CPE100: Digital Logic Design I

Section 1004: Dr. Morris
From Zero to One
Background: Digital Logic Design

• How have digital devices changed the world?
• How have digital devices changed your life?
Background

- Digital Devices have revolutionized our world
  - Internet, cell phones, rapid advances in medicine, etc.
- The semiconductor industry has grown from $21 billion in 1985 to over $300 billion in 2015
The Game Plan

• Purpose of course:
  • Learn the principles of digital design
  • Learn to systematically debug increasingly complex designs
Chapter 1: Topics

• The Art of Managing Complexity
• The Digital Abstraction
• Number Systems
• Addition
• Binary Codes
• Signed Numbers
• Logic Gates
• Logic Levels
• CMOS Transistors
• Power Consumption
The Art of Managing Complexity

- Abstraction
- Discipline
- The Three –y’s
  - Hierarchy
  - Modularity
  - Regularity
Abstraction

- What is abstraction?
  - Hiding details when they are not important

- Electronic computer abstraction
  - Different levels with different building blocks

<table>
<thead>
<tr>
<th>Application Software</th>
<th>&quot;hello world!&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Systems</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Micro-architecture</td>
<td></td>
</tr>
<tr>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>Digital Circuits</td>
<td></td>
</tr>
<tr>
<td>Analog Circuits</td>
<td></td>
</tr>
<tr>
<td>Devices</td>
<td></td>
</tr>
<tr>
<td>Physics</td>
<td></td>
</tr>
</tbody>
</table>

- focus of this course
  - programs
  - device drivers
  - instructions
  - registers
  - datapaths
  - controllers
  - adders
  - memories
  - AND gates
  - NOT gates
  - amplifiers
  - filters
  - transistors
  - diodes
  - electrons
Discipline

• Intentionally restrict design choices

• Example: Digital discipline
  – Discrete voltages (0 V, 5 V) instead of continuous (0V – 5V)
  – Simpler to design than analog circuits – can build more sophisticated systems
  – Digital systems replacing analog predecessors:
    • i.e., digital cameras, digital television, cell phones, CDs
The Three –y’s

• Hierarchy
  • A system divided into modules and submodules

• Modularity
  • Having well-defined functions and interfaces

• Regularity
  • Encouraging uniformity, so modules can be easily reused
Example: Flintlock Rifle

- Hierarchy

- Three main modules: Lock, stock, and barrel

- Submodules of lock: Hammer, flint, frizzen, etc.
Example Flintlock Rifle

- **Modularity**
  - Function of stock: mount barrel and lock
  - Interface of stock: length and location of mounting pins

- **Regularity**
  - Interchangeable parts
The Art of Managing Complexity

• Abstraction
• Discipline
• The Three –y’s
  • Hierarchy
  • Modularity
  • Regularity
The Digital Abstraction

• Most physical variables are **continuous**
  • Voltage on a wire (1.33 V, 9 V, 12.2 V)
  • Frequency of an oscillation (60 Hz, 33.3 Hz, 44.1 kHz)
  • Position of mass (0.25 m, 3.2 m)
• Digital abstraction considers **discrete subset** of values
  • 0 V, 5 V
  • “0”, “1”
The Analytical Engine

- Designed by Charles Babbage from 1834 – 1871
- Considered to be the first digital computer
- Built from mechanical gears, where each gear represented a discrete value (0-9)
- Babbage died before it was finished
Digital Discipline: Binary Values

• Two discrete values
  • 1 and 0
    • 1 = TRUE = HIGH = ON
    • 0 = FALSE = LOW = OFF

• How to represent 1 and 0
  • Voltage levels, rotating gears, fluid levels, etc.

• Digital circuits use voltage levels to represent 1 and 0
  • Bit = binary digit
    • Represents the status of a digital signal (2 values)
Why Digital Systems?

• Easier to design
• Fast
• Can overcome noise
• Error detection/correction
George Boole, 1815-1864

- Born to working class parents
- Taught himself mathematics and joined the faculty of Queen’s College in Ireland
- Wrote An Investigation of the Laws of Thought (1854)
- Introduced binary variables
- Introduced the three fundamental logic operations: AND, OR, and NOT
Number Systems

- Decimal
  - Base 10
- Binary
  - Base 2
- Hexadecimal
  - Base 16
Decimal Numbers

- Base 10 (our everyday number system)

\[ 5374_{10} = 5 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 4 \times 10^0 \]
Binary Numbers

- Base 2 (computer number system)

\[1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\]

<table>
<thead>
<tr>
<th>One</th>
<th>Eight</th>
<th>One</th>
<th>Four</th>
<th>Zero</th>
<th>Two</th>
<th>One</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Base 2
Powers of Two

• \(2^0 = \)
• \(2^1 = \)
• \(2^2 = \)
• \(2^3 = \)
• \(2^4 = \)
• \(2^5 = \)
• \(2^6 = \)
• \(2^7 = \)

• \(2^8 = \)
• \(2^9 = \)
• \(2^{10} = \)
• \(2^{11} = \)
• \(2^{12} = \)
• \(2^{13} = \)
• \(2^{14} = \)
• \(2^{15} = \)
## Powers of Two

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- Handy to memorize up to $2^{10}$

- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$
Bits, Bytes, Nibbles ...

- **Bits**

- **Bytes** = 8 bits

- **Nibble** = 4 bits

- **Words** = 32 bits
  - Hex digit to represent nibble
Decimal to Binary Conversion

- Two Methods:
  - Method 1: Find largest power of 2 that fits, subtract and repeat
  - Method 2: Repeatedly divide by 2, remainder goes in next most significant bit
D2B: Method 1

• Find largest power of 2 that fits, subtract, repeat

$53_{10}$
D2B: Method 1

- Find largest power of 2 that fits, subtract, repeat

\[
\begin{align*}
53_{10} & \quad 32 \times 1 \\
53 - 32 & = 21 \quad 16 \times 1 \\
21 - 16 & = 5 \quad 4 \times 1 \\
5 - 4 & = 1 \quad 1 \times 1 \\
\end{align*}
\]

\[= 110101_2\]
D2B: Method 2

• Repeatedly divide by 2, remainder goes in next most significant bit

\[ 53_{10} = \]
D2B: Method 2

- Repeatedly divide by 2, remainder goes in next most significant bit

\[ 53_{10} = \frac{53}{2} = 26 \text{ R1} \]
\[ 26/2 = 13 \text{ R0} \]
\[ 13/2 = 6 \text{ R1} \]
\[ 6/2 = 3 \text{ R0} \]
\[ 3/2 = 1 \text{ R1} \]
\[ 1/2 = 0 \text{ R1} \]

\[ = 110101_{2} \]
Number Conversion

• Binary to decimal conversion
  • Convert $10011_2$ to decimal
    $16 \times 1 + 8 \times 0 + 4 \times 0 + 2 \times 1 + 1 \times 1 = 19_{10}$

