of $f$ )
Essential Prime Implicants

Prime Implicants

function = xy'z + x'y + wx'y + w'xyz + wxyz
essential primes = x'y + xz
Essential Primes: K-Map

The selected function/file is:
\[ a'b'c'd'e' + a'b'c'd'e'f' + a'b'c'd'e'f + b'c'd'e' + acdf + ac'de + bcdf' + ab'c'd'e'f' \]

essential primes = 1 + 2 = \[ a'c'd'e' + bcdf' \]
Tabular Method

Iterated Consensus

- Express the function in minterm canonical form

Method

- Compare adjacent pairs
- If only one letter is different, a new term is added where the letter is replaced by a don’t care
- All terms that are used to form new terms are marked

- These are repeated until no more new terms are added
- All unmarked terms are prime implicants
Iterated Consensus

- \( Xy + Xy' = X \) : distance-1 merging

- A complete sum as a SOP formula composed of all the prime implicants of the function it represents

- A SOP formula is a complete sum if and only if
  - No term includes any other term
  - The consensus of any two terms of the formula either does not exist or is contained in some term of the formula
Computing the Prime Implicants

Theorem
If $F_1$ and $F_2$ are complete sums, then $\text{ABS}(F_1 F_2)$ is the complete sum for the function $F_1 F_2$

The function $\text{ABS}$ converts the product to a sum and removes cubes contained in other cubes

$$\text{ABS}((wy+xy)(y+z)) = \text{ABS}(wy+wyz+xy+xyz) = wy+xy$$

$$\text{CS}(f) = \text{ABS}([x_1+\text{CS}(f(0,x_2,\ldots,x_n))][x_1+\text{CS}(f(1,x_2,\ldots,x_n))])$$
Given an SOP $F$, $\text{ABS}(F)$ is the SOP derived from $F$ by deleting any cubes of $F$ whose minterms are contained in those of another cube of $F$.

\[ C_1 = abc', \quad \text{Lits}(C_1) = \{a, b, c'\}, \]
\[ C_2 = bc', \quad \text{Lits}(C_2) = \{b, c'\} \]
\[ (C_1 \subseteq C_2) \iff (\text{Lits}(C_2) \subseteq \text{Lits}(C_1)) \]
\[ \iff (\text{Minterms}(C_1) \subseteq \text{Minterms}(C_2)) \]

\[ \text{ABS}(x_1x_3 + x_1x_3x_4 + x_2x_3 + x_2x_3x_4) = x_1x_3 + x_2x_3 \]
**Picking a Subset of the Primes**

\[ F = yz + x'y + y'z' + xyz + x'z' \]

\[ \text{MCF}(F) = x'y'z' + x'yz' + x'yz + xyz + xy'z' \]

\[ \text{CS}(F) = x'y + x'z' + y'z' + yz \]

\[ F_{\text{Min}} = p_3p_4(p_1 + p_2) \]
The Trouble with Enumeration

- POS to SOP conversion, like many enumeration algorithms, is in class NP (up to $2^n$ minterms)

- Thus either it has no efficient (sub-exponential) implementation, or a host of other exhaustively studied problems, like the traveling salesman problem, also have efficient solutions

- So if you find a linear or quadratic algorithm for any of these problems, you’ll be rich and famous
Picking a Subset of the Primes

\[ F = yz + x'y + y'z' + xy'z + x'z' \]

\[ \text{MCF}(F) = x'y'z' + x'y'z + x'y'z + xy'z + xyz \]

\[ \text{CS}(F) = x'y + x'z' + y'z' + yz + xy' + xz \]

Cyclic!

\[ f=1 \]

No essentials, all primes redundant

\[
\begin{array}{cccccccc}
 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_4 \\
 x'y'z' & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
x'y'z' & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
x'y'z' & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
xy'z' & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
xy'z' & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
xyz & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]
Generalization

- Logic Minimization is just one of many important problems that can be formulated as a matrix covering problem
- Others include Technology Mapping, FSM State Minimization with DCs, and many others from various fields
- So we study this problem intensively
Unate SOPs

