Lecture 3:
Gates

Name These Gates

A
B
C
D
E
Last Times

• Defined “and”, “or”, “not” gates by their truth tables.

• Explained how to construct them out of switches and relays.

• Gave an example of their use in the Nimbot.

• This time: Demonstrate their generality by providing more complex examples.

• But first... how do you make a bit?

How to Make a Gate

- switches/relays
- hydraulic valves
- tinkertoys
- silicon: semiconductors/transistors
- soap bubble

- DNA
- quantum material
- optics
- nanotubes
- neurons
- dominoes
- legos/marbles
Or Gate (v4)

Release bottom row first

Wire, transmitting bits (v1)
Could It Work?

- My domino “or” gate requires 24 dominoes.
- The first Pentium processor had 3.3M transistors, or roughly 800K gates.
- So, perhaps 19M dominoes needed.
- World record for domino toppling: 4M.
- Oh, and the Pentium did its computations 60M times a second, whereas dominoes might require a week to set up once.

A Few New Gates

```python
def AND3(A, B, C):
    x = A and B
    y = x and C
    return y
```

```python
def OR4(A, B, C, D):
    return (A or B) or (C or D)
```
def IFTTHENELSE5(bit, char1, char2):
    return [(char1[0] and bit) or (char2[0] and not bit),
            (char1[1] and bit) or (char2[1] and not bit),
            (char1[2] and bit) or (char2[2] and not bit),
            (char1[3] and bit) or (char2[3] and not bit),
            (char1[4] and bit) or (char2[4] and not bit)]

- Takes 11 bits as input and makes 5 as output. For clarity, the bits are grouped.
- char1[0] means the leftmost bit of the group called “char1”.
- “bit” selects char1 (True) or char2 (False).
Why “Or”, “And”, “Not”?  

- In addition to being familiar, these gates are “universal”. That is, all other logical functions can be expressed using these building blocks.
- How many distinct logic functions on 2 bits?

Some Truth Tables

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
### More Truth Tables

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

### Universal Gate

- Take two inputs, A and B.
- Take four more inputs defining what the gate should output for each combination of A and B.
- Output the right bit!
def UNIV2(A, B, d00, d01, d10, d11):
    w = AND3(not A, not B, d00)
    x = AND3(not A, B, d01)
    y = AND3(A, not B, d10)
    z = AND3(A, B, d11)
    return OR4(w, x, y, z)

def equal(bit1, bit2):
    return ((bit1 and bit2) or (not bit1 and not bit2))

• Output True if either bit1 = True and bit2 = True, or bit1 = False and bit2 = False (they are equal).
Group Equality

- Now that we can test two bits for equality, we would like to test a group of 5 bits for equality (two alphabetic characters).
- Two groups are equal if each of their bits are equal: bit 0 = bit 0, bit 1 = bit 1, etc.

```python
def equal5(char1, char2):
    return (equal(char1[0],char2[0])
            and (equal(char1[1],char2[1])
            and (equal(char1[2],char2[2])
            and (equal(char1[3],char2[3])
            and equal(char1[4],char2[4]))))
```

Equal5 Diagram
Gates in EQUAL5

• 10 inputs (2 groups of 5), 1 output bit.
• The equal5 gate consists of
  - 4 “and” gates
  - 5 “equal” gates
  - 2 “and”, 2 “not”, 1 “or” (5 total)
  - Total = 29 gates

Gates: Could Create

• And-k: k ins, 1 out (True if all ins are True)
• Or-k: k ins, 1 out (True if any ins are True)
• Ifthenelse-k: 1 control bit in, k then ins, k else ins, k outs (outs match then if control bit is True, else otherwise)
• Equal-k: 2 k-bit blocks in, 1 out (True if blocks same)
• Universal-k: 2^k table in, k control bits in, 1 out (equal to the value in the table specified by the control bits)
Counting Boolean Functions

• With 2 input bits, there are $2^2=4$ rows of the truth table (combinations of truth assignments to these variables).

• Each row can take an output of true or false, for a total of $2^4=16$ tables.

• For $n$ inputs: $2^n$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>65536</td>
</tr>
<tr>
<td>5</td>
<td>4294967296</td>
</tr>
<tr>
<td>6</td>
<td>18446744073709551616</td>
</tr>
<tr>
<td>7</td>
<td>340282366920938463463374607431768211456</td>
</tr>
</tbody>
</table>

Can Represent Them All

• Almost all multi-input functions require an enormous number of logic gates.

• However, the most useful ones can be represented succinctly.
Next Time

- Read Hillis, remainder of Chapter 2.