1.1 Digital Systems and Switching Circuits

Analog vs. Digital Information Processing

Analog: physical quantities (signals) vary continuously over a range, continuously valued, most “real world” signals are analog

Digital: discrete-valued, signals take on one of a finite set of possible values

Ex 1: Representation (time) \( \uparrow \) vs. 5:05

analog    digital

Ex 2: Computation (addition)

\[ V_{out} = -(V_A + V_B) \quad \text{ideal} \]
\[ V_A = 4V \quad V_B = 2V \]
\[ V_{out} = -6V \pm V_{error} \approx -6V \]

\( V_{error} \) due to noise, nonideal circuits, etc.

\[ a = 4 \quad b = 2 \]

out = 6 exactly!

Trend is to replace analog with digital (music, video, telephony) because digital is arbitrarily accurate, stable (with respect to noise, temperature, manufacturing variations), and easier to design.

Digital System Design

Roth describes three parts: ① System design (180B, 170) ② Logic design (180A) ③ Circuit design (118, 116)

Logic design: interconnecting gates and flip-flops to perform a specific function
Switching Circuits

1. Combinational Logic Circuits
   Def: Outputs of a combinational block depend only on its present inputs.

   Ex: Light switch

2. Sequential Logic Circuits
   Def: Outputs of a sequential logic block depend on present inputs and past inputs, i.e., it has "memory"

   Ex: Nearly any useful digital device typically is made up of combinational circuits and memory.

   Microwave Oven

   Traffic Light

   Ex: Digital processors like microprocessors, digital signal processors (DSPs), graphics processors usually have

   1) Memory - stores programs and data
   2) Datapath - process information (Arithmetic/Logic Unit (ALU), adder, multiplier)
   3) Controller - direct memory controller, disk drive controller
   4) Supporting Circuitry
Positional Notation: Each digit is multiplied by an appropriate power of the base number depending on its position in the number.

Decimal (base 10) digit values: \{0, 1, 2, \ldots, 9\}

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
x 10^2 & x 10^1 & x 10^0 & x 10^{-1}
\end{array}
\]

\[123.4_{10} = 1 \times 100 + 2 \times 10 + 3 \times 1 + 4 \times 0.1\]

Binary (base 2) digit values: \{0, 1\}

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
x 2^2 & x 2^1 & x 2^0 & x 2^{-1}
\end{array}
\]

\[101.1_2 = 1 \times 4_{10} + 0 \times 2_{10} + 1 \times 1_{10} + 1 \times 0.5_{10} = 5.5_{10}\]

To convert to decimal, write the power series in terms of the decimal representation of the base and sum the series.

Octal (base 8) digit values: \{0, 1, 2, \ldots, 7\}

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
x 8^2 & x 8^1 & x 8^0 & x 8^{-1}
\end{array}
\]

\[123.4_{8} = 1 \times 64_{10} + 2 \times 8_{10} + 3 \times 1_{10} + 4 \times 0.125_{10} = 83.5_{10}\]
Hexadecimal (base 16) digit values: \( \{0, 1, \ldots, 9, A, B, C, D, E, F\} \)

\[
\begin{array}{cccc}
\times 16^2 & \times 16^1 & \times 16^0 & \times 16^{-1} \\
A & 0 & 1 & C \\
\times 256 & \times 16 & \times 1 & \times \frac{1}{16} \\
\end{array}
\]

\[
A \times 256_{10} + 0 \times 16_{10} + 1 \times 1_{10} + C \times \frac{1}{16_{10}}
= 10_{10} \times 256_{10} + 0 \times 16_{10} + 1 \times 1_{10} + 12_{10} \times \frac{1}{16_{10}}
= 2560_{10} \times \frac{1}{16}_{10} = 2561.75_{10}
\]

To convert from decimal to another base, use successive division by the new base for integers and successive multiplication by the new base for fractions.

For successive division, divide original number and each quotient by the new base until the quotient is 0. The remainders form the digits for the representation in the new base.

Ex: Convert \( 11_{10} \) to binary.

\[
\begin{array}{ccc}
\text{Quotient} & \text{Remainder} & \\
2/11 & 5 & 1 \\
2/5 & 2 & 1 \\
2/2 & 1 & 0 \\
2/1 & 0 & 1 \\
\end{array}
\]

\( 11_{10} = 1011_2 \)

Alternative method: Start with largest power of 2 less than original number, subtract, and repeat with the difference until the difference is 0.

\[
\begin{array}{cccc}
16 & 8 & 4 & 2 \\
10 & 0 & 1 & 1 \\
\end{array}
\]

Check: \( 1 \times 8_{10} + 0 \times 4_{10} + 1 \times 2_{10} + 1 \times 1_{10} = 8 + 2 + 1_{10} = 11_{10} \)
For successive multiplication, multiply original number and each fractional part by the new base until the fractional part is 0. The integer parts form the digits in the new base representation.

**Ex:** Convert 0.375<sub>10</sub> to base 2.

\[
\begin{array}{c}
0.375 \\
\times 2 \\
0.750 \\
\times 2 \\
1.500 \\
\times 2 \\
1.000
\end{array}
\]

0 1 1 \implies 0.375<sub>10</sub> = 0.011<sub>2</sub>

**Ex:** Convert 0.48046875<sub>10</sub> to hexadecimal (base 16).

\[
\begin{array}{c}
0.48046875 \\
\times 16 \\
7.6875 \\
\times 16 \\
11.000 \\
\end{array}
\]

11<sub>16</sub> = B<sub>16</sub> \implies 0.48046875<sub>10</sub> = 0.7B<sub>16</sub>

Binary ↔ Hexadecimal: 4 binary digits correspond to one hex digit.

