NAND and NOR Implementation

We found that we could implement general Boolean equations with these three primitives:
1. AND
2. OR
3. 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.

NOTE: The book uses the Positive Logic Convention (where all signals are active high) and uses a small circle to represent complementation.
NAND Gates

The basic positive logic NAND gate is denoted by the following symbol:

\[
\begin{array}{c}
\text{AND-Invert (NAND)} \\
X \\
Y \\
Z \\
\end{array}
\]

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

The term NAND comes from Not AND, referring to the fact that the AND function is followed by an invert. We will 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: \((X \cdot Y \cdot Z)' = X' + Y' + Z'\)

NAND Gates (Cont.)

The application of DeMorgan's Law leads to the following symbol for a NAND Gate:

\[
\begin{array}{c}
\text{Invert-Or} \\
X \\
Y \\
Z \\
\end{array}
\]

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

We call this symbol the Invert-OR since all inputs are inverted, followed by the OR function. Note that both symbols are used to represent the NAND gate - it is sometimes more convenient (or logically descriptive) to use one form over the other.

A single NAND gate with one input is degenerate and becomes an inverter.
NAND Implementation

NAND gates can be used to implement a simplified Sum-of-Products form of a Boolean equation. To see this, let us construct a two level NAND-NAND gate function thus:

\[
\begin{align*}
\text{A} & \quad \text{B} \\
\text{C} & \quad \text{D} \\
\text{G(A,B,C,D)} & = \text{A} \cdot \text{B} + \text{C} \cdot \text{D}
\end{align*}
\]

The first level uses two, 2-input NAND gates shown as the AND-Invert symbol. The second level uses one, 2-input NAND gate shown as the Invert-OR symbol. Using the NAND relationship, we have:

\[
\begin{align*}
\text{G(A,B,C,D)} & = ((\text{A} \cdot \text{B})' \cdot (\text{C} \cdot \text{D})')' \\
& = ((\text{A} \cdot \text{B})')' + ((\text{C} \cdot \text{D})')' \\
& = (\text{A} \cdot \text{B}) + (\text{C} \cdot \text{D})
\end{align*}
\]

NAND Implementation (Cont.)

In the previous implementation, note that the "Complement" circles are on opposite ends of the same line. Thus, they can be combined and deleted:

\[
\begin{align*}
\text{A} & \quad \text{B} \\
\text{C} & \quad \text{D} \\
\text{G(A,B,C,D)} & = \text{A} \cdot \text{B} + \text{C} \cdot \text{D}
\end{align*}
\]

This form of the implementation is the Sum-of-Products form. We may want to implement an equation like: \( F(A,B,C) = A + BC \)

The term "A" is a degenerate "AND" term, and must use an inverter to implement the function.
Degenerate AND Term

The degenerate AND term becomes an inverter:

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

It is easy to implement the complement of a function using NAND gates. Simply add an inverter to the output:

\[ F'(A, B, C) = (A + B \cdot C)' = A'(B' + C') \]

NAND-NAND Example

From the K-Map we see that \( F' \) takes fewer literals to implement than \( F \) in SOP form:

\[ F(w, x, y, z) = y'z' + w'x' + x'y' + w'z' \]

\[ F'(w, x, y, z) = xz + wy \]

and this leads to:

\[ F(w, x, y, z) = (x \cdot z + w \cdot y)' \]
**NAND-NAND Example**

From the K-Map we see that $F'$ takes fewer literals to implement than $F$ in SOP form:

$$F(w,x,y,z) = y'z' + w'x'y' + w'z'$$

$$F'(w,x,y,z) = xz + wy$$

and this leads to:

$$F(w,x,y,z) = (x'z + w'y)'$$

---

**NOR Gates**

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

**OR-Invert (NOR)**

$$F(X,Y,Z) = (X + Y + Z)'$$

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**

$$F(X,Y,Z) = X' \cdot Y' \cdot Z'$$

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

NOR gates can be used to implement a simplified Product-of-Sums form of a Boolean equation. To see this, let us construct a two level NOR-NOR gate function thus:

\[
G(a, b, c, d) = (a+b)\cdot(c+d)
\]

Using the NOR relationship:

\[
G = ((a+b)'+(c+d)')' = ((a+b)')'\cdot((c+d)')' = (a+b)\cdot(c+d)
\]

As with NAND-NAND implementations, you can also implement the complement of a function and use another level of inverter to complement it back to the desired function.

Useful Transformations

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

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

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:

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

Also, two complement bubbles in series can be removed from the network.

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

Implementation Example

Implement the function in NOR-OR. This transforms to OR-NAND, a POS form with the output variable complemented, so we want to implement the complement of F in POS form. Circle the 1's and minimize: F = B' + A'C' Use DeMorgan's Law:

F' = B'(A+C)

We can remove the "Inverter" and replace it with the complement of the input variable.
Implement F in AND-NOR form

The AND-NOR form is already in a consistent SOP form with an output variable complemented, so we implement the SOP form of \( F' \). This form is equivalent to NAND-AND. Circle the 0's on the K-Map and minimize \( F' \):

Implement the network:

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

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

- $F = A \overline{B} + A \overline{C} + B \overline{A} + \overline{B} \overline{C}$  
  15 inputs and 8 gates*
  
  $= A \overline{A} + A B + A \overline{C} + B \overline{A} + \overline{B} \overline{B} + \overline{B} C$
  
  $= A (A' + B' + C') + \overline{B} (A' + B' + C')$

* Counting inverters (NOTS) as 1 input and 1 gate

Multilevel NAND Example 2

- $F = AB + AD' + BC + CD'$  
  12 inputs & 5 gates
  
  $= A(B + D') + C(B + D')$  
  8 inputs & 5 gates