ANALOG VS. DIGITAL

• The analog world is based on continuous events. Observations can take on (real) any value.

• The digital world is based on discrete events. Observations can only take on a finite number of discrete values.

Q. Which is better?
A. Depends on what you are trying to do.

• Some tasks are better handled with analog data, others with digital data.
  – Analog means continuous/real valued signals with an infinite number of possible values
  – Digital signals are discrete [i.e. 1 of n values]
3.5 Analog vs. Digital

- How much money is in my checking account?
  - Analog: Oh, some, but not too much.
  - Digital: $243.67

3.6 Analog vs. Digital

- How much do you love me?
  - Analog: I love you with all my heart!!!!
  - Digital: $3.2 \times 10^3$ MegaHearts

3.7 The Real (Analog) World

- The real world is inherently analog.
- To interface with it, our digital systems need to:
  - Convert analog signals to digital values (numbers) at the input.
  - Convert digital values to analog signals at the output.
- Analog signals can come in many forms
  - Voltage, current, light, color, magnetic fields, pressure, temperature, acceleration, orientation

3.8 Digital is About Numbers

- In a digital world, numbers are used to represent all the possible discrete events
  - Numerical values
  - Computer instructions (ADD, SUB, BLE, ...)
  - Characters ('a', 'b', 'c', ...)
  - Conditions (on, off, ready, paper jam, ...)
- Numbers allow for easy manipulation
  - Add, multiply, compare, store, ...
- Results are repeatable
  - Each time we add the same two number we get the same result
Interpreting Binary Strings

• Given a string of 1’s and 0’s, you need to know the **representation system** being used, before you can understand the value of those 1’s and 0’s.

01000001 = ?

---

Binary Representation Systems

- **Integer Systems**
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2’s complement
    - **Excess-N**
    - 1’s complement*
- **Floating Point**
  - For very large and small (fractional) numbers

- **Codes**
  - Text
    - ASCII / Unicode
  - Decimal Codes
    - BCD (Binary Coded Decimal) / (8421 Code)

---

Number Systems

- Number systems consist of
  1. ________________
  2. ___ coefficients [_______]
- **Human System**: Decimal (Base 10): 0,1,2,3,4,5,6,7,8,9
- **Computer System**: Binary (Base 2): 0,1
- **Human systems for working with computer systems** (shorthand for human to read/write binary)
  - ________________
  - ________________

* = Not fully covered in this class
Anatomy of a Decimal Number

• A number consists of a string of explicit coefficients (digits).
• Each coefficient has an implicit place value which is a power of the base.
• The value of a decimal number (a string of decimal coefficients) is the sum of each coefficient times its place value.

\[
(934)_{10} = 9 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 = 934
\]

\[
(3.52)_{10} = 3 \times 10^0 + 5 \times 10^{-1} + 2 \times 10^{-2} = 3.52
\]

Anatomy of a Binary Number

• Same as decimal but now the coefficients are 1 and 0 and the place values are the powers of 2.

\[
(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11
\]

General Conversion From Base r to Decimal

• A number in base r has place values/weights that are the powers of the base.
• Denote the coefficients as: \( a_i \)

\[
(a_3a_2a_1a_0a_{-1}a_{-2})_r = a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2}
\]

Examples

\[
(746)_8 =
\]

\[
(1A5)_{16} =
\]

\[
(AD2)_{16} =
\]
Binary Examples

(1001.1)_2 =

(10110001)_2 =

Powers of 2

2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
2^{10} = 1024

Unique Combinations

• Given n digits of base r, how many unique numbers can be formed? __
  – What is the range? [________]

2-digit, decimal numbers (r=10, n=2)
3-digit, decimal numbers (r=10, n=3)
4-bit, binary numbers (r=2, n=4)
6-bit, binary numbers (r=2, n=6)

Approximating Large Powers of 2

• Often need to find decimal approximation of a large powers of 2 like 2^{16}, 2^{32}, etc.

2^{10} = 2^{6} \cdot 2^{4}
2^{20} = 2^{16} \cdot 2^{4}
2^{30} = 2^{24} \cdot 2^{6}
2^{40} = 2^{32} \cdot 2^{8}

• Use following approximations:
  – 2^{10} = ____________
  – 2^{20} = ____________
  – 2^{30} = ____________
  – 2^{40} = ____________

• For other powers of 2, decompose into product of 2^{10} or 2^{20} or 2^{30} and a power of 2 that is less than 2^{10}
  – 16-bit half word: 64K numbers
  – 32-bit word: 4G numbers
  – 64-bit dword: 16 million trillion numbers
Decimal to Unsigned Binary