• Decimal to binary conversion
  • Convert $47_{10}$ to binary
    $32 \times 1 + 16 \times 0 + 8 \times 1 + 4 \times 1 + 2 \times 1 + 1 \times 1 = 101111_2$
D2B Example

- Convert $75_{10}$ to binary
D2B Example

- Convert $75_{10}$ to binary

$$75_{10} = 64 + 8 + 2 + 1 = 1001011_2$$

- Or

$$\begin{align*}
75/2 &= 37 \quad \text{R1} \\
37/2 &= 18 \quad \text{R1} \\
18/2 &= 9 \quad \text{R0} \\
9/2 &= 4 \quad \text{R1} \\
4/2 &= 2 \quad \text{R0} \\
2/2 &= 1 \quad \text{R0} \\
1/2 &= 0 \quad \text{R1}
\end{align*}$$
Binary Values and Range

• N-digit decimal number
  • How many values?
  • Range?

• Example:
  3-digit decimal number
  • Possible values
  • Range
Binary Values and Range

• N-digit decimal number
  • How many values?
    • $10^N$
  • Range?
    • $[0, 10^N - 1]$

• Example:
  3-digit decimal number
  • Possible values
    • $10^3 = 1000$
  • Range
    • $[0, 999]$
Binary Values and Range

• N-bit binary number
  • How many values?
  • Range?

• Example:
  3-bit binary number
  • Possible values
  • Range
Binary Values and Range

- N-bit binary number
  - How many values?
    - $2^N$
  - Range?
    - $[0, 2^N - 1]$

- Example:
  3-bit binary number
  - Possible values
    - $2^3 = 8$
  - Range
    - $[0, 7] = [000_2, 111_2]$
Binary Values and Range

- N-digit decimal number
  - How many values?
    - \(10^N\)
  - Range?
    - \([0, 10^N - 1]\)

- Example:
  3-digit decimal number
  - Possible values
    - \(10^3 = 1000\)
  - Range
    - \([0, 999]\)

- N-bit binary number
  - How many values?
    - \(2^N\)
  - Range?
    - \([0, 2^N - 1]\)

- Example:
  3-bit binary number
  - Possible values
    - \(2^3 = 8\)
  - Range
    - \([0, 7] = [000_2, 111_2]\)
Hexadecimal Numbers

• Base 16 number system

• Shorthand for binary
  • Four binary digits (4-bit binary number) is a single hex digit
### Hexadecimal Numbers

<table>
<thead>
<tr>
<th>Hex Digit</th>
<th>Decimal Equivalent</th>
<th>Binary Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
# Hexadecimal Numbers

<table>
<thead>
<tr>
<th>Hex Digit</th>
<th>Decimal Equivalent</th>
<th>Binary Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>
Hexadecimal to Binary Conversion

• Hexadecimal to binary conversion:
  • Convert $4AF_{16}$ (also written $0x4AF$) to binary

• Hexadecimal to decimal conversion:
  • Convert $0x4AF$ to decimal
Hexadecimal to Binary Conversion

• Hexadecimal to binary conversion:
  • Convert $4AF_{16}$ (also written $0x4AF$) to binary
    • $0x4AF = 0100\ 1010\ 1111_2$

• Hexadecimal to decimal conversion:
  • Convert $0x4AF$ to decimal
    • $4 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 1199_{10}$
Number Systems

• Popular
  • Decimal Base 10
  • Binary Base 2
  • Hexadecimal Base 16

• Others
  • Octal Base 8
  • Any other base
Octal Numbers

- Same as hex with one less binary digit

<table>
<thead>
<tr>
<th>Octal Digit</th>
<th>Decimal Equivalent</th>
<th>Binary Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>
Number Systems

• In general, an N-digit number \( \{a_{N-1}a_{N-2} \ldots a_1a_0\} \) of base \( R \) in decimal equals
  
  \[ a_{N-1}R^{N-1} + a_{N-2}R^{N-2} + \cdots + a_1R^1 + a_0R^0 \]

• Example: 4-digit \( \{5173\} \) of base 8 (octal)
Number Systems

• In general, an N-digit number \( \{a_{N-1}a_{N-2} \ldots a_1a_0\} \) of base \( R \) in decimal equals
  \[
  a_{N-1}R^{N-1} + a_{N-2}R^{N-2} + \cdots + a_1R^1 + a_0R^0
  \]

• Example: 4-digit \( \{5173\} \) of base 8 (octal)
  \[
  5 \times 8^3 + 1 \times 8^2 + 7 \times 8^1 + 3 \times 8^0 = 2683_{10} \]
Decimal to Octal Conversion

- Remember two methods for D2B conversion
  - 1: remove largest multiple; 2: repeated divide
- Convert $29_{10}$ to octal
Decimal to Octal Conversion

• Remember two methods for D2B conversion
  • 1: remove largest multiple; 2: repeated divide
• Convert $29_{10}$ to octal
• Method 2

\[
\begin{align*}
29 / 8 &= 3 \quad \text{R5} \quad \text{lsb} \\
3 / 8 &= 0 \quad \text{R3} \quad \text{msb}
\end{align*}
\]

$29_{10} = 35_8$
Decimal to Octal Conversion

• Remember two methods for D2B conversion
  • 1: remove largest multiple; 2: repeated divide
• Convert $29_{10}$ to octal
• Method 1

\[
\begin{align*}
29 & \quad 8 \times 3 = 24 \\
29 - 24 & = 5 \\
\end{align*}
\]

\[
29_{10} = 24 + 5 = 3 \times 8^1 + 5 \times 8^0 = 35_8
\]

• Or (better scalability)

\[
29_{10} = 16 + 8 + 4 + 1 = 11101_2 = 35_8
\]
Octal to Decimal Conversion

• Convert $163_8$ to decimal
Octal to Decimal Conversion

• Convert $163_8$ to decimal

  • $163_8 = 1 \times 8^2 + 6 \times 8^1 + 3$
  • $163_8 = 64 + 48 + 3$
  • $163_8 = 115_{10}$
Recap: Binary and Hex Numbers

• Example 1: Convert $83_{10}$ to hex

• Example 2: Convert $01101011_2$ to hex and decimal

• Example 3: Convert $0xCA3$ to binary and decimal
Recap: Binary and Hex Numbers

- **Example 1:** Convert $83_{10}$ to hex
  - $83_{10} = 64 + 16 + 2 + 1 = 1010011_2$
  - $1010011_2 = 101 0011_2 = 53_{16}$

