ECE 2300
Digital Logic & Computer Organization
Spring 2024

Combinational Logic Minimization
Announcements

• Weekly calendar is posted on the course web

• HW 1 will be released tonight
## Recap: Minterms & Maxterms

<table>
<thead>
<tr>
<th>XYZ</th>
<th>Minterm</th>
<th>Minterm name</th>
<th>Maxterm</th>
<th>Maxterm name</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>X’•Y’•Z’</td>
<td>m₀</td>
<td>X+Y+Z</td>
<td>M₀</td>
</tr>
<tr>
<td>001</td>
<td>X’•Y’•Z</td>
<td>m₁</td>
<td>X+Y+Z’</td>
<td>M₁</td>
</tr>
<tr>
<td>010</td>
<td>X’•Y•Z’</td>
<td>m₂</td>
<td>X+Y’+Z</td>
<td>M₂</td>
</tr>
<tr>
<td>011</td>
<td>X’•Y•Z</td>
<td>m₃</td>
<td>X+Y’+Z’</td>
<td>M₃</td>
</tr>
<tr>
<td>100</td>
<td>X•Y’•Z’</td>
<td>m₄</td>
<td>X’+Y+Z</td>
<td>M₄</td>
</tr>
<tr>
<td>101</td>
<td>X•Y’•Z</td>
<td>m₅</td>
<td>X’+Y+Z’</td>
<td>M₅</td>
</tr>
<tr>
<td>110</td>
<td>X•Y•Z’</td>
<td>m₆</td>
<td>X’+Y’+Z</td>
<td>M₆</td>
</tr>
<tr>
<td>111</td>
<td>X•Y•Z</td>
<td>m₇</td>
<td>X’+Y’+Z’</td>
<td>M₇</td>
</tr>
</tbody>
</table>

A maxterm is the complement of its corresponding minterm
Boolean Function Representations

- **Problem:** \( Y = 1 \) if and only if \( A \cdot B = (B+C') \)
- **Step 1:** Lay out the truth table
- **Step 2:** Derive canonical forms
  - **Canonical sum** \((Y=1\) if input “hits” the on-set\)
    \[
    Y = A' \cdot B' \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' + A \cdot B \cdot C
    = \sum_{A,B,C} (1,5,6,7)
    \]
  - **Canonical product** \((Y=1\) if input “avoids” the off-set\)
    \[
    Y = (A' \cdot B' \cdot C)'(A' \cdot B \cdot C)'(A' \cdot B + C')'(A + B + C)
    = \prod_{A,B,C} (0,2,3,4)
    \]
- **Step 3:** Simplification (this lecture)

<table>
<thead>
<tr>
<th>(ABC)</th>
<th>(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
</tr>
</tbody>
</table>
Combinational Logic

• Outputs depend only on current inputs
  – Example: Detect if the input is an odd number
  – Can be represented in two-level or multi-level forms

• In contrast, sequential logic has “memory” or “state”
  – Example: Detect if the last two inputs are odd
  – We’ll cover sequential logic later
Two-Level Logic: Sum-of-Products

- **Sum of product terms (SOP)**
  - e.g., $A' \cdot B' + A' \cdot C' + A \cdot B \cdot C$

- **Circuits look something like this**
  - A *bubble* indicates the signal is inverted

![Circuit Diagram](image)
Two-Level Logic: Product-of-Sums

- **Product of sum terms (POS)**
  - e.g., \((A'+C') \cdot (B'+C') \cdot (A+B+C)\)

- **Circuits look something like this**

```
  OR-AND

  Combinational logic can be expressed as SOP or POS
```
Algebraic Simplification

• Apply theorems to canonical sum (or product) to reduce (1) the number of terms, and (2) the number of literals in each term

• Results in a more compact expression and lower cost digital logic implementation

• We focus on minimizing two-level logic in this lecture
Algebraic Simplification Example

- **Binary adder**
  - inputs: A, B, Carry-in (Cin)
  - outputs: Sum, Carry-out (Cout)

- **Truth Table → Canonical sum**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Cout</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S = A' \cdot B' \cdot Cin + A' \cdot B \cdot Cin' + A \cdot B' \cdot Cin' + A \cdot B \cdot Cin \]

\[ Cout = A' \cdot B \cdot Cin + A \cdot B' \cdot Cin + A \cdot B \cdot Cin' + A \cdot B \cdot Cin \]
Algebraic Simplification Example

\[
C_{\text{out}} = A'B'C_{\text{in}} + A'B'C_{\text{in}}' + A'B'C_{\text{in}}' + A'B'C_{\text{in}}
\]

\[
= A'B'C_{\text{in}} + A'B'C_{\text{in}} + A'B'C_{\text{in}}' + A'B'C_{\text{in}}' + A'B'C_{\text{in}}
\quad \text{(idempotency)}
\]

\[
= B'C_{\text{in}} + A'B'C_{\text{in}} + A'B'C_{\text{in}}' + A'B'C_{\text{in}}
\quad \text{(combining)}
\]

\[
= B'C_{\text{in}} + A'B'C_{\text{in}} + A'B'C_{\text{in}}' + A'B'C_{\text{in}}
\quad \text{(combining)}
\]

\[
= B'C_{\text{in}} + A' + A'B
\quad \text{(combining)}
\]