An SOP in which each variable appears only as positive literals or only as negative literals is called **Unate**

Unate: \( wy + wxyz + xy + xyz = wy + xy \)

Non-Unate: \( xz + xy' \) (Note possible consensus)

If \( F \) is a unate SOP, \( \text{ABS}(F) \) is the complete sum of the function represented by \( F \), and **all the primes are essential**
A non-unate SOP \textit{binate}

The problem that is obtained by relaxing the assumption that the constraint equation is unate is called \textit{binate covering problem}
Optimal Nutrition:

A good diet should contain adequate amounts of proteins (P), vitamins (V), fats (F), and cookies (C). An astronaut, who must travel light, can choose from five different preparations:

- Preparation 1 (logic variable \( p_1 \)) contains: V and P;
- Preparation 2 (logic variable \( p_2 \)) contains: V and F;
- Preparation 3 (logic variable \( p_3 \)) contains: P and F;
- Preparation 4 (logic variable \( p_4 \)) contains: V;
- Preparation 5 (logic variable \( p_5 \)) contains: C.

Can the astronaut have a balanced diet with only two preparations?

Proteins: \((p_1+p_3)\),  Vitamins: \((p_1+p_2+p_4)\),  .....

The Astronaut and the Cookies
The Astronaut and the Cookies

Brute Force Enumeration Approach:
- Encode candidates with binary variables $p_i$ (like primes)
- Express constraints in POS form (like minterm coverage)
- Convert POS to SOP (enumerate all possibilities)
- Pick “smallest” cube (fewest literals)

\[ 1 = (\text{Proteins})(\text{Vitamins})(\text{Fats})(\text{Cookies}) = (p_1 + p_3)(p_1 + p_2 + p_4)(p_2 + p_3)(p_5) = (p_1 + p_3(p_2 + p_4))(p_2 + p_3)(p_5) = (p_1)(p_2 + p_3)(p_5) + p_3(p_2 + p_4)(p_5) = p_1p_2p_5 + p_1p_3p_5 + p_2p_3p_5 + p_3p_4p_5 \]

All possible solutions!
Reductions: Essential Columns

- Variables with singleton constraints, like $P_5$ in the cookies covering problem, are essential and are present in every optimum solution.

- If Row $i$ of $M$ contains a single nonzero in Col $j$, then add $j$ to the partial solution and delete all rows of $M$ with a nonzero in Col $j$.

- Col $j$, plus the optimum solution to the reduced matrix, is an optimum solution of the original problem.
Essential Columns: Example

\[ 1 = (p_1 + p_3)(p_1 + p_2 + p_4)(p_2 + p_3)(p_5) \]

\[
\begin{array}{c|cccc}
R_1 & p_1 & p_2 & p_3 & p_4 & p_5 \\
M = R_2 & 1 & 0 & 1 & 0 & 0 \\
R_3 & 0 & 1 & 1 & 0 & 0 \\
R_4 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
R_1 & p_1 & p_2 & p_3 & p_4 \\
M' = R_2 & 1 & 0 & 1 & 0 \\
R_2 & 1 & 1 & 0 & 1 \\
R_3 & 0 & 1 & 1 & 0 \\
\end{array}
\]

\[ S = \{p_5\} \cup S' \]

\[ p_5, \text{ plus an optimum solution (say } p_1p_2 \text{) to the reduced matrix, is an optimum solution of the original problem} \]
• Row $i$ of $M$ dominates Row $k$ if every nonzero of Row $k$ is matched by a nonzero of Row $i$ in the same column, that is, $M_{kj} = 1 \Rightarrow M_{ij} = 1, \forall j$
• Any set of columns that covers Row $k$ will also cover Row $i$
• Hence dominating rows may be deleted without affecting the size of the optimum solution

• Such deletion may lead to column dominance (later today)
Row Dominance: Example