- **Example 2:** Convert $01101011_2$ to hex and decimal
  - $01101011_2 = 0110 1011_2 = 6B_{16}$
  - $0x6B = 6 \times 16^1 + 11 \times 16^0 = 96 + 11 = 107$

- **Example 3:** Convert $0xCA3$ to binary and decimal
  - $0xCA3 = 1100 1010 0011_2$
  - $0xCA3 = 12 \times 16^2 + 10 \times 16^1 + 3 \times 16^0 = 3235_{10}$
Large Powers of Two

- $2^{10} = 1 \text{kilo} \approx 1000 (1024)$
- $2^{20} = 1 \text{mega} \approx 1 \text{million} \ (1,048,576)$
- $2^{30} = 1 \text{giga} \approx 1 \text{billion} \ (1,073,741,824)$
- $2^{40} = 1 \text{tera} \approx 1 \text{trillion} \ (1,099,511,627,776)$
Large Powers of Two: Abbreviations

- $2^{10} = 1 \text{ kilo} \approx 1000$ (1024)
  
  for example: 1 kB = 1024 Bytes
  1 kb = 1024 bits

- $2^{20} = 1 \text{ mega} \approx 1 \text{ million}$ (1,048,576)
  
  for example: 1 MiB, 1 Mib (1 megabit)

- $2^{30} = 1 \text{ giga} \approx 1 \text{ billion}$ (1,073,741,824)
  
  for example: 1 GiB, 1 Gib
Estimating Powers of Two

• What is the value of $2^{24}$?

• How many values can a 32-bit variable represent?
Estimating Powers of Two

- What is the value of $2^{24}$?
  - $2^4 \times 2^{20} \approx 16$ million

- How many values can a 32-bit variable represent?
  - $2^2 \times 2^{30} \approx 4$ billion
Binary Codes

Another way of representing decimal numbers in binary

**Example binary codes:**
- Weighted codes
  - Binary Coded Decimal (BCD) (8-4-2-1 code)
  - 6-3-1-1 code
  - 8-4-2-1 code (simple binary)
- Gray codes
- Excess-3 code
- 2-out-of-5 code
## ASCII Code

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>1 0 1 0 1 0</td>
<td>`</td>
<td>1 1 0 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>1 0 1 0 0 0</td>
<td>A</td>
<td>1 0 1 0 0 1</td>
<td>a</td>
<td>1 1 0 0 0 1</td>
</tr>
<tr>
<td>&quot;</td>
<td>1 0 1 0 0 1</td>
<td>B</td>
<td>1 0 0 1 0 0</td>
<td>b</td>
<td>1 1 0 0 0 1</td>
</tr>
<tr>
<td>#</td>
<td>1 0 1 0 1 0</td>
<td>C</td>
<td>1 0 0 1 0 1</td>
<td>c</td>
<td>1 1 0 0 0 1</td>
</tr>
<tr>
<td>$</td>
<td>1 0 1 0 1 1</td>
<td>D</td>
<td>1 0 0 1 1 0</td>
<td>d</td>
<td>1 1 0 0 1 0</td>
</tr>
<tr>
<td>%</td>
<td>1 0 1 0 0 0</td>
<td>E</td>
<td>1 0 0 1 0 1</td>
<td>e</td>
<td>1 1 0 0 1 0</td>
</tr>
<tr>
<td>&amp;</td>
<td>1 0 1 0 1 1</td>
<td>F</td>
<td>1 0 0 1 1 1</td>
<td>f</td>
<td>1 1 0 1 0 0</td>
</tr>
<tr>
<td>(</td>
<td>1 0 1 0 0 1</td>
<td>G</td>
<td>1 0 1 0 0 0</td>
<td>g</td>
<td>1 1 0 1 0 0</td>
</tr>
<tr>
<td>)</td>
<td>1 0 1 0 1 0</td>
<td>H</td>
<td>1 0 1 0 0 1</td>
<td>h</td>
<td>1 1 0 1 0 1</td>
</tr>
<tr>
<td>*</td>
<td>1 0 1 0 1 1</td>
<td>I</td>
<td>1 0 1 0 1 0</td>
<td>i</td>
<td>1 1 0 1 0 1</td>
</tr>
<tr>
<td>+</td>
<td>1 0 1 1 0 1</td>
<td>J</td>
<td>1 0 1 0 0 0</td>
<td>j</td>
<td>1 1 0 1 0 1</td>
</tr>
<tr>
<td>,</td>
<td>1 0 1 0 0 0</td>
<td>K</td>
<td>1 0 1 0 1 0</td>
<td>k</td>
<td>1 1 0 1 1 0</td>
</tr>
<tr>
<td>.</td>
<td>1 0 1 0 1 0</td>
<td>L</td>
<td>1 0 1 0 0 1</td>
<td>l</td>
<td>1 1 0 1 1 1</td>
</tr>
<tr>
<td>/</td>
<td>1 0 1 0 1 1</td>
<td>M</td>
<td>1 0 1 0 1 0</td>
<td>m</td>
<td>1 1 0 1 1 1</td>
</tr>
<tr>
<td>0</td>
<td>1 0 1 0 0 1</td>
<td>N</td>
<td>1 0 1 0 1 0</td>
<td>n</td>
<td>1 1 0 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1 0 0 0</td>
<td>O</td>
<td>1 0 1 0 1 1</td>
<td>o</td>
<td>1 1 0 1 1 1</td>
</tr>
<tr>
<td>2</td>
<td>1 0 1 0 1 0</td>
<td>P</td>
<td>1 0 1 1 0 0</td>
<td>p</td>
<td>1 1 1 0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>1 0 1 1 0 0</td>
<td>Q</td>
<td>1 0 1 1 0 1</td>
<td>q</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>4</td>
<td>1 0 1 1 0 1</td>
<td>R</td>
<td>1 0 1 1 1 0</td>
<td>r</td>
<td>1 1 1 0 1 0</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1 1 0</td>
<td>S</td>
<td>1 0 1 1 1 1</td>
<td>s</td>
<td>1 1 1 0 1 1</td>
</tr>
<tr>
<td>6</td>
<td>1 0 1 1 1 1</td>
<td>T</td>
<td>1 0 1 1 0 0</td>
<td>t</td>
<td>1 1 1 0 1 0</td>
</tr>
<tr>
<td>7</td>
<td>1 0 1 1 0 0</td>
<td>U</td>
<td>1 0 1 1 0 1</td>
<td>u</td>
<td>1 1 1 0 1 0</td>
</tr>
<tr>
<td>8</td>
<td>1 0 1 1 0 1</td>
<td>V</td>
<td>1 0 1 1 1 0</td>
<td>v</td>
<td>1 1 1 0 1 1</td>
</tr>
<tr>
<td>9</td>
<td>1 0 1 1 1 0</td>
<td>W</td>
<td>1 0 1 1 1 1</td>
<td>w</td>
<td>1 1 1 0 1 1</td>
</tr>
<tr>
<td>:</td>
<td>1 0 1 1 1 0</td>
<td>X</td>
<td>1 0 1 1 0 0</td>
<td>x</td>
<td>1 1 1 0 0 0</td>
</tr>
<tr>
<td>;</td>
<td>1 0 1 1 1 1</td>
<td>Y</td>
<td>1 0 1 0 0 0</td>
<td>y</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>&lt;</td>
<td>1 0 1 1 0 1</td>
<td>Z</td>
<td>1 0 1 0 0 1</td>
<td>z</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>=</td>
<td>1 0 1 1 1 0</td>
<td>[</td>
<td>1 0 1 1 0 1</td>
<td>[</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>&gt;</td>
<td>1 0 1 1 1 1</td>
<td>\</td>
<td>1 0 1 1 1 0</td>
<td>\</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>?</td>
<td>1 0 1 1 1 0</td>
<td>]</td>
<td>1 0 1 1 1 1</td>
<td>]</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>?</td>
<td>1 0 1 1 1 1</td>
<td>^</td>
<td>1 0 1 1 1 0</td>
<td>^</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td>~</td>
<td>1 0 1 1 1 0</td>
<td>_</td>
<td>1 0 1 1 1 1</td>
<td>_</td>
<td>1 1 1 0 0 1</td>
</tr>
<tr>
<td></td>
<td>1 1 1 1 1 1</td>
<td>delete</td>
<td>1 1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Cengage Learning 2014
# Binary Codes

