NAND and NOR Implementation

- We found that we could implement general Boolean equations with these three primitives:
  - AND
  - OR
  - NOT

- In this section we will find that either of two gates, the NAND gate or the NOR gate can be used to implement arbitrary logic functions.

- We use the Positive Logic Convention (where all signals are active high) and a small circle to on a symbol to represent NOT or invert.
NAND Gates

- The basic positive logic NAND gate is denoted by the following symbol:
  - AND-Invert (NAND)
    
    ![NAND Gate Diagram]

    $$F(X, Y, Z) = \overline{X \cdot Y \cdot Z}$$

- NAND comes from NOT AND, i.e., the AND function with a NOT applied. We call this symbol for a NAND gate an AND-Invert. The small circle represents the invert function.

- If we apply DeMorgan's Law we get:
  $$\overline{X \cdot Y \cdot Z} = \overline{X} + \overline{Y} + \overline{Z}$$

NAND Gates (Cont.)

- Applying DeMorgan's Law gives:
  - Invert-OR (NAND)
    
    ![Invert-OR NAND Gate Diagram]

    $$F(X, Y, Z) = \overline{X} + \overline{Y} + \overline{Z}$$

- We call this symbol for a NAND gate the Invert-OR since all inputs are inverted, followed by the OR function.

- Both symbols represent the NAND gate - it is sometimes more logically descriptive to use one form over the other.

- A NAND gate with one input degenerates to an inverter.
NAND Function Implementation

- NAND gates can implement a simplified Sum-of-Products form. Constructing two level NAND-NAND gate circuit:

\[ G(A, B, C, D) = A \cdot B + C \cdot D \]

- The first level is two 2-input NAND gates using AND-Invert. The second level is one 2-input NAND gate using Invert-OR. Using the NAND relationship, we have:

\[
G(A, B, C, D) = \overline{AB} \cdot \overline{CD} \\
= AB + CD \\
= A \cdot B + C \cdot D
\]

NAND Implementation (Cont.)

- In the implementation, the bubbles are on opposite ends of the same line.
- By \( X = \overline{X} \), they can be combined and deleted:

\[ G(A, B, C, D) \]

- A sum-of-products (SOP) form results
- To implement an equation like: \( F(A, B, C) = A + BC \), the NAND for \( A \) degenerates to a NOT since there is only one input
Degenerate AND Term

- The degenerate NAND becomes an inverter:

- To implement the complement of F using NAND gates, add an inverter to the output:

NAND-NAND Example

- Implement: $F'(w, x, y, z) = \overline{\overline{y}z + \overline{w}x + \overline{x}y + \overline{y}z}$
Summary: Two-Level NAND Circuits

- Find minimum literal SOP form for $F$ and $\overline{F}$
- Select SOP form with smallest literal count
- Convert selected form to NAND circuit using AND-invert (inverters for single literal AND terms) and invert-OR symbols
- If SOP form for $\overline{F}$ used, add inverter to circuit output.

NOR Gates

The basic positive logic NOR gate (Not-OR) is denoted by the following symbol:

```
  OR-Invert (NOR)  X
                Y
                Z

  F(X, Y, Z) = X + Y + Z
```

This is called the OR-Invert, since it is logically an OR function followed by an invert. By DeMorgan's Law we have the following Invert-AND symbol for a NOR gate:

```
  Invert-AND  X
            Y
            Z
```

A single-input NOR gate is an inverter, too.
NOR Gates

- The basic positive logic NOR gate is denoted by the following symbol:
  - OR-Invert (NOR)

  \[ F(X, Y, Z) = \overline{X} \cdot \overline{Y} \cdot \overline{Z} \]

- NOR comes from NOT OR, i.e., the OR function with a NOT applied. We call this symbol for a NOR gate an OR-Invert. The small circle represents the invert function.

- If we apply DeMorgan's Law we get:
  \[ \overline{X + Y + Z} = \overline{X} \cdot \overline{Y} \cdot \overline{Z} \]

NOR Gates (Cont.)

- Applying DeMorgan's Law gives:
  - Invert-AND (NOR)

  \[ F(X, Y, Z) = X \cdot Y \cdot Z \]

- We call this symbol for a NOR gate the Invert-AND since all inputs are inverted, followed by the AND function.

