Karnaugh Maps

- Applications of Boolean logic to circuit design
  - The basic Boolean operations are AND, OR and NOT
  - These operations can be combined to form complex expressions, which can also be directly translated into a hardware circuit
  - Boolean algebra helps us simplify expressions and circuits

- Karnaugh Map: A graphical technique for simplifying an expression into a minimal sum of products (MSP) form:
  - There are a minimal number of product terms in the expression
  - Each term has a minimal number of literals

- Circuit-wise, this leads to a minimal two-level implementation
Review: Minterm

- A **product** term in which all the variables appear exactly once, either complemented or uncomplemented, is called a **minterm**.

- A minterm represents exactly one combination of the binary variables in a truth table. It has the value of 1 for that combination and 0 for the others.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Product Term</th>
<th>Symbol</th>
<th>m₀</th>
<th>m₁</th>
<th>m₂</th>
<th>m₃</th>
<th>m₄</th>
<th>m₅</th>
<th>m₆</th>
<th>m₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \overline{X} \overline{Y} \overline{Z} )</td>
<td>m₀</td>
<td>1</td>
<td>0</td>
<td>0</td>
<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>( \overline{X} \overline{Y} Z )</td>
<td>m₁</td>
<td>0</td>
<td>1</td>
<td>0</td>
<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>( \overline{X} Y \overline{Z} )</td>
<td>m₂</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( X \overline{Y} \overline{Z} )</td>
<td>m₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( X \overline{Y} \overline{Z} )</td>
<td>m₄</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>( X \overline{Y} Z )</td>
<td>m₅</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( X \overline{Y} \overline{Z} )</td>
<td>m₆</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( X Y Z )</td>
<td>m₇</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 2-6 Minterms for Three Variables*
Review: Maxterm

- A **sum** term in which all the variables appear exactly once, either complemented or uncomplemented, is called a **maxterm**

- A maxterm represents exactly one combination of the binary variables in a truth table. It has the value of 0 for that combination and 1 for the others.

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>Sum Term</th>
<th>Symbol</th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
<th>$M_6$</th>
<th>$M_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$X + Y + Z$</td>
<td>$M_0$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$X + Y + \overline{Z}$</td>
<td>$M_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$X + \overline{Y} + Z$</td>
<td>$M_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$X + \overline{Y} + \overline{Z}$</td>
<td>$M_3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\overline{X} + Y + Z$</td>
<td>$M_4$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$\overline{X} + Y + \overline{Z}$</td>
<td>$M_5$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\overline{X} + \overline{Y} + Z$</td>
<td>$M_6$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\overline{X} + \overline{Y} + \overline{Z}$</td>
<td>$M_7$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2-7 Maxterms for Three Variables

- A minterm and maxterm with the same subscript are the complements of each other, i.e., $M_j = m'_j$
A Boolean function can be represented algebraically from a given truth table by forming the logical sum of all the minterms that produce a 1 in the function. This expression is called a **sum of minterms**

\[
F = \sum \text{m}(0, 2, 5, 7)
\]

\[
F(X, Y, Z) = X'Y'Z' + X'YZ' + XY'Z + XYZ
\]

\[
= m_0 + m_2 + m_5 + m_7
\]
Review: Product of Maxterms

- A Boolean function can be represented algebraically from a given truth table by forming the logical product of all the maxterms that produce a 0 in the function. This expression is called a product of maxterms.

<table>
<thead>
<tr>
<th>(a)</th>
<th>X</th>
<th>Y</th>
<th>Z</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>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</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>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
F = (X+Y+Z')(X+Y'+Z')(X'+Y+Z)(X'+Y'+Z) = M_1 \cdot M_3 \cdot M_4 \cdot M_6
\]

\[
F(X,Y,Z) = \prod M(1,3,4,6)
\]

- To convert a Boolean function \(F\) from SoM to PoM:
  - Find \(F'\) in SoM form
  - Find \(F = (F')'\) in PoM form
Review: Important Properties of Minterms

• There are $2^n$ minterms for $n$ Boolean variables. These minterms can be evaluated from the binary numbers from 0 to $2^n-1$

• Any Boolean function can be expressed as a logical sum of minterms

• The complement of a function contains those minterms not included in the original function

$$F(X,Y,Z) = \Sigma m(0,2,5,7) \Rightarrow F'(X,Y,Z) = \Sigma m(1,3,4,6)$$

• A function that includes all the $2^n$ minterms is equal to logic 1

$$G(X,Y) = \Sigma m(0,1,2,3) = 1$$
Review: Sum-of-Products

• The sum-of-minterms form is a standard algebraic expression that is obtained from a truth table.

