Boolean Logic

Boolean algebra

Some elementary Boolean operators:
- Not(x)
- And(x, y)
- Or(x, y)
- Nand(x, y)

Boolean functions:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>( f(x, y, z) = (x + y)\overline{z} )</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>0</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>

- Functional expression VS truth table expression
- Important result: Every Boolean function can be expressed using And, Or, Not
All Boolean functions of 2 variables

<table>
<thead>
<tr>
<th>Function</th>
<th>x</th>
<th>y</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>And</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x And Not y</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Not x And y</td>
<td>y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Or</td>
<td>y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xor</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Boolean algebra

Given: Nand(a,b), false

- Not(a) = Nand(a,a)
- true = Not(false)
- And(a,b) = Not(Nand(a,b))
- Or(a,b) = Not(And(Not(a),Not(b)))
- Xor(a,b) = Or(a,Not(b),Not(a),b))
- Etc.
Gate logic

- Gate logic – a gate architecture designed to implement a Boolean function

- Elementary gates:
  - And
  - Or
  - Not

- Composite gates:
  - Interface VS implementation.

Claude Shannon, 1916-2001
("Symbolic Analysis of Relay and Switching Circuits")
Circuit implementations

- Physical realizations of logic gates are irrelevant to computer science.

Project 1: elementary logic gates

**Given:** Nand(a, b), false

**Build:**
- Not(a) = ...
- true = ...
- And(a, b) = ...
- Or(a, b) = ...
- Mux(a, b, sel) = ...
- Etc. - 12 gates altogether.
**Multiplexer**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>sel</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</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>1</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>0</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>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sel</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
</tbody>
</table>

- Implementation: based on Not, And, Or gates.

---

**Project 1 tips**

- Gate construction “work flow”: see the “building an And gate” slides from the intro lecture
- Course web site
- Download the TECS software suite
- Go through the hardware simulator
- You’re in business.
Suspect function (a-la-Leibnitz): Each suspect may or may not have an alibi \((a)\), a motivation to commit the crime \((m)\), and a relationship to the weapon found in the scene of the crime \((w)\). The police decides to focus attention only on suspects for whom the proposition \(\neg \(a\) And \((m Or w)\)\) is true.

Truth table of the "suspect" function \(s(a, m, w) = \overline{a} \cdot (m + w)\)

<table>
<thead>
<tr>
<th>(a)</th>
<th>(m)</th>
<th>(w)</th>
<th>(\text{truth})</th>
<th>(\text{truth}(a Or m Or w))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(m_0 = \overline{a} \cdot \overline{m} \cdot w)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(m_1 = \overline{a} \cdot \overline{m} \cdot w)</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(m_2 = \overline{a} \cdot m \cdot \overline{w})</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(m_3 = \overline{a} \cdot m \cdot w)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(m_4 = a \cdot \overline{m} \cdot \overline{w})</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(m_5 = a \cdot \overline{m} \cdot w)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(m_6 = a \cdot m \cdot \overline{w})</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(m_7 = a \cdot m \cdot w)</td>
<td>0</td>
</tr>
</tbody>
</table>

Canonical form: \(s(a, m, w) = \overline{a} \cdot m \cdot \overline{w} + \overline{a} \cdot m \cdot w\)

\(s(a, m, w) = \overline{a} \cdot (m + w)\)

\(s(a, m, w) = \overline{a} \cdot (m + w)\)

\(s(a, m, w) = \overline{a} \cdot m \cdot \overline{w} + \overline{a} \cdot m \cdot w\)

\(s(a, m, w) = \overline{a} \cdot m \cdot \overline{w} + \overline{a} \cdot m \cdot w\)
End notes: Programmable Logic Device for 3-way functions

- Each Boolean function has a canonical representation
- The canonical representation is expressed in terms of And, Not, Or
- And, Not, Or can be expressed in terms of Nand alone
- Ergo, every Boolean function can be realized by a standard PLD consisting of Nand gates only
- Mass production
- Universal building blocks, unique topology
- Gates, neurons, atoms, …