Section 3.2. XOR operation

- XOR (⊕)
  - Exclusive-OR
  - Some interpretations:
    • X not equal to Y
    • X and Y has odd number of 1’s

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ⊕ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ X \oplus Y = X'Y + XY' \]
• Theorems:
  - \( X \oplus 0 = X \)
  - \( X \oplus 1 = X' \)
  - \( X \oplus X = 0 \)
  - \( X \oplus X' = 1 \)
  - \( X \oplus Y = Y \oplus X \)
  - \( (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z \)
  - \( X(Y \oplus Z) = XY \oplus XZ \)
  - \( (X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y' \)

• XNOR (\( \Xi \))
  - Inverted exclusive-OR
  - \( \Xi \) notation not used in other text
  - Some interpretations:
    • \( X \) equal to \( Y \) (i.e., equivalence operator)
    • \( X \) and \( Y \) has even number of 1's

\[
\begin{array}{c|c|c}
X & Y & (X \oplus Y)' \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0 \\
\end{array}
\]

\( (X \oplus Y)' = X'Y' + XY \)
• Equivalence is the complement of XOR:

\((X \oplus Y)' = (X'Y + XY')'\)

\[= (X + Y')(X' + Y)\]

\[= X (X' + Y) + Y'(X' + Y)\]

\[= XX' + XY + Y'X' + Y'Y\]

\[= XY + X'Y'\]

\[= (X \equiv Y)\]

Assignment

• Read Section 3.2
• Problems:
  3.8, 3.9, 3.10
Chap 4. Minterm and Maxterm Expansions

Outline

• Conversion of English Sentence to Boolean Equations
• Combinational Logic Design Using a Truth Table
• Minterm and Maxterm Expansions
• Incompletely Specified Functions
• Examples of Truth Table Construction
Conversion of English Sentence to Boolean Equations

- **Mary watches TV if it is Monday night and she has finished her homework**
  
  \[ F = A \land B \]

- **The alarm will ring iff the alarm switch is on and the door is not closed or it is after 6pm and the window is not closed**
  
  \[ Z = AB' + CD' \]

  ![Diagram of alarm circuit](Image)

- Consider A, B, C are three votes, which can be yes (1) or no (0). An output U is 1 iff the vote is unanimous
  - U is 1 iff ABC are all 1’s or all 0’s
  - U = ABC + A’B’C’

- An output M is 1 iff there is a majority vote
  - M = ?
  - Somewhat tedious (and error-prone) to derive the logic expression
  - Truth table is a systematic way to derive the expression
Combinational Logic Design Using a Truth Table

• Switching circuit Logic design using truth table

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>f</th>
<th>f'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>0</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>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

• f is 1 if (row 3 is 1) or (row 4 is 1) or (row 5 is 1) or (row 6 is 1) or (row 7 is 1)

• Logic expression for
  (row 3 is 1): A'BC
  (row 4 is 1): AB'C'
  (row 5 is 1): AB'C
  (row 6 is 1): ABC'
  (row 7 is 1): ABC

• Derive algebraic expression for \( f = 1 \)
  \[
f = A'BC + AB'C' + AB'C + ABC' + ABC
  \]

• f can simplified
  \[
f = A'BC + AB' + AB
  = A'BC + A' + A  \quad \text{by Thm 8D: } x+yz=(x+y)(x+z)
  = A + BC
  \]
• It is easier to think in term of SOP  
• It is less intuitive to derive POS expression directly  
• Alternative way to construct POS expression  
  – Find SOP expression for \( f' \)  
  – Invert \( f' \) to get \( f \)  

\[
f' = A'B'C' + A'B'C + A'BC' \\
f = (f')' \\
= (A'B'C' + A'B'C + A'BC')' \\
= (A + B + C)(A + B + C')(A + B' + C) \\
f\text{ can be simplified} \\
= (A + B)(A + B' + C) \\
= A + B(B' + C) \\
= A + BC \\
\]

• An different interpretation (less intuitive):  
  f is 0 if one of the first three rows is 0  
  – Row 0 is 0: \( A=B=C=0; \) i.e., \( A + B + C \) is 0  
  – Row 1 is 0: \( A=B=0 \ C=1; \) i.e., \( A + B + C' \) is 0  
  – Row 2 is 0: \( A=C=0 \ B=1; \) i.e., \( A + B' + C \) is 0  

\[
f = (A + B + C)(A + B + C')(A + B' + C) \\
\]

\[
\begin{array}{ccc|c|c}
A & B & C & f & f' \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 \\
\end{array}
\]
Minterm and Maxterm Expansions

- A minterm of \( n \) variables is a product of \( n \) literals in which each variable appears exactly once in true or complemented form, but not in both.
- Sum of minterms is referred to as a minterm expansion or a standard SOP.
- Minterm is written in abbreviated form as \( m_i \) where \( i \) is the row whose values make \( m_i = 1 \).
- The row is represented as a binary number.

### 3-variable truth table

- Note that the order of variables is important i.e., the row is \((A \ B \ C)\) not \((A \ C \ B), \ (C \ A \ B)\) etc.