Each code combination represents a single decimal digit.

<table>
<thead>
<tr>
<th>Decimal #</th>
<th>8-4-2-1 (BCD)</th>
<th>6-3-1-1</th>
<th>Excess-3</th>
<th>2-out-of-5</th>
<th>Gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>0011</td>
<td>00011</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
<td>0100</td>
<td>00101</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
<td>0101</td>
<td>00110</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0100</td>
<td>0110</td>
<td>01001</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0101</td>
<td>0111</td>
<td>01010</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0111</td>
<td>1000</td>
<td>01100</td>
<td>1110</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>1000</td>
<td>1001</td>
<td>10001</td>
<td>1010</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>1001</td>
<td>1010</td>
<td>10010</td>
<td>1011</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1011</td>
<td>1011</td>
<td>10100</td>
<td>1001</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1100</td>
<td>1100</td>
<td>11000</td>
<td>1000</td>
</tr>
</tbody>
</table>
Gray Codes

- Next number differs in only one bit position
  - **Example:** 000, 001, 011, 010, 110, 111, 101, 100

- **Example use:** Analog-to-Digital (A/D) converters. Changing 2 bits at a time (i.e., 011 → 100) could cause large inaccuracies.

- Will use in K-maps

<table>
<thead>
<tr>
<th>Decimal #</th>
<th>Gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>1110</td>
</tr>
<tr>
<td>6</td>
<td>1010</td>
</tr>
<tr>
<td>7</td>
<td>1011</td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
</tr>
</tbody>
</table>
Addition

• Decimal

\[
\begin{array}{c}
3734 \\
+ 5168 \\
\hline
5168
\end{array}
\]

• Binary

\[
\begin{array}{c}
1011 \\
+ 0011 \\
\hline
0011
\end{array}
\]
Addition

- Decimal

  \[
  \begin{array}{c}
  3734 \\
  + 5168 \\
  \hline
  8902
  \end{array}
  \]

  \[11 \xrightarrow{\text{carries}}\]

- Binary

  \[
  \begin{array}{c}
  1011 \\
  + 0011 \\
  \hline
  1011
  \end{array}
  \]
Addition

- Decimal

  \[
  \begin{array}{c}
  3734 \\
  \text{+} \\
  5168 \\
  \hline
  8902
  \end{array}
  \]

  \[
  \begin{array}{c}
  11 \xleftarrow{\text{carries}}
  \end{array}
  \]

- Binary

  \[
  \begin{array}{c}
  1011 \\
  \text{+} \\
  0011 \\
  \hline
  1110
  \end{array}
  \]

  \[
  \begin{array}{c}
  11 \xleftarrow{\text{carries}}
  \end{array}
  \]
Binary Addition Examples

• Add the following 4-bit binary numbers

  1001
  + 0101
  _______  
  1010

• Add the following 4-bit binary numbers

  1011
  + 0110
  _______  
  1101
Binary Addition Examples

- Add the following 4-bit binary numbers:

  \[
  \begin{array}{c}
  1001 \\
  + 0101 \\
  \hline
  1110
  \end{array}
  \]

- Add the following 4-bit binary numbers:

  \[
  \begin{array}{c}
  1011 \\
  + 0110 \\
  \hline
  1010
  \end{array}
  \]
Binary Addition Examples

• Add the following 4-bit binary numbers

\[
\begin{array}{ccc}
1001 & + & 0101 \\
\hline
1110
\end{array}
\]

Overflow!

• Add the following 4-bit binary numbers

\[
\begin{array}{ccc}
111 & + & 0110 \\
\hline
10001
\end{array}
\]

Overflow!
Overflow

- Digital systems operate on a **fixed number of bits**
- Overflow: when result is too big to fit in the available number of bits
- See previous example of $11 + 6$
Signed Binary Numbers

- Sign/Magnitude Numbers
- Two’s Complement Numbers
Sign/Magnitude

- 1 sign bit, \( N-1 \) magnitude bits
- Sign bit is the most significant (left-most) bit
  - Positive number: sign bit = 0
  - Negative number: sign bit = 1

\[
A = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} a_i 2^i
\]

- Example, 4-bit sign/magnitude representations of ±6:
  - +6 = 
  - -6 =

- Range of an \( N \)-bit sign/magnitude number:
Sign/Magnitude

- 1 sign bit, \( N-1 \) magnitude bits
- Sign bit is the most significant (left-most) bit
  - **Positive number**: sign bit = 0
  - **Negative number**: sign bit = 1

\[
A = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} a_i 2^i
\]

- Example, 4-bit sign/magnitude representations of ±6:
  - \(+6 = 0110\)
  - \(-6 = 1110\)

- Range of an \( N \)-bit sign/magnitude number:
  - \([-\left(2^{N-1}-1\right), 2^{N-1}-1]\)
Sign/Magnitude Numbers

• Problems:
  • Addition doesn’t work, for example -6 + 6:
    \[
    \begin{array}{c}
    1110 \\
    + 0110 \\
    \hline
    1010
    \end{array}
    \]
  • Two representations of 0 (± 0):
    • (+0) =
    • (−0) =
Sign/Magnitude Numbers

• Problems:
  • Addition doesn’t work, for example $-6 + 6$:
    $\begin{align*}
    1 & 1 1 0 \\
    + & 0 1 1 0 \\
    \hline
    1 & 0 1 0 0 \quad (\text{wrong!})
    \end{align*}$
  • Two representations of 0 ($\pm 0$):
    • $(+0) = 0000$
    • $(−0) = 1000$
Two’s Complement Numbers

- Don’t have same problems as sign/magnitude numbers:
  - Addition works
  - Single representation for 0

- Range of representable numbers not symmetric
  - One extra negative number
Two’s Complement Numbers

- msb has value of $-2^{N-1}$

\[ A = a_{n-1} \left( -2^{n-1} \right) + \sum_{i=0}^{n-2} a_i 2^i \]

- The most significant bit still indicates the sign (1 = negative, 0 = positive)

- Range of an $N$-bit two’s comp number?

