eXclusive-OR and Binary Adder

Reouven Elbaz
reouven@uwaterloo.ca
Office room: DC3576
Outline

- **eXclusive OR gate (XOR)**
  - Definition
  - Properties
  - Examples of Applications
    - Odd Function
    - Parity Generation and Checking

- **Binary Adder**
  - Half adder
  - Full adder
  - Binary ripple carry adder
  - Carry lookahead generator
The eXclusive-OR operation is denoted by the symbol: $\oplus$

\[ a \oplus b = ab' + a'b \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>XOR</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>0</td>
</tr>
</tbody>
</table>

Gate Symbol:
XOR properties

- \( a \oplus 0 = a \)
- \( a \oplus 1 = a' \)
- \( a \oplus a = 0 \)
- \( a \oplus a' = 1 \)
- \( a \oplus b' = a' \oplus b = (a \oplus b)' \)

- Commutative: \( a \oplus b = b \oplus a \)
- Associative operation: \( (a \oplus b) \oplus c = (a \oplus c) \oplus b = (b \oplus c) \oplus a \)
- **At home:** show the Commutative and Associative properties by replacing the XOR by its equivalent Boolean expression.
2-input XOR gate construction

- **1st solution**: using AND-OR-NOT gates

- **2nd solution**: using NAND
Odd Function

- An odd function detects an odd number of one in an \( n \)-bit word.
- An \( n \)-variable exclusive-OR operation requires an odd number of variables set to ‘1’ to be evaluated as true (‘1’).
- Ex: \( f = a \oplus b \oplus c \)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f</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>0</td>
<td>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
3-input Odd Function

- \( f = a \oplus b \oplus c = (ab' + a'b)c' + (ab' + a'b)'c \)
  
  \[= ab'c' + a'bc' + (a'+b)(a+b')c\]
  
  \[= ab'c' + a'bc' + a'b'c + abc\]

3-input Even Function: An Even function is true (=‘1’) when it has an even number of ‘1’ as inputs.
A Parity bit (P) is used to detect errors during transmission of binary information.

A parity bit is added to a message to be transmitted to make the number of ‘1’ in the message either even (even parity bit) or odd (odd parity bit).

Parity generator: circuit that generates the parity bit at the transmitter end.

Parity checker: circuit that verifies the parity bit at the receiver end.

How to construct a Parity generator that generates an even parity bit?
The even parity bit must be added to the 3-bit message to make the number of ‘1’ even \( P='1' \) when the number of ‘1’ in the original message is odd.

\[ P = a \oplus b \oplus c \] Odd function

-> How to construct a Parity checker that verifies the parity?

<table>
<thead>
<tr>
<th>3-bit message</th>
<th>Parity bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Parity checker

- How many input bits? ➔ 4: the 3-bit message and P
- When is C (output of the parity checker) evaluated to ‘1’?
  ➔ when the number of ‘1’ at its inputs is odd...
- $C = a \oplus b \oplus c \oplus P$ ➔ Odd function

How to implement a Parity generator with a Parity checker circuit?

➔ Setting $P=\text{‘}0\text{’}$ in the Parity checker circuit provides us with a parity generator (since $c \oplus 0 = c$)
Outline

- **eXclusive OR gate (XOR)**
  - Definition
  - Properties
  - Examples of Applications
    - Odd Function
    - Parity Generation and Checking

- **Binary Adder**
  - Half adder
  - Full adder
  - Binary ripple carry adder
  - Carry lookahead generator
A Binary Adder circuit produces the sum of two $n$-bit words.

Example: Binary addition of two 4-bit words $A$ and $B$:
- $A = "1011"$ and $B = "0011"$ $S = A + B = "1110"

Input carry
\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{S}
\end{array}
\begin{array}{c}
1011 \\
0011 \quad 0 \quad 0
\end{array}
\begin{array}{c}
1011 \\
0011 \quad 0 \quad 1
\end{array}
\begin{array}{c}
1011 \\
0011 \quad 1 \quad 0
\end{array}
\begin{array}{c}
1011 \\
0011 \quad 1 \quad 1
\end{array}
\]

Output carry
\[
\begin{array}{c}
1 \\
1 \\
1 \\
0
\end{array}
\begin{array}{c}
0 \\
10 \\
110 \\
1110
\end{array}
\begin{array}{c}
0 \\
11 \\
110 \\
1110
\end{array}
\begin{array}{c}
0 \\
0 \\
0 \\
0
\end{array}
\]

- Two kind of operations: addition of 2 bits and addition of 3 bits (augend and addend bits and the previous carry)
A Half Adder circuit produces the sum of 2 binary bits and the corresponding carry:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>s</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>

- $S = a \oplus b$
- $C = ab$
A Full Adder circuit produces the sum of 3 binary bits and outputs a carry.

\[
\begin{array}{c|c|c|c|c}
    a & b & C_{in} & C_{out} & S \\
    \\
    0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 1 & 0 & 1 \\
    0 & 1 & 0 & 0 & 1 \\
    0 & 1 & 1 & 1 & 0 \\
    1 & 0 & 0 & 0 & 1 \\
    1 & 0 & 1 & 1 & 0 \\
    1 & 1 & 0 & 1 & 0 \\
    1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Karnaugh map for \( C_{in} \)
\[
S = ab'C_{in}' + a'b' C_{in} + abC_{in} + a'bC_{in}'
\]

Karnaugh map for \( C_{in} \)
\[
C_{out} = aC_{in} + bC_{in} + ab
\]
Full adder (2/2)

- S is an odd function of the 3 input bits
  \( S = a \oplus b \oplus C_{in} \)

- \( C_{out} = ab \ C_{in} + ab \ C_{in}' + ab' \ C_{in} + a'b \ C_{in} \)
  = \( ab + C_{in} (ab' + a'b) \)
  = \( ab + C_{in} (a \oplus b) \)

- Full Adder \( \Leftrightarrow \) 2 Half Adders and 1 OR gate
Binary Adder circuit

- Example: Binary addition of two 4-bit words A and B:
  - A = “1011” and B = “0011” S = A + B = “1110”

  The carries are connected in chain through the full adders → **Binary ripple carry adder**

- For a n-bit ripple carry adder, the longest propagation delay for S_n to settle to its steady-state is defined by the propagation of C_0 to C_n: 2n level of gates.
Binary ripple carry adder

2n level of gates $\Rightarrow$ 4-bit binary adder: 8 level of gates
Let's define two binary variables depending on inputs only:

- The *carry propagate*: $P_i = A_i \oplus B_i$
- The *carry generate*: $G_i = A_i B_i$

\[ S_i = P_i \oplus C_i \]
\[ C_{i+1} = P_i C_i + G_i \]

4-bit binary adder:

- $C_0 = \text{Input carry}$
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$
- $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$
- $C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$
Carry Lookahead Generator
The outputs $S_1$ to $S_3$ have equal propagation delay times.

The # of levels of gates is constant for any pair of bits to add: 4