- Both symbols represent the NOR gate - it is sometimes more logically descriptive to use one form over the other.

- A NOR gate with one input degenerates to an inverter.
NOR Function Implementation

- NAND gates can implement a simplified Sum-of-Products form. Constructing two-level NOR-NOR circuit:

  \[ G(A, B, C, D) = (A + B) \cdot (C + D) \]

- The first level is two 2-input NOR gates using OR-Invert. The second level is one 2-input NOR gate using Invert-AND.

- Using the NOR relationship, we have:

  \[ G(A, B, C, D) = (A + B) \cdot (C + D) \]
  \[ = (A + B) \cdot (C + D) \]
  \[ = (A + B) \cdot (C + D) \]

Useful Transformations

From Involution (i.e. \((A')' = A\)) and DeMorgan's Law, we get the following useful equivalences:

\[ (A \cdot B) = ((A \cdot B)')' \Leftrightarrow (A' + B')' \]
\[ (A + B) = \Leftrightarrow (A' \cdot B')' \]
\[ ((A + B)')' \Leftrightarrow (A' + B') \]
\[ (A \cdot B)' \Leftrightarrow (A' + B') \]
\[ (A + B)' \Leftrightarrow (A' \cdot B') \]

These simple transformations can be used to manipulate a two level network.
Graphical Transformations

The relations from the previous slide lead to the following transformations:

\[
\begin{align*}
(A \cdot B)' &= (A' + B')' \\
(A + B)' &= (A \cdot B)' \\
(A' + B')' &= (A + B)' \\
(A' \cdot B)' &= (A + B)'
\end{align*}
\]

Recall that two bubbles in series can be removed from the circuit.

General Two-level Implementations

We need to consider whether the form of a two-level implementation is to be:

1. **SOP** (AND-OR) or
2. **POS** (OR-AND).

Complemented output functions (i.e. AND-NOR or OR-NAND) can be handled by complementing the function.

Given a function \( F \) expressed as a Karnaugh Map, we can use the same general procedures we have used before to minimize the function and express it in SOP or POS form.
General Implementations (Cont.)

Given a two level implementation desired, use the previous transforms to get it into one of the below forms. Then follow the steps to transform the function to the desired form:

<table>
<thead>
<tr>
<th>For Type:</th>
<th>Use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND-OR (SOP Form)</td>
<td>Circle 1's in the K-Map and minimize</td>
</tr>
<tr>
<td></td>
<td>(Also use for NAND-NAND)</td>
</tr>
<tr>
<td>AND-NOR (SOP</td>
<td>Circle 0's in the K-Map and minimize</td>
</tr>
<tr>
<td>complemented)</td>
<td></td>
</tr>
<tr>
<td>OR-AND (POS Form)</td>
<td>Circle 0's in the K-Map and minimize</td>
</tr>
<tr>
<td></td>
<td>SOP. Use DeMorgan's to transform to POS.</td>
</tr>
<tr>
<td></td>
<td>(Also use for NOR-NOR)</td>
</tr>
<tr>
<td>OR-NAND (POS</td>
<td>Circle 1's in the K-Map and minimize</td>
</tr>
<tr>
<td>complemented)</td>
<td>SOP. Use DeMorgan's to transform to POS.</td>
</tr>
</tbody>
</table>

Implementation Example 1

Implement the function in NOR-OR.

We can remove the "Inverter" and replace it with the complement of the input variable.
Implementation Example 2

Implement the function in AND-NOR.

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

Multi-level NAND Implementations

- Add inverters in two-level implementation into the cost picture
- Attempt to “combine” inverters to reduce the term count
- Attempt to reduce literal + term count by factoring expression into POSOP or SOPOS
Multi-level NAND Example 1

- \( F = A \cdot B' + A \cdot C' + B \cdot A' + B \cdot C' \)  
  15 inputs and 8 gates*
  
  \[= A \cdot A' + A \cdot B' + A \cdot C' + B \cdot A' + B \cdot B' + B \cdot C'\]
  
  \[= A \cdot (A' + B' + C') + B \cdot (A' + B' + C')\]
  
* Counting inverters (NOTS) as 1 input and 1 gate

---

Multilevel NAND Example 2

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