- Most positive 4-bit number?

- Most negative 4-bit number?
Two’s Complement Numbers

• msb has value of $-2^{N-1}$

$$A = a_{n-1}(-2^{n-1}) + \sum_{i=0}^{n-2} a_i 2^i$$

• The most significant bit still indicates the sign (1 = negative, 0 = positive)

• Range of an $N$-bit two’s comp number?
  • $[-(2^{N-1}), 2^{N-1} - 1]$  

• Most positive 4-bit number? 0111  

• Most negative 4-bit number? 1000
“Taking the Two’s Complement”

- **Flips the sign** of a two’s complement number
- **Method:**
  1. Invert the bits
  2. Add 1
- **Example:** Flip the sign of $3_{10} = 0011_2$
“Taking the Two’s Complement”

• Flips the sign of a two’s complement number

• Method:
  1. Invert the bits
  2. Add 1

• Example: Flip the sign of $3_{10} = 0011_2$
  1. $1100$
  2. $+ 1$
     $1101 = -3_{10}$
Two’s Complement Examples

• Take the two’s complement of $6_{10} = 0110_2$

• What is the decimal value of the two’s complement number $1001_2$?
Two’s Complement Examples

- Take the two’s complement of $6_{10} = 0110_2$
  1. 1001
  2. $+1$
  $1010_2 = -6_{10}$

- What is the decimal value of the two’s complement number $1001_2$?
  1. 0110
  2. $+1$
  $0111_2 = 7_{10}$, so $1001_2 = -7_{10}$
Two’s Complement Addition

- Add $6 + (-6)$ using two’s complement numbers

\[
\begin{array}{c}
0110 \\
+ 1010 \hline
\end{array}
\]

- Add $-2 + 3$ using two’s complement numbers

\[
\begin{array}{c}
1110 \\
+ 0011 \hline
\end{array}
\]
Two’s Complement Addition

- Add 6 + (-6) using two’s complement numbers

\[
\begin{array}{c}
111 \\
0110 \\
+ 1010 \\
\hline
10000
\end{array}
\]

- Add -2 + 3 using two’s complement numbers

\[
\begin{array}{c}
1110 \\
+ 0011 \\
\hline
10000
\end{array}
\]
Two’s Complement Addition

• Add 6 + (-6) using two’s complement numbers

\[
\begin{array}{c}
111 \\
0110 \\
+ 1010 \\
\hline
10000
\end{array}
\]

• Add -2 + 3 using two’s complement numbers

\[
\begin{array}{c}
111 \\
1110 \\
+ 0011 \\
\hline
10001
\end{array}
\]
Increasing Bit Width

• Extend number from N to M bits \((M > N)\):
  • Sign-extension
  • Zero-extension
Sign-Extension

- Sign bit copied to msb’s
- Number value is same

- Example 1
  - 4-bit representation of 3 = 0011
  - 8-bit sign-extended value:

- Example 2
  - 4-bit representation of -7 = 1001
  - 8-bit sign-extended value:
Sign-Extension

• Sign bit copied to msb’s
• Number value is same

• Example 1
  • 4-bit representation of 3 = 0011
  • 8-bit sign-extended value: 00000011

• Example 2
  • 4-bit representation of -7 = 1001
  • 8-bit sign-extended value: 11111001
Zero-Extension

• Zeros copied to msb’s
• Value changes for negative numbers

• Example 1
  • 4-bit value = \(0011_2\)
  • 8-bit zero-extended value:

• Example 2
  • 4-bit value = \(1001\)
  • 8-bit zero-extended value:
Zero-Extension

- Zeros copied to msb’s
- Value changes for negative numbers

- Example 1
  - 4-bit value = 0011\textsubscript{2}
  - 8-bit zero-extended value: 00000011

- Example 2
  - 4-bit value = 1001
  - 8-bit zero-extended value: 00001001
Zero-Extension

- Zeros copied to msb’s
- Value changes for negative numbers

- Example 1
  - 4-bit value = \(0011_2 = 3_{10}\)
  - 8-bit zero-extended value: \(00000011 = 3_{10}\)

- Example 2
  - 4-bit value = \(1001 = -7_{10}\)
  - 8-bit zero-extended value: \(00001001 = 9_{10}\)
## Number System Comparison

<table>
<thead>
<tr>
<th>Number System</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned</td>
<td>$[0, 2^{N-1}]$</td>
</tr>
<tr>
<td>Sign/Magnitude</td>
<td>$[-(2^{N-1}-1), 2^{N-1}-1]$</td>
</tr>
<tr>
<td>Two’s Complement</td>
<td>$[-2^{N-1}, 2^{N-1}-1]$</td>
</tr>
</tbody>
</table>

For example, 4-bit representation:

- **Unsigned**
  - $0000$ to $0111$
  - $1000$ to $1111$

- **Two’s Complement**
  - $1000$ to $1111$
  - $0000$ to $1111$

- **Sign/Magnitude**
  - $0000$ to $1111$
  - $1000$ to $1111$
Logic Gates

- **Perform logic functions:**
  - inversion (NOT), AND, OR, NAND, NOR, etc.
- **Single-input:**
  - NOT gate, buffer
- **Two-input:**
  - AND, OR, XOR, NAND, NOR, XNOR
- **Multiple-input**
Single-Input Logic Gates

### NOT

![NOT Gate Diagram]

\[ Y = \overline{A} \]

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

### BUF

![BUF Gate Diagram]

\[ Y = A \]

<table>
<thead>
<tr>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
• Bubble on wire indicates inversion

\[ \text{NOT} \]

\[ Y = \overline{A} \]

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

\[ \text{BUF} \]

\[ Y = A \]

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

• Note: bar over variable indicates complement (invert value)
Two-Input Logic Gates

**AND**

\[ Y = AB \]

<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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**OR**

\[ Y = A + B \]

<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>
Two-Input Logic Gates

**AND**

\[ Y = AB \]