**Ex:** 1010 0011 • 0111 0010<sub>2</sub>

A 3 7 2 = A372<sub>16</sub>

Binary ↔ Octal: 3 binary digits (bits) correspond to one octal digit.

**Ex:** 100 110 • 011

4 6.3 \implies 46.3<sub>8</sub>

It is often easier to convert to binary first before converting to hexadecimal or octal!
1.3 Binary Arithmetic  (Note: unsigned binary numbers)

**Addition**  Same procedure as decimal addition.

Ex: \( 1 \) carry

\[
\begin{align*}
1100_2 & = 12_{10} \\
+0110_2 & = 6_{10} \\
1010_2 & = 18_{10}
\end{align*}
\]

Ex: \( 1102 \)

\[
\begin{align*}
0110_2 & = 6_{10} \\
0011_2 & = 3_{10} \\
+1110_2 & = 14_{10} \\
10111_2 & = 23_{10}
\end{align*}
\]

**Subtraction**  (1) Similar to decimal subtraction; borrow from more significant columns.

(2) Add the negative: \( a - b = a + (-b) \)

We’ll concentrate on method (2), used much more commonly in hardware.

**Multiplication**  Same procedure as base 10 multiplication.

Ex: \( 1112 = 7_{10} \)

\[
\begin{align*}
\times 101_2 & = 5_{10} \\
111 \\
000 \\
\hline
100011_2 & = 35_{10}
\end{align*}
\]

"Partial products" are very simple in binary!

**Division**  Similar to decimal division, but must subtract using subtraction methods above.

\[
\begin{array}{c|cccc}
12 & 111_2 \\
\hline
101_2 & 101_2 \\
\hline
10_2 \\
\end{array}
\]

Quotient is \( 12 \) with remainder \( 10_2 \).
1.4 Negative Numbers

1. Sign and Magnitude Representation (similar to decimal)

Base 10: +5
-5

\[ + \] magnitude (absolute value) = 5

Base 2: +101 \Rightarrow 0101 \text{ (overshadowed)} \text{ magnitude = 101}_2 = 5_{10}
-101 \Rightarrow 101

↑ "Sign" bit, typically 0 = positive, 1 = negative

- Not very convenient representation for addition/subtraction hardware. 😞
- Great for multiplication 🎉
- Two representations for zero: +0 = 0000 ≠ -0 = 1000 😞

2. Two's Complement (2's Complement)

Most common signed binary number representation.

- For an n-bit number N, 2's complement \( N^* = -N = 2^n - N \)

Ex: -3: +3 = 0011
-3 = \[ 2^3 \] - 0011
= 10000 - 0011
= 1101

- Same as the unsigned representation, except Most Significant Bit (MSB) corresponds to negative number.

\[ \begin{array}{cccccccc}
  & & & 1 & 0 & 1 & 1 & 1 \\
  \text{x} & x^2 & x^4 & x^8 & x^{16} & x^{32} \\
  \hline
  1 & 0 & 1 & 1 & & & & \\
\end{array} \]

\[ 11011_2 = 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 16 + 4 + 2 + 1 = 23 \]

- Easiest method to negate a number: "Flip (Invert) all bits and add 1"

Ex: +3 = 0011

<table>
<thead>
<tr>
<th>1100</th>
<th>flip bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>add 1</td>
</tr>
</tbody>
</table>

-3 = 1101

<table>
<thead>
<tr>
<th>0010</th>
<th>flip bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>add 1</td>
</tr>
</tbody>
</table>

+3 = 0011
2's complement features:
1) MSB is sign bit, 0 = positive, 1 = negative
2) One representation for zero: all 0's, e.g., 0000
3) All 1's represent -1, e.g., 1111

Ex: \(111 = -4 + 2 + 1 = -1\)
\(1111 = -16 + 8 + 4 + 2 + 1 = -16 + 15 = -1\)

4) Extending (replicating) the sign bit (called sign extension) will never change a number and is often required/convenient:

Ex: positive \(+2 = 010 = 0010 = 0000\ 0010\)

negative \(-2 = 110 = 1110 = 1111\ 1110\)

5) Errors can result if operation overflows/underflows

3) One's complement (1's complement)
   Good to know, but not often used.

* Negate by inverting all bits
  Ex: \(+3 = 011\)
  \(-3 = 100\) flip bits

* Two representations for zero:
  \(+0 = 0000\ \neq \ -0 = 1111\)

1.5 Binary Codes
Format is a hybrid of base 10 (decimal) and base 2 (binary) that is simple for humans to read.

Numbers are made up of base 10 digits represented by base 2 digits.

Ex: \(129_{10} \Rightarrow 1\ 2\ 9\)
\(0001\ 0010\ 1001 = 0001\ 0010\ 1001\) _BCD_

BCD = Binary Coded Decimal

BTW, \(129_{10} = 10000001_{2}\)

Ex: \(11001_{2} = 25_{10} = 0010\ 0101\) _BCD_

Some old computers did math in BCD, today most use 2's complement binary and convert to BCD for input/output.