- To convert a decimal number, \( x \), to binary:
  - Only coefficients of 1 or 0. So simply find place values that add up to the desired values, starting with larger place values and proceeding to smaller values and place a 1 in those place values and 0 in all others

\[
25_{10} = \begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

\[
73_{10} = \begin{array}{cccccc}
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

\[
87_{10} = \begin{array}{cccccc}
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

\[
145_{10} = \begin{array}{cccccc}
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

\[
0.625_{10} = \begin{array}{cccccc}
\frac{1}{2} & \frac{1}{4} & \frac{1}{8} \\
\end{array}
\]

Decimal to Another Base

- To convert a decimal number, \( x \), to base \( r \):
  - Use the place values of base \( r \) (powers of \( r \)). Starting with largest place values, fill in coefficients that sum up to desired decimal value without going over.

\[
75_{10} = \begin{array}{cccc}
256 & 16 & 1 \\
\end{array}_{\text{hex}}
\]

Hexadecimal and Octal

**SHORTHAND FOR BINARY**
Binary, Octal, and Hexadecimal

• Octal (base 8 = \(2^3\))
• 1 Octal digit ( _ )\(_8\) can represent: __________
• 3 bits of binary (_ _ _)\(_2\) can represent: 000-111 = __________
• Conclusion...
__ Octal digit = ___ bits

• Hex (base 16=\(2^4\))
• 1 Hex digit ( _ )\(_{16}\) can represent: 0-F (_____)
• 4 bits of binary (_ _ _ _)\(_2\) can represent: 0000-1111 = __________
• Conclusion...
__ Hex digit = ___ bits

Binary to Octal or Hex

• Make groups of 3 bits starting from radix point and working outward
• Add 0’s where necessary
• Convert each group of 3 to an octal digit

101001110.11

Octal or Hex to Binary

• Expand each octal digit to a group of 3 bits

317.2\(_8\)

• Expand each hex digit to a group of 4 bits

D93.8\(_{16}\)

Hexadecimal Representation

• Since values in modern computers are many bits, we use hexadecimal as a shorthand notation (4 bits = 1 hex digit)
  – 11010010 = D2 hex or 0xD2 if you write it in C/C++
  – 0111011011001011 = 76CB hex or 0x76CB if you write it in C/C++
Binary Representation Systems

- **Integer Systems**
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2’s complement
    - 1’s complement*
    - Excess-N*
- **Floating Point**
  - For very large and small (fractional) numbers

* = Not covered in this class

**Codes**

- **Text**
  - ASCII / Unicode
- **Decimal Codes**
  - BCD (Binary Coded Decimal) / (8421 Code)

---

**Binary Codes**

- Using binary we can represent any kind of information by coming up with a code
- Using $n$ bits we can represent $2^n$ distinct items

Colors of the rainbow:
- Red = 000
- Orange = 001
- Yellow = 010
- Green = 100
- Blue = 101
- Purple = 111
- $'A'$ = 00000
- $'B'$ = 00001
- $'C'$ = 00010
- $'Z'$ = 11001

**BCD**

- Rather than convert a decimal number to binary which may lose some precision (i.e. $0.1_{10}$ = infinite binary fraction), BCD represents each decimal digit as a separate group of bits (exact decimal precision)
  - Each digit is represented as a _________ number (using place values 8,4,2,1 for each dec. digit)
  - Often used in financial and other applications where decimal precision is needed

(439)$_{10}$

BCD Representation:

Unsigned Binary Rep.: 110110111$_2$

**Important:** Some processors have specific instructions to operate on #’s represented in BCD
### ASCII Code
- Used for representing text characters
- Originally 7-bits but usually stored as 8-bits = 1-byte in a computer
- Example:
  - "Hello\n";
  - Each character is converted to ASCII equivalent
    - ‘H’ = 0x48, ‘e’ = 0x65, ...
    - \n = newline character is represented by either one or two ASCII character

### UniCode
- ASCII can represent only the English alphabet, decimal digits, and punctuation
  - 7-bit code \( \Rightarrow 2^7 = \) ______ characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - 16-bit Code \( \Rightarrow \) ______ characters
  - Represents many languages alphabets and characters
  - Used by Java as standard character code

### Boolean Algebra Intro

![Unicode hex value](i.e. FB52 => 1111101101010010)
Boolean Algebra

- A set of theorems to help us manipulate logical expressions/equations
- Axioms = Basis / assumptions used
- Theorems = manipulations that we can use