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

**OR**

\[ Y = A + B \]

\[
\begin{array}{ccc}
A & B & Y \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
More Two-Input Logic Gates

### XOR

\[ Y = A \oplus B \]

<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>0</td>
</tr>
</tbody>
</table>

### NAND

\[ Y = \overline{A B} \]

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

### NOR

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

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

### XNOR

\[ Y = A \oplus B \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
## More Two-Input Logic Gates

### XOR

<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>0</td>
</tr>
</tbody>
</table>

\[ Y = A \oplus B \]

![XOR gate](image)

### NAND

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

\[ Y = \overline{A} \overline{B} \]

![NAND gate](image)

### NOR

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

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

![NOR gate](image)

### XNOR

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

\[ Y = \overline{A} \oplus \overline{B} \]

![XNOR gate](image)
Multiple-Input Logic Gates

**NOR3**

\[ Y = \overline{A + B + C} \]

<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>0</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>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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**AND3**

\[ Y = ABC \]

<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>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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Multiple-Input Logic Gates

NOR3

\[ Y = \overline{A + B + C} \]

\[
\begin{array}{ccc|c}
A & B & C & Y \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
\end{array}
\]

AND3

\[ Y = ABC \]

\[
\begin{array}{ccc|c}
A & B & C & Y \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

- Multi-input XOR = Odd parity (#on inputs odd → 1)
Logic Levels

- Discrete voltages represent 1 and 0
- For example:
  - $0 = \text{ground (GND) or 0 volts}$
  - $1 = V_{DD}$ or 5 volts
- What about 4.99 volts? Is that a 0 or a 1?
- What about 3.2 volts?
Logic Levels

• Must have *range* of voltages for 1 and 0

• Different ranges for inputs and outputs to allow for *noise*
What is Noise?

• Anything that degrades the signal
  • E.g., resistance, power supply noise, coupling to neighboring wires, etc.

• Example: a gate (driver) outputs 5 V but, because of resistance in a long wire, receiver gets 4.5 V
The Static Discipline

- With logically valid inputs, every circuit element must produce logically valid outputs.

- Use limited ranges of voltages to represent discrete values.
Real Logic Levels

- Want driver to output “clean” high/low and receiver to handle noisy high/low
Real Logic Levels

- Want driver to output “clean” high/low and receiver to handle noisy high/low
Real Logic Levels

Driver Receiver

Output Characteristics

Input Characteristics

Logic High Output Range

Logic Low Output Range

Logic High Input Range

Logic Low Input Range

\[ NM_H = V_{OH} - V_{IH} \]

\[ NM_L = V_{IL} - V_{OL} \]
V_{DD} Scaling

- In 1970’s and 1980’s, V_{DD} = 5 V
- V_{DD} has dropped
  - 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V, 1.0 V, ...

- Avoid frying tiny transistors
- Save power

- Be careful connecting chips with different supply voltages
  - Easy to fry if not careful
### Logic Family Examples

<table>
<thead>
<tr>
<th>Logic Family</th>
<th>$V_{DD}$</th>
<th>$V_{IL}$</th>
<th>$V_{IH}$</th>
<th>$V_{OL}$</th>
<th>$V_{OH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTL</td>
<td>5 (4.75 - 5.25)</td>
<td>0.8</td>
<td>2.0</td>
<td>0.4</td>
<td>2.4</td>
</tr>
<tr>
<td>CMOS</td>
<td>5 (4.5 - 6)</td>
<td>1.35</td>
<td>3.15</td>
<td>0.33</td>
<td>3.84</td>
</tr>
<tr>
<td>LVTTL</td>
<td>3.3 (3 - 3.6)</td>
<td>0.8</td>
<td>2.0</td>
<td>0.4</td>
<td>2.4</td>
</tr>
<tr>
<td>LVCMOS</td>
<td>3.3 (3 - 3.6)</td>
<td>0.9</td>
<td>1.8</td>
<td>0.36</td>
<td>2.7</td>
</tr>
</tbody>
</table>
Transistors

• Logic gates built from transistors
• Simple model: 3-ported voltage-controlled switch
  • 2 ports connected depending on voltage of 3rd
  • d and s are connected (ON) when g is 1
Robert Noyce, 1927-1990

- Nicknamed “Mayor of Silicon Valley”
- Cofounded Fairchild Semiconductor in 1957
- Cofounded Intel in 1968
- Co-invented the integrated circuit
Silicon

- Transistors built from silicon, a semiconductor
- Pure silicon is a poor conductor (no free charges)
- Doped silicon is a good conductor (free charges)
  - n-type (free negative charges, electrons)
  - p-type (free positive charges, holes)
MOS Transistors

- Metal oxide silicon (MOS) transistors:
  - Polysilicon (used to be metal) gate
  - Oxide (silicon dioxide) insulator
  - Doped silicon

![Diagram of nMOS transistor]

- nMOS
- Polysilicon
- SiO₂
- Source
- Drain
- Gate
- Substrate
- p
- n
nMOS Transistors

- Gate = 0
- OFF (no connection between source and drain)

- Gate = 1
- ON (channel between source and drain)

Diode connection from p to n doped area → current cannot travel from n → p
pMOS Transistors

- pMOS transistor is opposite of nMOS
  - ON when Gate = 0
  - OFF when Gate = 1
Transistor Function

- Voltage controlled switch

nMOS

pMOS

\[ g = 0 \]

\[ g = 1 \]
Transistor Composition

- **nMOS**: pass good 0’s
  - Connect source to GND
  - “Pull down” transistor

- **pMOS**: pass good 1’s
  - Connect source to VDD
  - “Pull up” transistor

- Build logic gates from composition
  - CMOS = complementary MOS
CMOS Gate Structure

- Pull-up pMOS network connects to $V_{DD}$
- Pull-down nMOS network connects to $GND$
- Use series and parallel connections to implement gate logic
CMOS Gates: NOT Gate

\[
Y = \overline{A}
\]

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

<table>
<thead>
<tr>
<th>A</th>
<th>P1</th>
<th>N1</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CMOS Gates: NOT Gate

\[
\begin{array}{c|c}
A & Y \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

\[
Y = \overline{A}
\]

\[
\begin{array}{c|c|c|c}
A & P1 & N1 & Y \\
\hline
0 & ON & OFF & 1 \\
1 & OFF & ON & 0 \\
\end{array}
\]

\[
\text{NOT} \quad V_{DD} \\
\quad A \quad Y \\
\quad \text{N1} \\
\quad \text{GND}
\]
CMOS Gates: NAND Gate

**NAND**

\[ Y = \overline{AB} \]

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

**Truth Table**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P1</th>
<th>P2</th>
<th>N1</th>
<th>N2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# CMOS Gates: NAND Gate

## NAND Gate

The NAND gate is a digital logic gate that outputs false only when both inputs are true. It is the complement of the OR gate. The symbol for a NAND gate is shown below.