• When we simplify a function in SoM form by reducing the number of product terms or by reducing the number of literals in the terms, the simplified expression is said to be in Sum-of-Products form.

• Sum-of-Products expression can be implemented using a two-level circuit.

\[ F = \Sigma m(0, 1, 2, 3, 4, 5, 7) \quad \text{(SoM)} \]
\[ = Y' + X'YZ' + XY \quad \text{(SoP)} \]

Fig. 2-5 Sum-of-Products Implementation
Review: Product-of-Sums

- The product-of-maxterms form is a standard algebraic expression that is obtained from a truth table.

- When we simplify a function in PoM form by reducing the number of sum terms or by reducing the number of literals in the terms, the simplified expression is said to be in Product-of-Sums form.

- Product-of-Sums expression can be implemented using a two-level circuit.

\[
F = \prod M(0,2,3,4,5,6) \quad \text{(PoM)} \\
\quad = X(Y' + Z)(X + Y + Z') \quad \text{(PoS)}
\]

Fig. 2-7 Product-of-Sums Implementation
Re-arranging the Truth Table

• A two-variable function has four possible minterms. We can re-arrange these minterms into a Karnaugh map

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$xy'$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$x'y$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$xy'$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$xy$</td>
</tr>
</tbody>
</table>

• Now we can easily see which minterms contain common literals
  - Minterms on the left and right sides contain $y'$ and $y$ respectively
  - Minterms in the top and bottom rows contain $x'$ and $x$ respectively
Karnaugh Map Simplifications

- Imagine a two-variable sum of minterms:
  
  \[ x'y' + x'y \]

- Both of these minterms appear in the top row of a Karnaugh map, which means that they both contain the literal \( x' \)

\[
\begin{array}{c|c|c}
X & x'y' & x'y \\
\hline
x'y' & \boxed{x'y'} & \boxed{x'y} \\
xy' & xy' & xy \\
\end{array}
\]

- What happens if you simplify this expression using Boolean algebra?

  \[
x'y' + x'y = x'(y' + y) \quad \text{[ Distributive ]}
  = x' \cdot 1 \quad \text{[ } y + y' = 1 \text{ ]}
  = x' \quad \text{[ } x \cdot 1 = x \text{ ]}
\]
More Two-Variable Examples

- Another example expression is \( x'y + xy \)
  - Both minterms appear in the right side, where \( y \) is uncomplemented
  - Thus, we can reduce \( x'y + xy \) to just \( y \)

- How about \( x'y' + x'y + xy \)?
  - We have \( x'y' + x'y \) in the top row, corresponding to \( x' \)
  - There's also \( x'y + xy \) in the right side, corresponding to \( y \)
  - This whole expression can be reduced to \( x' + y \)
A Three-Variable Karnaugh Map

• For a three-variable expression with inputs x, y, z, the arrangement of minterms is more tricky:

• Another way to label the K-map (use whichever you like):

\[
\begin{array}{c|cccc}
& 00 & 01 & 11 & 10 \\
\hline
x'z' & x'y'z & x'y'z & x'yz & x'yz' \\
xy'z & xy'z & xyz & xyz' \\
x'y'z' & x'y'z & x'yz & x'yz' \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& m_0 & m_1 & m_3 & m_2 \\
\hline
m_4 & m_5 & m_7 & m_6 \\
\end{array}
\]
Why the funny ordering?

- With this ordering, any group of 2, 4 or 8 adjacent squares on the map contains common literals that can be factored out.

- “Adjacency” includes wrapping around the left and right sides:

- We’ll use this property of adjacent squares to do our simplifications.
Example K-map Simplification

- Let's consider simplifying \( f(x, y, z) = xy + y'z + xz \)

- First, you should convert the expression into a sum of minterms form, if it's not already
  - The easiest way to do this is to make a truth table for the function, and then read off the minterms
  - You can either write out the literals or use the minterm shorthand

- Here is the truth table and sum of minterms for our example:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>( f(x,y,z) )</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>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
f(x,y,z) = x'y'z + xy'z + xyz' + xyz = m_1 + m_5 + m_6 + m_7
\]
Unsimplifying Expressions