Axioms

- Axioms are the basis for Boolean Algebra
- Notice that these axioms are simply restating our definition of digital/binary logic
  - $A_1/A_1' = ___
  - $A_2/A_2' = ___$
  - $A_3,A_4,A_5 = ___$
  - $A_3',A_4',A_5' = ___$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$X = 0$ if $X \neq 1$</td>
<td>$(A_1')$</td>
</tr>
<tr>
<td>$(A_2)$</td>
<td>If $X = 0$, then $\overline{X} = 1$</td>
<td>$(A_2')$</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$0 \cdot 0 = 0$</td>
<td>$(A_3')$</td>
</tr>
<tr>
<td>$A_4$</td>
<td>$1 \cdot 1 = 1$</td>
<td>$(A_4')$</td>
</tr>
<tr>
<td>$A_5$</td>
<td>$1 \cdot 0 = 0 \cdot 1 = 0$</td>
<td>$(A_5')$</td>
</tr>
</tbody>
</table>

Duality

- Every truth statement can yields another truth statement
  - I exercise if I have time and energy (original statement)
  - I don’t exercise if I don’t have time or don’t have energy (dual statement)
- To express the dual, swap...

Duality

- The “dual” of an expression is not equal to the original

$$1 + 0 \neq 0 \cdot 1$$

- Taking the “dual” of both sides of an equation yields a new equation

$$X + 1 = 1 \quad \Rightarrow \quad X \cdot 0 = 0$$
Single Variable Theorems

- Provide some simplifications for expressions containing:
  - a single variable
  - a single variable and a constant bit
- Each theorem has a dual (another true statement)
- Each theorem can be proved by writing a truth table for both sides (i.e. proving the theorem holds for all possible values of X)

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T1'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$X + 0 = X$</td>
<td>$X + 1 = 1$</td>
</tr>
<tr>
<td>2.</td>
<td>$X + 1 = 1$</td>
<td>$X + 0 = 0$</td>
</tr>
<tr>
<td>3.</td>
<td>$X + X = X$</td>
<td>$X + X = X$</td>
</tr>
<tr>
<td>4.</td>
<td>$(X')' = X$</td>
<td>$X + X' = 1$</td>
</tr>
<tr>
<td>5.</td>
<td>$X + X' = 1$</td>
<td>$X + X' = 0$</td>
</tr>
</tbody>
</table>

Single Variable Theorem (T1)

$X + 0 = X$ (T1) $X + 1 = 1$ (T1')

When a variable is OR'ed with 0, the output will be the same as the variable...
"0 OR anything equals that ___" When a variable is AND'ed with 1, the output will be the same as the variable...
"1 AND anything equals that ___"

Single Variable Theorem (T2)

$X + 1 = 1$ (T2) $X + 0 = 0$ (T2')

Whenever a variable is OR'ed with 1, the output will be 1...
"1 OR anything equals ___"
Whenever a variable is AND'ed with 0, the output will be 0...
"0 AND anything equals ___"

Single Variable Theorem (T3)

$X + X = X$ (T3) $X + X = X$ (T3')

Whenever a variable is OR'ed with itself, the result is just the value of the variable
Whenever a variable is AND'ed with itself, the result is just the value of the variable

This theorem can be used to reduce two identical terms into one OR to replicate one term into two.
### Single Variable Theorem (T4)

\[(X')' = X \quad (T4)\]
\[(\bar{X}) = X \quad (T4)\]

Anything inverted twice yields its original value.

### Single Variable Theorem (T5)

\[X + \bar{X} = 1 \quad (T5)\]
\[X \cdot \bar{X} = 0 \quad (T5')\]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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>

**OR**

Whenever a variable is OR’ed with its complement, one value has to be 1 and thus the result is 1.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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>

**AND**

Whenever a variable is AND’ed with its complement, one value has to be 0 and thus the result is 0.

This theorem can be used to simplify variables into a constant or to expand a constant into a variable.

### Application: Channel Selector

- Given 4 input, digital music/sound channels and 4 output channels.
- Given individual “select” inputs that select 1 input channel to be routed to 1 output channel.

### Application: Steering Logic

- 4-input music channels (ICHx)
  - Select one input channel (use ISELx inputs)
  - Route to one output channel (use OSELx inputs)

---

I Ch0
ICH1
ICH2
ICH3

O CH0
O CH1
O CH2
O CH3

Input Channel Select
ISEL0
ISEL1
ISEL2
ISEL3

Output Channel Select
OSEL0
OSEL1
OSEL2
OSEL3
Application: Steering Logic

- 1st Level of AND gates act as barriers only passing 1 channel
- OR gates combines 3 streams of 0's with the 1 channel that got passed (i.e. ICH1)
- 2nd Level of AND gates passes the channel to only the selected output

Your Turn

- Build a circuit that takes 3 inputs: S, IN0, IN1 and outputs a single bit Y.
- It’s functions should be:
  - If S = 0, Y = IN0 (IN0 passes to Y)
  - If S = 1, Y = IN1 (IN1 passes to Y)