Note: Row 4 dominates Row 1

\[
\begin{array}{c|ccccc}
 & p_1 & p_2 & p_3 & p_4 & p_5 \\
1 & 0 & 0 & 1 & 0 & 1 \\
2 & 1 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 0 \\
4 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{c|ccccc}
 & p_1 & p_2 & p_3 & p_4 & p_5 \\
1 & 0 & 0 & 1 & 0 & 1 \\
2 & 1 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 0 \\
\end{array}
\]

\[S = S'\]
Reductions: Column Dominance

• Col $j$ of $M$ dominates Col $k$ if every nonzero of Col $k$ is matched by a nonzero of Col $j$ in the same row, that is $M_{ik} = 1 \Rightarrow M_{ij} = 1, \forall i$

• Any set of columns that contains Col $j$ will also cover all rows covered by Col $k$.

• Hence dominated columns may be deleted without affecting the size of the optimum solution

• Such deletion may lead to row dominance or even reveal new essential columns
Column Dominance: Example

\[ M = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 1 \\
3 & 0 & 1 & 1 & 0 & 0 \\
4 & 1 & 0 & 1 & 0 & 1
\end{bmatrix}, \quad M' = \begin{bmatrix}
1 & 1 & 0 & 1 \\
2 & 1 & 1 & 0 \\
3 & 0 & 1 & 1
\end{bmatrix} \]

\[ S = S' \]

- Col 1 dominates Cols 4 and 5
- Row 4 co-dominates Row 1
- Note cyclic core
Matrix Reduction

Procedure REDUCE($M$)
1. $EC = COLS\_OF\_SINGLETON\_ROWS($M$)
2. "delete cols in $EC$ and rows with cols in $EC$"
3. "Add cols in $EC$ to optimum solution"
4. "delete rows which dominate other rows"
5. "delete cols which are dominated by other cols"
6. if ($M \neq \emptyset$) repeat 1-6

- Essential Column reduction
- Row Dominance reduction
- Column Dominance reduction
- Iterate: Result is unique “Cyclic Core”
Matrix Reduction

Unate Covering

\[
\begin{array}{cccccc|cccc}
 & p_1 & p_2 & p_3 & p_4 & p_5 & p_1 & p_2 & p_3 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
M = 2 & 1 & 1 & 0 & 1 & 0 & & & \\
3 & 0 & 1 & 1 & 0 & 0 & & & \\
4 & 1 & 0 & 1 & 1 & 1 & & & \\
\end{array}
\]

Note: Row dominance kills constraints

\[
M : (p_1 + p_3 + p_5)(p_1 + p_2 + p_4)(p_2 + p_3)(p_1 + p_3 + p_4 + p_5) \\
= (p_1 + (p_3 + p_5)(p_2 + p_4))(p_2 + p_3) \\
= p_1(p_2 + p_3) + (p_3 + p_5)(p_2 + p_3) \\
= p_1p_2 + p_1p_3 + p_2p_3 + p_3p_4 + p_2p_5 \\
M' : (p_1 + p_3)(p_1 + p_2)(p_2 + p_3) \\
= (p_1 + p_2p_3)(p_2 + p_3) = p_1p_2 + p_1p_3 + p_2p_3
\]

Note: Col dominance kills alternative solutions
Exploring Search Space

- When the constraint matrix cannot be further reduced
  - If the matrix has no rows left, problem solved
  - Otherwise, cyclic

- To solve large enumeration problems, use implicit enumeration

- Branch and Bound
Lower Bounds

\[ M = \begin{array}{cccccc}
1 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 \\
4 & 1 & 0 & 1 & 0 & 1 \\
\end{array} \]

- Note rows 1, 2, 3 are “column disjoint”, that is, the nonzero column sets \{1,3\}, \{2,4\}, \{5,6\} are pairwise disjoint
- Thus at least 3 columns are required to cover the rows of \( M \)
- Thus if we somehow find a solution of size 3, we can take it and quit --- it is an optimum solution
Lower Bounds

• Row $i$ of $M$ is disjoint from Row $k$ if no nonzero of Row $i$ is matched by a nonzero of Row $k$ in the same column, and conversely. That is,

\[ M_{ij} = 1 \implies M_{kj} = 0, \quad M_{kj} = 1 \implies M_{ij} = 0, \quad \forall j \]

• $m$ distinct columns are required to cover $m$ pair-wise disjoint rows
• Hence the cost of covering a matrix that contains $m$ pair-wise disjoint rows is at least $m$

• Lower bounds can prune vast regions of the search
An independent set of rows is called **maximal** if it intersects every other row of the covering matrix.