• You can also convert the expression to a sum of minterms with Boolean algebra
  - Apply the distributive law in reverse to add in missing variables.
  - Very few people actually do this, but it’s occasionally useful.

\[
xy + y'z + xz = (xy \cdot 1) + (y'z \cdot 1) + (xz \cdot 1) \\
= (xy \cdot (z' + z)) + (y'z \cdot (x' + x)) + (xz \cdot (y' + y)) \\
= (xyz' + xyz) + (x'y'z + xy'z) + (xy'z + xyz) \\
= xyz' + xyz + x'y'z + xy'z
\]

• In both cases, we’re actually “unsimplifying” our example expression
  - The resulting expression is larger than the original one!
  - But having all the individual minterms makes it easy to combine them together with the K-map
Making the Example K-map

- Next up is drawing and filling in the K-map
  - Put 1s in the map for each minterm, and 0s in the other squares
  - You can use either the minterm products or the shorthand to show where the 1s and 0s belong
- In our example, we can write \( f(x,y,z) \) in two equivalent ways

\[
f(x,y,z) = x'y'z + xy'z + xyz' + xyz
\]

\[
f(x,y,z) = m_1 + m_5 + m_6 + m_7
\]

- In either case, the resulting K-map is shown below
K-maps From Truth Tables

- You can also fill in the K-map directly from a truth table
  - The output in row $i$ of the table goes into square $m_i$ of the K-map
  - Remember that the rightmost columns of the K-map are “switched”

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$f(x,y,z)$</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>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Grouping the Minterms Together

- The most difficult step is grouping together all the 1s in the K-map
  - Make rectangles around groups of one, two, four or eight 1s
  - All of the 1s in the map should be included in at least one rectangle
  - Do not include any of the 0s

- Each group corresponds to one product term. For the simplest result:
  - Make as few rectangles as possible, to minimize the number of products in the final expression.
  - Make each rectangle as large as possible, to minimize the number of literals in each term.
  - It's all right for rectangles to overlap, if that makes them larger.
Reading the MSP from the K-map

• Finally, you can find the minimal SoP expression
  - Each rectangle corresponds to one product term
  - The product is determined by finding the common literals in that rectangle

For our example, we find that $xy + y'z + xz = y'z + xy$. (This is one of the additional algebraic laws from last time.)
Practice K-map 1

- Simplify the sum of minterms $m_1 + m_3 + m_5 + m_6$

\[
\begin{array}{ccc}
X & & y \\
& & \\
& Z & \\
\hline
X & m_0 & m_1 & m_3 & m_2 \\
& m_4 & m_5 & m_7 & m_6 \\
\end{array}
\]
Solutions for Practice K-map 1

- Here is the filled in K-map, with all groups shown
  - The magenta and green groups overlap, which makes each of them as large as possible
  - Minterm \( m_6 \) is in a group all by its lonesome

\[
\begin{array}{cccc}
X & 0 & 1 & 0 & 1 \\
Y & 1 & 1 & 0 & 0 \\
Z & 0 & 1 & 0 & 1 \\
\end{array}
\]

- The final MSP here is \( x'z + y'z + xyz' \)
K-maps can be tricky!

- There may not necessarily be a unique MSP. The K-map below yields two valid and equivalent MSPs, because there are two possible ways to include minterm \( m_7 \):

\[
\begin{array}{ccc}
X & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
Y & 0 & 1 \\
0 & 1 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
Z & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
y'z + yz' + xy
\]

\[
y'z + yz' + xz
\]

- Remember that overlapping groups is possible, as shown above.
Four-variable K-maps

- We can do four-variable expressions too!
  - The minterms in the third and fourth columns, and in the third and fourth rows, are switched around.
  - Again, this ensures that adjacent squares have common literals.

