Logic Design Process

Function definition

Truth table

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

Boolean expression

\[ \text{Sum} = (A\bar{B}) + (\bar{A}B) \]

\[ \text{Carry} = AB \]

Logic block
Sum of the Product

- Each row of the truth table represents a product term
- Product term -- each row in which the output column is a 1 contributes a single ANDed term of input variables to the Boolean expressions
  - If the column associated with variable X has a 0 in it, the expression X’ is part of the ANDed term., otherwise, X is part of the ANDed term
- **Sum of the product**
  - Product terms are ORed together
Examples

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</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>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>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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{Sum} = (A\overline{B}) + (\overline{A}B)
\]
\[
\text{Carry} = AB
\]
\[
S = (A\overline{B}C_{in}) + (\overline{A}BC_{in}) + (A\overline{B}C_{in}) + (ABC_{in})
\]
\[
C_{out} = (\overline{A}BC_{in}) + (\overline{A}BC_{in}) + (A\overline{B}C_{in}) + (ABC_{in})
\]
\[
C_{out} = (AC_{in}) + (BC_{in}) + (AB)
\]

You will learn how to reduce the expression to a simple format.
Truth Table → Boolean Expression

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Carry</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Sum = \((AB) + (\overline{A}\overline{B})\)

Carry = \(AB\)
From Truth Table to Minimized Boolean Expression

Truth table $\rightarrow$ sum of product $\rightarrow$ Boolean expression $\rightarrow$ Boolean minimization

OR

Truth table $\rightarrow$ Boolean minimization
Use KMAP
2-variable Karnaugh maps are trivial but can be used to introduce the methods you need to learn. The map for a 2-input OR gate looks like this:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>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>1</td>
</tr>
</tbody>
</table>
Mapping from truth table to k-map

(a)

\[
\begin{array}{ccc}
A & B & F \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

(b)

\[
\begin{array}{ccc}
A & B & G \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
A & B & \text{A} \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
A & B & \text{B} \\
0 & 0 & 2 \\
1 & 1 & 3 \\
\end{array}
\]
3-Variable K-Map

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

\[ \overline{B} + A\overline{C} \]
Mapping from truth table to k-map

\[
\begin{array}{cccc|c}
A & B & C_{in} & C_{out} \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
C_{in} = BC_{in} + AC_{in} + AB
\]
4-Variable K-Map

Out = AB + CD
4-Variable K-Map

Out = \overline{ABC}D + \overline{AB}CD + A\overline{BC}D + AB\overline{C}D

Out = \overline{BD}
Don’t cares

- X: don’t care.
- Do not confuse this with an undefined value.
- Any actual implementation of the circuit will generate some output for the don’t-care cases.
- In a truth table, an X simply means that we have a choice of assigning a 0 or 1 to the truth table entry.
- We should choose the value that will lead to the simplest implementation.
Question 1

- The function has 3 inputs \((a, b, c)\) and 1 output \(f\), \(f, a, b, c\) can only be 0 or 1.
- When \(c = 0\), \(f = a \mid b\) (\(\mid\) represents logical or);
  \(c = 1\), \(f = a + b\). (+ represents add w/o carry out)

Your tasks:
- Draw the truth table
- Derive sum of the product form of \(f\) from the truth table
- Use K-map to minimize \(f\)
- Draw the logic gate implementation of \(f\)
Question 2
Derive the minimized sum of product expression of the following K-map, by choosing the values of x. assuming input is A B C D, and output is F. Draw the new K-map and derive F.
Question 3

- The four inputs to a circuit (A,B,C,D) represent a binary coded decimal digit (*only up to* a single digit, 0--9). Design a circuit so that the output Z is 1 if the decimal number represented by the input is exactly divisible by 3; otherwise if Z < 10, then Z is 0.

- Your tasks:
  - Draw the truth table
  - Use K-map to minimize Z
Bonus Question

- Derive Q as a function of input R and S.
- Q, R, S can only be 0 or 1