Unless some of the decisions already made in building the set are reversed, the set cannot grow larger while retaining its independence.
Quick Lower Bound Algorithm

Procedure MIS_QUICK( M )
1  MIS = ∅
2  do {
3    i = CHOOSE_SHORTEST_ROW( M )
4    MIS = MIS ∪ {i}
5    M = DELETE_INTERSECTING_ROWS( M )
6  } while(∥ M ∥ > 0) continue
7  return row set MIS

This is a cheap heuristic:
Finding “best” lower bound can be harder than solving the original covering problem
**MIS QUICK Example**

**Procedure** MIS QUICK( M )

1. \( MIS = \emptyset \)
2. do 
   3. \( i = \text{CHOOSE}_\text{SHORTEST}_\text{ROW}( M ) \)
   4. \( MIS = MIS \cup \{ i \} \)
   5. \( M = \text{DELETE}_\text{INTERSECTING}_\text{ROWS}( M ) \)
3. while(\( |M| > 0 \)) continue
4. return row set MIS

\[
\begin{array}{ccccccc}
1 & \times & 1 & 1 & 0 & 0 & 0 & 0 \leftarrow MIS = \{1\} \\
2 & \times & 0 & 1 & 1 & 0 & 0 & 0 \\
3 & \times & 0 & 0 & 1 & 1 & 0 & 0 \rightarrow 3 & \times & 0 & 0 & 1 & 1 & 0 & 0 \leftarrow MIS = \{1,3\} \\
4 & \times & 0 & 0 & 0 & 1 & 1 & 0 & 4 & \times & 0 & 0 & 0 & 1 & 1 & 0 \\
5 & \times & 0 & 0 & 0 & 0 & 1 & 1 & 5 & 0 & 0 & 0 & 0 & 1 & 1 \\
6 & \times & 1 & 0 & 0 & 0 & 0 & 1 & 5 & \times & 0 & 0 & 0 & 0 & 1 & 1 & 1 \leftarrow MIS = \{1,3,5\}
\end{array}
\]
### Example

Branch and Bound

<table>
<thead>
<tr>
<th>M</th>
<th>100010</th>
<th>$w_1^1=2$</th>
<th>$w_2^1=5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>110100</td>
<td>$w_1^2=3$</td>
<td>$w_2^2=9$</td>
<td></td>
</tr>
<tr>
<td>011000</td>
<td>$w_1^3=2$</td>
<td>$w_2^3=6$</td>
<td></td>
</tr>
<tr>
<td>000111</td>
<td>$w_1^4=3$</td>
<td>$w_2^4=7$</td>
<td></td>
</tr>
<tr>
<td>001100</td>
<td>$w_1^5=2$</td>
<td>$w_2^5=5$</td>
<td></td>
</tr>
<tr>
<td>010001</td>
<td>$w_1^6=2$</td>
<td>$w_2^6=5$</td>
<td></td>
</tr>
<tr>
<td>101000</td>
<td>$w_1^7=2$</td>
<td>$w_2^7=6$</td>
<td></td>
</tr>
</tbody>
</table>

**MIS={1}** on the first pass

**MIS={1,3}** on the second pass
Example

For second heuristic, MIS={1} on the first pass

M 011000  \quad w^1_3=2 \quad w^2_3=4
001100  \quad w^1_5=2 \quad w^2_5=3
010001  \quad w^1_6=2 \quad w^2_6=3

MIS={1,5} on the second pass
MIS={1,5,6} on the third pass
“Effective”: Enumeration

• Accept the inevitability of exponential worst case performance -- Implicitly Enumerate the search space

• Reduce the problem into simplest equivalent problem--at least one optimum solution remains

• Prune the search space by computing lower bounds