- Grouping minterms is similar to the three-variable case, but:
  - You can have rectangular groups of 1, 2, 4, 8 or 16 minterms
  - You can wrap around all four sides
Four-variable K-maps

\begin{align*}
\begin{array}{cccc}
  & 00 & 01 & 11 & 10 \\
wx & X & Y & Z \\
00 & w'x'y'z' & w'x'y'z & w'x'yz & w'x'yz' \\
01 & w'xy'z' & w'xy'z & w'xyz & w'xyz' \\
11 & wxy'z' & wxy'z & wxyz & wxyz' \\
10 & wx'y'z' & wx'y'z & wx'yz & wx'yz' \\
w & Y & Z & X \\
\end{array}
\end{align*}
Example: Simplify $m_0 + m_2 + m_5 + m_8 + m_{10} + m_{13}$

- The expression is already a sum of minterms, so here's the K-map:

  \[
  \begin{array}{ccc}
  & Y & \\
  W & 1 & 0 & 0 & 1 \\
  & 0 & 1 & 0 & 0 \\
  Z & 0 & 1 & 0 & 0 \\
  & 1 & 0 & 0 & 1 \\
  \end{array}
  \]

  \[
  \begin{array}{ccc}
  & Y & \\
  W & m_0 & m_1 & m_3 & m_2 \\
  & m_4 & m_5 & m_7 & m_6 \\
  Z & m_{12} & m_{13} & m_{15} & m_{14} \\
  & m_8 & m_9 & m_{11} & m_{10} \\
  \end{array}
  \]

- We can make the following groups, resulting in the MSP $x'z' + xy'z$

  \[
  \begin{array}{ccc}
  & Y & \\
  W & 1 & 0 & 0 & 1 \\
  & 0 & 1 & 0 & 0 \\
  Z & 0 & 1 & 0 & 0 \\
  & 1 & 0 & 0 & 1 \\
  \end{array}
  \]

  \[
  \begin{array}{ccc}
  & Y & \\
  W & w'xy'z & w'xyz & w'xyz & w'xyz \\
  & w'xy'z & w'xy'z & w'xyz & w'xyz \\
  Z & wxy'z & wxy'z & wxyz & wxyz \\
  & wxy'z & wxy'z & wxyz & wxyz \\
  \end{array}
  \]
Five-variable K-maps

\[ \begin{array}{c|c|c|c}
 V = 0 & \text{Y} & \text{Z} \\
00 & 01 & 11 & 10 \\
\hline
00 & & & \\
01 & & & \\
11 & & & \\
10 & & & \\
\end{array} \]

\[ \begin{array}{c|c|c|c}
 V = 1 & \text{Y} & \text{Z} \\
00 & 01 & 11 & 10 \\
\hline
00 & & & \\
01 & & & \\
11 & & & \\
10 & & & \\
\end{array} \]
Simplify \( f(V,W,X,Y,Z) = \Sigma m(0,1,4,5,6,11,12,14,16,20,22,28,30,31) \)

\[
f = XZ' + V'W'Y' + W'Y'Z' + VWXY + V'WX'YZ = m11
\]
PoS Optimization from SoP

F(W,X,Y,Z) = Σm(0,1,2,5,8,9,10)

= ΠM(3,4,6,7,11,12,13,14,15)

F(W,X,Y,Z) = (W' + X')(Y' + Z')(X' + Z)
SoP Optimization from PoS

\[ F(W,X,Y,Z) = \prod M(0,2,3,4,5,6) \]

\[ = \Sigma m(1,7,8,9,10,11,12,13,14,15) \]

\[ F(W,X,Y,Z) = W + XYZ + X'Y'Z \]
I don’t care!

• You don’t always need all $2^n$ input combinations in an $n$-variable function
  - If you can guarantee that certain input combinations never occur
  - If some outputs aren’t used in the rest of the circuit

• We mark don’t-care outputs in truth tables and K-maps with Xs.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>$f(x,y,z)$</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>X</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>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• Within a K-map, each X can be considered as either 0 or 1. You should pick the interpretation that allows for the most simplification.
• Find a MSP for

\[ f(w,x,y,z) = \Sigma m(0,2,4,5,8,14,15), \ d(w,x,y,z) = \Sigma m(7,10,13) \]

This notation means that input combinations \( wxyz = 0111, 1010 \) and 1101 (corresponding to minterms \( m_7, m_{10} \) and \( m_{13} \)) are unused.
Solutions for Practice K-map 3

• Find a MSP for:

\[ f(w,x,y,z) = \Sigma m(0, 2, 4, 5, 8, 14, 15), \quad d(w,x,y,z) = \Sigma m(7, 10, 13) \]

\[ f(w,x,y,z) = x'z' + w'xy' + wxy \]
### AND, OR, and NOT

<table>
<thead>
<tr>
<th>Name</th>
<th>Distinctive shape</th>
<th>Algebraic equation</th>
<th>Truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>![AND Symbol]</td>
<td>$F = XY$</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>OR</td>
<td>![OR Symbol]</td>
<td>$F = X + Y$</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>NOT (inverter)</td>
<td>![NOT Symbol]</td>
<td>$F = \overline{X}$</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Buffer and 3-State Buffer