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

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

## Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P1</th>
<th>P2</th>
<th>N1</th>
<th>N2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>0</td>
</tr>
</tbody>
</table>
CMOS Gates: NOR Gate

• How can you build three input \((A, B, C)\) NOR gate?
• How can you build three input \((A, B, C)\) NOR gate?

Only high output when all three pMOS in series are “on” and create a path from output to \(V_{DD}\).
CMOS Gates: AND Gate

- How can you build 2 input AND gate?
CMOS Gates: AND Gate

- How can you build a 2 input AND gate?

![Diagram of AND gate]
CMOS Gates: AND Gate

- How can you build a 2-input AND gate?

![Diagram of a 2-input AND gate](image)

Note: AND requires 2 more gates than NAND. Inverted logic is more efficient implementation.
Transmission Gates

- nMOS pass 1’s poorly, pMOS pass 0’s poorly
- Transmission gate is for passing signal
  - Pass both 0 and 1 well
- When EN = 1, the switch is ON:
  - \( \overline{EN} = 0 \) and A is connected to B
- When EN = 0, the switch is OFF:
  - A is not connected to B
Psuedo-nMOS

- Replace pull-up network with weak pMOS transistor that is always on
  - pMOS gate tied to ground
- pMOS transistor: pulls output HIGH only when nMOS network not pulling it LOW
Psuedo-nMOS Example: NOR4

- How many transistors needed?
Psuedo-nMOS Example: NOR4

- How many transistors needed?
- Only 5 since a single pMOS is used
Gordon Moore, 1929- 

• Cofounded Intel in 1968 with Robert Noyce.

• Moore’s Law: number of transistors on a computer chip doubles every year (observed in 1965)
  • Since 1975, transistor counts have doubled every two years.
Moore’s Law

- Transistor count doubles every 2 years
“If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get one million miles to the gallon, and explode once a year . . .”

– Robert Cringley
Power Consumption

- Power = Energy consumed per unit time

- Two types of power
  - Dynamic power consumption
  - Static power consumption
Dynamic Power Consumption

- Power to charge transistor gate capacitances
  - Energy required to charge a capacitance, $C$, to $V_{DD}$ is $CV_{DD}^2$
  - Circuit running at frequency $f$: transistors switch (from 1 to 0 or vice versa) at that frequency
  - Capacitor is charged $f/2$ times per second (discharging from 1 to 0 is free)

- Dynamic power consumption
  $$P_{\text{dynamic}} = \frac{1}{2} CV_{DD}^2 f$$
Static Power Consumption

• Power consumed when no gates are switching

• Caused by the quiescent supply current, $I_{DD}$ (also called the leakage current)

• Static power consumption

\[ P_{\text{static}} = I_{DD} V_{DD} \]
Power Consumption Example

- Estimate the power consumption of a wireless handheld computer
  - $V_{DD} = 1.2 \text{ V}$
  - $C = 20 \text{ nF}$
  - $f = 1 \text{ GHz}$
  - $I_{DD} = 20 \text{ mA}$

- Total power is sum of dynamic and static
Power Consumption Example

- Estimate the power consumption of a wireless handheld computer
  - $V_{DD} = 1.2 \text{ V}$
  - $C = 20 \text{ nF}$
  - $f = 1 \text{ GHz}$
  - $I_{DD} = 20 \text{ mA}$

- Total power is sum of dynamic and static

$$P = \frac{1}{2} CV_{DD}^2 f + I_{DD} V_{DD}$$

$$= \frac{1}{2} \left(20 \text{ nF}\right)(1.2)^2(1 \text{ GHz})$$

$$+ (20 \text{ mA})(1.2)$$

$$= (14.4 + 0.024) \text{ W}$$

$$= 14.4 \text{ W}$$
Binary Codes

Another way of representing decimal numbers

Example binary codes:

- Weighted codes
  - Binary Coded Decimal (BCD) (8-4-2-1 code)
  - 6-3-1-1 code
  - 8-4-2-1 code (simple binary)
- Gray codes
- Excess-3 code
- 2-out-of-5 code
# Binary Codes

