Lecture 14

Today we will
- Learn how to implement mathematical logical functions using logic gate circuitry, using
  - Sum-of-products formulation
  - NAND-NAND formulation
- Learn how to simplify implementation using
  - Boolean algebra
  - Karnaugh maps

Logic Gates

Properties of Logic Functions

- These new functions, AND, OR, etc., are mathematical functions just like +, -, sin(), etc.
- The logic functions are only defined for the domain \{0, 1\} (logic functions can only have 0 or 1 as inputs).
- The logic functions have range \{0, 1\} (logic functions can only have 0 or 1 as outputs).
- AND acts a lot like multiplication.
- OR acts a lot like addition.
- Learn the properties so you can simplify equations!

Properties of Logic Functions

\[
\begin{align*}
A + 0 &= A \\
A + A &= A \\
A + B &= B + A \\
A + (B + C) &= (A + B) + C \\
A \cdot B &= A \cdot A \\
A \cdot B &= B \cdot A \\
A \cdot (B + C) &= A \cdot B + A \cdot C \\
A + B &= A + B \\
\text{DeMorgan’s Law:} & \\
\overline{A \cdot B} &= \overline{A} + \overline{B} \\
\overline{A + B} &= A \cdot B
\end{align*}
\]
De Morgan’s Law

\[ A \cdot B = \overline{A} + \overline{B} \]
\[ A + B = \overline{A} \cdot \overline{B} \]

Logical Synthesis

Suppose we are given a truth table or Boolean expression defining a mathematical logic function. Is there a method to implement the logical function using basic logic gates? One way that always works is the “sum of products” formulation. It may not always be the best implementation for a particular purpose, but it works.

Sum-of-Products Method

1. Create a Boolean expression for the function in sum-of-products form. This means represent the function \( F \) by groups of ANDed inputs (products) that are then ORed together (sum of products).

\[ F = A \cdot B \cdot C + A \cdot B \cdot D \]

\( F = A \cdot B \cdot (C + D) \) is not in sum-of-products form

How to get to sum-of-products form?
- Use properties to manipulate given Boolean equation
- Look at each “1” in truth table, write product of inputs that creates this “1”, OR them all together

Sum-of-Products Method

2. Implement sum-of-products expression with one stage of inverters, one stage of ANDs, and one big OR:

\[ F = A \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot D \]
Example (Adder)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S₁</th>
<th>S₀</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>1</td>
<td>0</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>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Input | Output

S₁ using sum-of-products:

1) Find where S₁ is “1”
2) Write down product of inputs which create each “1”
3) Sum all products
4) Draw circuit

\[
\begin{align*}
\overline{A} \overline{B} \overline{C} + A \overline{B} C + A B \overline{C} + A B C
\end{align*}
\]

NAND-NAND Implementation

- We can easily turn our sum-of-products circuit into one that is made up solely of NANDs (generally cheaper):

Karnaugh Maps

To find a simpler sum-of-products expression, write the truth table of your circuit into a special table.

Example (Adder)

Simplification for S₁:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S₁</th>
<th>S₀</th>
<th>CD</th>
</tr>
</thead>
<tbody>
<tr>
<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>0</td>
<td>1</td>
<td>0</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>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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>
<td>0</td>
</tr>
</tbody>
</table>

Input | Output

2 Inputs | 3 Inputs | 4 Inputs

For each “1”, circle the biggest 2m by 2n block of “1’s” that includes that particular “1”. Write the product that corresponds to that block, and finally sum.