- **Buffer**
  - Used to amplify an electrical signal
  - Reconstructing the signal
  - More gates to be attached to the output

- **Three state buffer**
  - E (Enable): Controls the output
  - Hi-Z: High impedance
NAND and NOR

- NAND: Not AND, NOR: Not OR
- Both NAND and NOR are universal gates
- Universal gate: A gate that alone can be used to implement all Boolean functions
- It is sufficient to show that NAND (NOR) can be used to implement AND, OR, and NOT operations
NANDs are special!

- The NAND gate is universal: it can replace all other gates!
  
  - NOT
    \[(xx)' = x' \quad \text{[because } xx = x \text{]}\]

  - AND
    \[((xy)' (xy))' = xy \quad \text{[from NOT above]}\]

  - OR
    \[((xx)' (yy))' = (x' y')' \quad \text{[xx = x, and yy = y]}\]
    \[= x + y \quad \text{[DeMorgan's law]}\]
XOR and XNOR

• **Exclusive-OR (XOR):** \( X \oplus Y = XY' + X'Y \)

• **Exclusive-NOR (XNOR):** \((X \oplus Y)' = XY + X'Y'\)

\( X \oplus 0 = X \)
\( X \oplus 1 = X' \)
\( X \oplus X = 0 \)
\( X \oplus X' = 1 \)
\( X \oplus Y' = (X \oplus Y)' \)
\( X' \oplus Y = (X \oplus Y)' \)
\( X \oplus Y = Y \oplus X \)
\( (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z \)
More on XOR

• The general XOR function is true when an odd number of its arguments are true.

• For example, we can use Boolean algebra to simplify a three-input XOR to the following expression and truth table.

\[
x \oplus (y \oplus z)
= x \oplus (y'z + yz')
= x'(y'z + yz') + x(y'z + yz')'
= x'y'z + x'yz' + x(y'z + yz')'
= x'y'z + x'yz' + x((y'z)'(yz'))'
= x'y'z + x'yz' + x((y + z')(y' + z))'
= x'y'z + x'yz' + xyz + xy'z'
= x'y'z + x'yz' + xyz + xy'z'
\]

- Definition of XOR
- Definition of XOR
- Distributive
- DeMorgan’s
- DeMorgan’s
- Distributive
- Distributive

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>x \oplus y \oplus z</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>
High-Impedance Outputs

- Gates with only output values logic 0 and logic 1
  - The output is connected to either Vcc or Gnd
- A third output value: High-Impedance (Hi-Z, Z, or z)
  - The output behaves as an open-circuit, i.e., it appears to be disconnected
- Gates with Hi-Z output values can have their outputs connected together if no two gates drive the line at the same time to opposite 0 and 1 values
- Gates with only logic 0 and logic 1 outputs cannot have their outputs connected together
Three-State Buffers

(a) Logic symbol

(b) Truth table

<table>
<thead>
<tr>
<th>EN</th>
<th>IN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Hi-Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Three-State Buffers

(a) Logic Diagram

(b) Truth table

<table>
<thead>
<tr>
<th>EN1</th>
<th>EN0</th>
<th>IN1</th>
<th>IN0</th>
<th>OL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>Hi-Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<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>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</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>
Transmission Gate

If $C = 1$ ($C' = 0$) $\Rightarrow Y = X$

If $C = 0$ ($C' = 1$) $\Rightarrow Y = Hi-Z$
Transmission Gate XOR

(a)

(b)

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>TG1</th>
<th>TG0</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No path</td>
<td>Path</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Path</td>
<td>No path</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>No path</td>
<td>Path</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Path</td>
<td>No path</td>
<td>0</td>
</tr>
</tbody>
</table>
K-map Summary

• K-maps are an alternative to algebra for simplifying expressions
  - The result is a *minimal sum of products*, which leads to a minimal two-level circuit
  - It’s easy to handle don’t-care conditions
  - K-maps are really only good for manual simplification of small expressions...

• Things to keep in mind:
  - Remember the correct order of minterms on the K-map
  - When grouping, you can wrap around all sides of the K-map, and your groups can overlap
  - Make as few rectangles as possible, but make each of them as large as possible. This leads to fewer, but simpler, product terms
  - There may be more than one valid solution