We can apply these theorems in a more intuitive fashion using a *Karnaugh Map*
Reduction in Hardware Cost

\[ \text{Cout} = A'B\cdot C_{\text{in}} + A\cdot B'\cdot C_{\text{in}} + A\cdot B\cdot C_{\text{in}}' + A\cdot B\cdot C_{\text{in}} \]

\[ = B\cdot C_{\text{in}} + A\cdot C_{\text{in}} + A\cdot B \]

3 inverters
4 three-input ANDs
1 four-input OR

\[ \rightarrow \]
3 two-input ANDs
1 three-input OR
Karnaugh Map (K-Map)

• Idea: Use combining and idempotency theorems visually to simplify canonical forms into two-level SOP or POS

• Multidimensional representation of a truth table

• Adjacent cells represent minterms (or maxterms) that differ by exactly one literal
  – Cyclic encoding along each dimension

• At most two variables per dimension
### K-Map for Cout

#### Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### K-Map

![K-Map Diagram](image-url)

- **AB:**
  - 00: 001100
  - 01: 011111
  - 11: 111111
  - 10: 111110
Some K-Map Definitions

- **1-cell**: Minterm of a canonical sum
- **Implicant**: Set of adjacent 1-cells
  - A rectangle in K-Map
  - Number of cells contained must be a power of 2
- **Prime implicant**: Implicant that cannot be contained in a larger implicant
Minimization Using K-Map

• **Goal:** Cover all 1-cells with the minimum number of prime implicants

• **Procedure**
  – Plot 1’s corresponding to minterms of function
  – Circle largest possible **rectangular** sets of 1’s
    • Must be power of 2
    • Including “wrap-around” sets
  – Repeat until all minterms are covered
  – OR product terms derived from each circle

• **Minimizes number of gates and gate inputs in the SOP form**
Simplifying Cout Using a K-Map

\[ \text{Cout} = A'B'Cin + AB'Cin + AB'Cin' + AB'Cin \]

\[ = A'B'Cin + AB'Cin + AB'Cin' + AB'Cin + AB'Cin \]  \hspace{1cm} \text{(idempotency)}

\[ = B'Cin + AB'Cin + AB'Cin' + AB'Cin \] \hspace{1cm} \text{(combining)}

\[ = B'Cin + AB'Cin + AB'Cin' + AB'Cin \] \hspace{1cm} \text{(idempotency)}

\[ = B'Cin + AB'Cin + AB'Cin' + AB'Cin \] \hspace{1cm} \text{(combining)}

\[ = B'Cin + AB'Cin + A'B \] \hspace{1cm} \text{(combining)}

Circles (ovals) indicate applying combining theorem

Idempotency theorem allows circles to overlap
3 Variable K-Map Example

\[ F = \Sigma_{A,B,C}(4,5,6,7) \]

F = A
Another Example

\[ F = \Sigma_{A,B,C}(0,1,2,4,5,7) \]

\[ F = A' \cdot C' + B' + A \cdot C \]
Intuition Behind K-Map

• Spatial encoding!
4 Variable K-Map
Combining Theorem in Action

\[ A'B'C'D' + A'B'CD + ABC'D' + ABCD + A'BD + ABD + BD \]
Combining Theorem in Action

A'B'C'D' + A'B'CD' + AB'C'D' + AB'CD'

A'B'D' + AB'D' + B'D'
• Detect all even digits from 0 to 15 except 6
  \[ F = \Sigma_{A,B,C,D}(0,2,4,8,10,12,14) \]

\[ F = B'D' + AD' + C'D' \]
Minimizing Product-of-Sums

• **Procedure**
  – Plot 0’s corresponding to maxterms of function
  – Circle largest possible rectangular sets of 0’s
    • **Must be power of 2**
    • **Including “wrap-around” sets**
  – Repeat until all maxterms are covered
  – **AND sum terms derived from each circle**
Product-of-Sums Example

\[ F = \Pi_{A,B,C,D}(0, 2, 6, 7, 8, 10) \]

Corners: \( B + D \)

Other: \( A + B' + C' \)

\[ F = (B + D) \cdot (A + B' + C') \]
Don’t Cares Combinations

• Sometimes the output for a particular input combination is unspecified or irrelevant
  – Such as an input combination that will never happen

• Represent as a ‘d’ (or ‘x’) in the truth table

• Example: Detect all even decimal digits except 6
  – Four input bits are still required, but inputs 0-9 would appear
  – 10-15 are “don’t care” values
Don’t Cares in Karnaugh Map

• Represent as a ‘d’ (or ‘x’) in the K-Map

• Don’t cares can be used as 1- or 0-cells as needed

• Only circle if doing so creates a larger prime implicant (and thus a more minimal expression)
Don’t-Care Example

- Detect all even decimal digits except 6
  - Inputs 10-15 will never occur

F = B’D’ + C’D’
Before Next Class

• H&H 1.7

Next Time

CMOS Logic