Each code combination represents a **single decimal digit**.
### ASCII Code

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>0 1 0 0 0 0 0</td>
<td>@</td>
<td>1 0 0 0 0 0 0</td>
<td>,</td>
<td>1 1 0 0 0 0 0</td>
</tr>
<tr>
<td>!</td>
<td>0 1 0 0 0 0 1</td>
<td>A</td>
<td>1 0 0 0 0 0 1</td>
<td>a</td>
<td>1 1 0 0 0 0 1</td>
</tr>
<tr>
<td>“</td>
<td>0 1 0 0 0 1 1</td>
<td>B</td>
<td>1 0 0 0 0 1 1</td>
<td>b</td>
<td>1 1 0 0 0 1 1</td>
</tr>
<tr>
<td>#</td>
<td>0 1 0 0 1 0 1</td>
<td>C</td>
<td>1 0 0 0 1 0 1</td>
<td>c</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>$</td>
<td>0 1 0 0 1 1 1</td>
<td>D</td>
<td>1 0 0 0 1 1 1</td>
<td>d</td>
<td>1 1 0 0 1 1 1</td>
</tr>
<tr>
<td>%</td>
<td>0 1 0 1 0 0 1</td>
<td>E</td>
<td>1 0 0 0 1 0 1</td>
<td>e</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>&amp;</td>
<td>0 1 0 1 0 1 1</td>
<td>F</td>
<td>1 0 0 0 1 1 1</td>
<td>f</td>
<td>1 1 0 0 1 1 1</td>
</tr>
<tr>
<td>(</td>
<td>0 1 0 1 0 0 1</td>
<td>G</td>
<td>1 0 0 0 0 0 1</td>
<td>g</td>
<td>1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>)</td>
<td>0 1 0 1 0 0 0</td>
<td>H</td>
<td>1 0 0 0 1 0 0</td>
<td>h</td>
<td>1 1 0 1 0 0 0</td>
</tr>
<tr>
<td>+</td>
<td>0 1 0 1 0 0 1</td>
<td>I</td>
<td>1 0 0 0 0 1 1</td>
<td>i</td>
<td>1 1 0 1 0 0 1</td>
</tr>
<tr>
<td>,</td>
<td>0 1 0 1 1 0 0</td>
<td>J</td>
<td>1 0 0 1 0 0 0</td>
<td>j</td>
<td>1 1 0 1 0 1 0</td>
</tr>
<tr>
<td>-</td>
<td>0 1 0 1 1 0 1</td>
<td>K</td>
<td>1 0 0 1 0 1 1</td>
<td>k</td>
<td>1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>.</td>
<td>0 1 0 1 0 1 1</td>
<td>L</td>
<td>1 0 0 1 1 0 0</td>
<td>l</td>
<td>1 1 0 1 1 0 0</td>
</tr>
<tr>
<td>/</td>
<td>0 1 0 1 1 1 0</td>
<td>M</td>
<td>1 0 0 1 1 1 0</td>
<td>m</td>
<td>1 1 0 1 1 1 0</td>
</tr>
<tr>
<td>0</td>
<td>0 1 1 0 0 0 0</td>
<td>N</td>
<td>1 0 0 1 1 1 1</td>
<td>n</td>
<td>1 1 0 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 0 0 0 1</td>
<td>O</td>
<td>1 0 1 0 0 0 0</td>
<td>o</td>
<td>1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 1 1 0 0 1 0</td>
<td>P</td>
<td>1 0 1 0 0 0 1</td>
<td>p</td>
<td>1 1 1 0 0 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1 0 1 0 1</td>
<td>Q</td>
<td>1 0 1 0 1 0 1</td>
<td>q</td>
<td>1 1 1 0 1 0 1</td>
</tr>
<tr>
<td>4</td>
<td>0 1 1 0 1 1 0</td>
<td>R</td>
<td>1 0 1 0 1 1 0</td>
<td>r</td>
<td>1 1 1 0 1 1 0</td>
</tr>
<tr>
<td>5</td>
<td>0 1 1 1 0 1 0</td>
<td>S</td>
<td>1 0 1 1 0 0 1</td>
<td>s</td>
<td>1 1 1 1 0 0 1</td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 1 1 0 1</td>
<td>T</td>
<td>1 0 1 1 0 1 1</td>
<td>t</td>
<td>1 1 1 1 0 1 0</td>
</tr>
<tr>
<td>7</td>
<td>0 1 1 1 1 1 0</td>
<td>U</td>
<td>1 0 1 1 1 0 0</td>
<td>u</td>
<td>1 1 1 1 0 1 1</td>
</tr>
<tr>
<td>8</td>
<td>0 1 1 1 1 0 0</td>
<td>V</td>
<td>1 0 1 1 1 0 1</td>
<td>v</td>
<td>1 1 1 1 1 0 0</td>
</tr>
<tr>
<td>9</td>
<td>0 1 1 1 0 1 0</td>
<td>W</td>
<td>1 0 1 1 1 1 0</td>
<td>w</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>:</td>
<td>0 1 1 1 1 0 1</td>
<td>X</td>
<td>1 0 1 1 1 1 0</td>
<td>x</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>;</td>
<td>0 1 1 1 1 0 0</td>
<td>Y</td>
<td>1 0 1 1 1 1 0</td>
<td>y</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>&lt;</td>
<td>0 1 1 1 1 0 1</td>
<td>Z</td>
<td>1 0 1 1 1 1 0</td>
<td>z</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>=</td>
<td>0 1 1 1 1 1 0</td>
<td>[</td>
<td>1 0 1 1 1 1 0</td>
<td>[</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>&gt;</td>
<td>0 1 1 1 1 1 1</td>
<td>\</td>
<td>1 0 1 1 1 1 0</td>
<td>\</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>?</td>
<td>0 1 1 1 1 1 0</td>
<td>^</td>
<td>1 0 1 1 1 1 0</td>
<td>^</td>
<td>1 1 1 1 1 1 0</td>
</tr>
</tbody>
</table>

**Table 1-3** ASCII Code

© Cengage Learning 2014
Weighted Codes

- Weighted codes: each bit position has a given weight
  - Binary Coded Decimal (BCD) (8-4-2-1 code)
    - Example: $726_{10} = 0111\ 0010\ 0110_{BCD}$
  - 6-3-1-1 code
    - Example: $1001$ (6-3-1-1 code) = $1 \times 6 + 0 \times 3 + 0 \times 1 + 1 \times 1$
    - Example: $726_{10} = 1001\ 0011\ 1000_{6311}$

- BCD numbers are used to represent fractional numbers exactly (vs. floating point numbers – which can’t - see Chapter 5)
Weighted Codes

<table>
<thead>
<tr>
<th>Decimal #</th>
<th>8-4-2-1 (BCD)</th>
<th>6-3-1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0111</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>1001</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1011</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1100</td>
</tr>
</tbody>
</table>

- **BCD Example:**
  \[726_{10} = 0111 \ 0010 \ 0110_{\text{BCD}}\]

- **6-3-1-1 code Example:**
  \[726_{10} = 1001 \ 0011 \ 1000_{6311}\]
Excess-3 Code

- Add 3 to number, then represent in binary
  - Example: $5_{10} = 5 + 3 = 8 = 1000_2$
- Also called a biased number
- Excess-3 codes (also called XS-3) were used in the 1970’s to ease arithmetic

<table>
<thead>
<tr>
<th>Decimal #</th>
<th>Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0011</td>
</tr>
<tr>
<td>1</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>0101</td>
</tr>
<tr>
<td>3</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>0111</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>1001</td>
</tr>
<tr>
<td>7</td>
<td>1010</td>
</tr>
<tr>
<td>8</td>
<td>1011</td>
</tr>
<tr>
<td>9</td>
<td>1100</td>
</tr>
</tbody>
</table>

- Excess-3 Example:
  $7_{10} = \text{1010 0101 1001}_\text{xs3}$
2-out-of-5 Code

• 2 out of the 5 bits are 1

• Used for error detection:
  • If more or less than 2 of 5 bits are 1, error

<table>
<thead>
<tr>
<th>Decimal #</th>
<th>2-out-of-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00011</td>
</tr>
<tr>
<td>1</td>
<td>00101</td>
</tr>
<tr>
<td>2</td>
<td>00110</td>
</tr>
<tr>
<td>3</td>
<td>01001</td>
</tr>
<tr>
<td>4</td>
<td>01010</td>
</tr>
<tr>
<td>5</td>
<td>01100</td>
</tr>
<tr>
<td>6</td>
<td>10001</td>
</tr>
<tr>
<td>7</td>
<td>10010</td>
</tr>
<tr>
<td>8</td>
<td>10100</td>
</tr>
<tr>
<td>9</td>
<td>11000</td>
</tr>
</tbody>
</table>
Gray Codes

- Next number differs in only one bit position
  - **Example**: 000, 001, 011, 010, 110, 111, 101, 100

- **Example use**: Analog-to-Digital (A/D) converters. Changing 2 bits at a time (i.e., 011 → 100) could cause large inaccuracies.

<table>
<thead>
<tr>
<th>Decimal #</th>
<th>Gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>1110</td>
</tr>
<tr>
<td>6</td>
<td>1010</td>
</tr>
<tr>
<td>7</td>
<td>1011</td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
</tr>
</tbody>
</table>