<table>
<thead>
<tr>
<th>Row No.</th>
<th>( A ) ( B ) ( C )</th>
<th>Minterms</th>
<th>Maxterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>( A'B'C' = m_0 )</td>
<td>( A + B + C = M_6 )</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td>( A'B'C = m_1 )</td>
<td>( A + B + C' = M_4 )</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td>( A'BC' = m_2 )</td>
<td>( A + B' + C = M_3 )</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1</td>
<td>( A'BC = m_3 )</td>
<td>( A + B' + C' = M_5 )</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0</td>
<td>( AB'C' = m_4 )</td>
<td>( A' + B + C = M_4 )</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1</td>
<td>( AB'C = m_5 )</td>
<td>( A' + B + C' = M_5 )</td>
</tr>
<tr>
<td>6</td>
<td>1 1 0</td>
<td>( ABC' = m_6 )</td>
<td>( A' + B' + C = M_7 )</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1</td>
<td>( ABC = m_7 )</td>
<td>( A' + B' + C' = M_7 )</td>
</tr>
</tbody>
</table>
• f represented in sum-of-minterms format:
  \[ f(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7 \]
  \[ = \Sigma m(3, 4, 5, 6, 7) \]

• Again, argument order (A, B, C) is important

Finding sum-of-minterms for a logic expression

- Truth table
- Algebraic derivation

E.g.,
  \[ f(a, b, c, d) = a'(b' + d) + acd' \]
  \[ f = a'b' + a'd + acd' \]
  \[ = a'b'(c + c')(d + d') + a'd(b + b')(c + c') + acd'(b + b') \]
  \[ = a'b'c'd' + a'b'c'd + a'b'cd' + a'b'cd + a'b'cd + a'b'cd + a'bc'd' + a'bc'd + a'bc'd + a'bc'd \]
  \[ + a'bc'd + a'bc'd + a'bc'd + a'bc'd + a'bc'd + a'bc'd + a'bc'd + a'bc'd \]
Maxterm

- A maxterm of \( n \) variables is the sum of \( n \) literals in which each variable appears exactly once in true or complemented form, but not in both.
- Product of maxterms is referred to as a maxterm expansion or a standard POS.
- Maxterm is written in abbreviated form as \( M_i \) where \( i \) is the row whose values make \( M_i = 0 \);

- Relationship between \( m_i \) and \( M_i \)
  
  \[(m_i)' = M_i \quad m_i = (M_i)'

- \( f \) represented in product-of-maxterms format:

  \[f(A, B, C) = M_0M_1M_2\]
  
  \[= \prod M(0, 1, 2)\]

  - Argument order \((A,B,C)\) is important

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( f )</th>
<th>( f' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sum-of-minterms and Product-of-maxterms conversion

• Procedure
  – Given: function f in sum-of-minterms format
  – Write f' in sum-of-minterms format
    Note that f' consists of all minterms not in f
  – Find (f')', which is in product-of-maxterms format
    Note that f = (f')'

• E.g.,
  f(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7
  f' (A, B, C) = m_0 + m_1 + m_2
  f = (f')' = M_0M_1M_2

<table>
<thead>
<tr>
<th>A B C</th>
<th>f</th>
<th>f'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Incompletely Specified Functions

• Some input combinations of function may not be used or possible
• e.g., F and B are two signals control a model car to drive forward or backward. An function M is 1 if car in motion and 0 if car is still. The input combination “11” is not possible.
• The condition is known as don’t care (denoted d in truth table).
• When the function is implemented, either 0 or 1 can be assigned
• Good assignment can simplified the final circuit
• E.g.,
  A large digital system would be divided into many subsystems, i.e., \( N_1 \) and \( N_2 \) as an example.

• Assume \( N_1 \) does not generate all possible combinations of values for \( A, B, C \) (e.g., 001, 110).

• \( F = \Sigma m(0, 3, 7) + \Sigma d(1, 6) \)

• \( F = \Pi M(2, 4, 5) \cdot \Pi D(1, 6) \)

• Alternatives:
  1. Assume \( d = 0 \) for both X’s:
     \( F = A'B'C' + A'BC + ABC = A'B'C' + BC \)
  2. Assume \( d = 1 \) to the first X and \( d = 0 \) for the 2nd X:
     \( F = A'B'C' + A'B'C + A'BC + ABC = A'B' + BC \)
  3. Assume \( d = 1 \) to both X’s:
     \( F = A'B'C' + A'B'C + A'BC + ABC' + ABC = A'B' + BC + AB \)

• Choice 2 leads to the simplest solution
Examples of Truth Table Construction

- Design simple binary adder that adds two 1-bit binary numbers \((a, b)\) and produces 2-bit sum.

- \(X = AB\)
- \(Y = A'B + AB'\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>X</th>
<th>Y</th>
<th>SUM</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Design simple binary adder that adds two 2-bit binary numbers \((N_1, N_2)\) and produces 3-bit sum \((N_3)\).

- \(X(A,B,C,D)= \Sigma m(7,10,11,13,14,15)\)
- \(Y(A,B,C,D)= \Sigma m(2,3,5,6,8,9,12,15)\)
- \(Z(A,B,C,D)= \Sigma m(1,3,4,6,9,11,12,14)\)
Assignment

• Read Sections 4.1, 4.2, 4.3, 4.5, 4.6
• Section 4.7 to be covered later
• Problems:
  4.2, 4.3, 4.7, 4.9, 4.13, 4.16, 4.21, 4.23, 4.24