# 1. Introduction

- Students are required to go through this introductory material on their own.
- Essential concepts will be discussed in Tutorial 1.

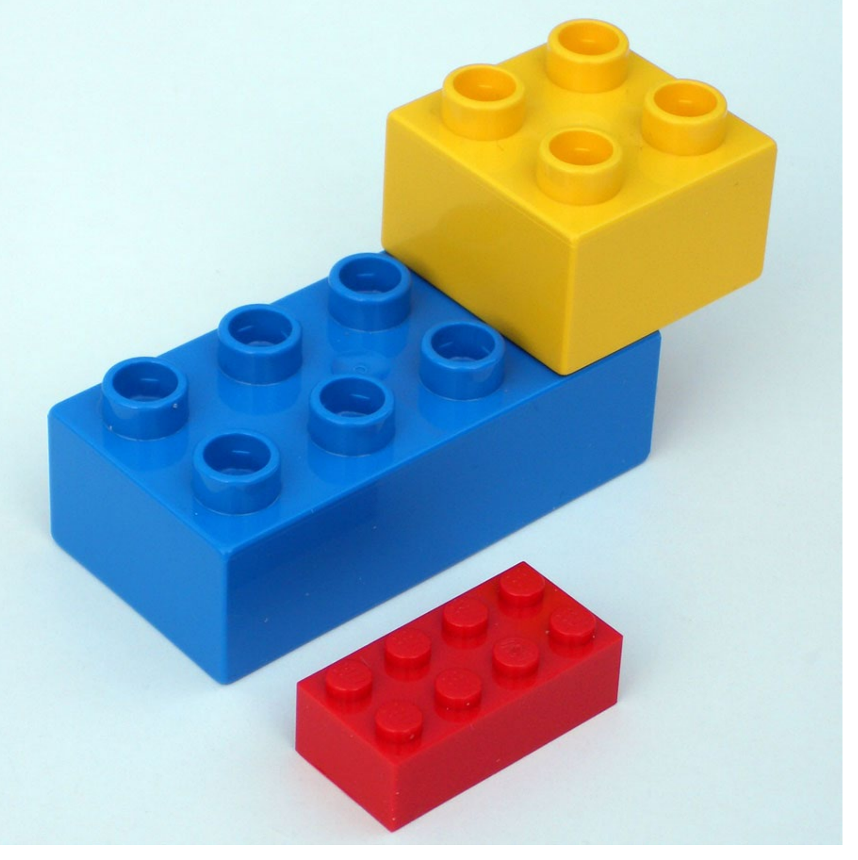
# Quick links to each section

# About Digital Design

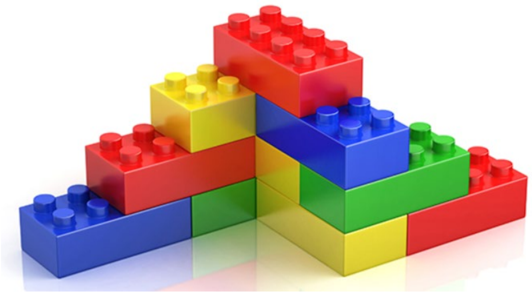
- Digital Design is also known as Logic Design, or Digital Electronics.
- Why should we learn it?

  - it provides the fundamentals of designing modern digital gadgets and computer systems, including medical equipment, smart phones, tablet PCs, laptops, security systems, etc.

# Digital design is like playing with building blocks… YES, it can be fun!



# You can create almost anything… limited only by your imagination



# A Successful Digital Designer should be competent in

- Debugging (troubleshoot systematically – not by trial-and-error)
- Business requirements & practices (documentation, specifications)
- Risk taking (in making design decisions)
- Communication (both directions: speak and listen)

You will appreciate all these when you embark on SC2079 Multidisciplinary Design Project (MDP)

# Analog versus Digital

- Analog quantities happen in the nature around us, examples are time, temperature, light intensity, sound volume, etc.
- An analog quantity changes in a <u>continuous</u> manner over time (e.g. it gets bright gradually in the early morning).

- Digital quantities are countable, examples are
  - the number of people in a lecture theatre
  - the amount of school fee you pay
  - the number of AUs you need to obtain in order to graduate

# Analog versus Digital

Analog:

The speaker volume can be increased or decreased by very small amount.

Digital:

The speaker volume can only be increased or decreased in steps.

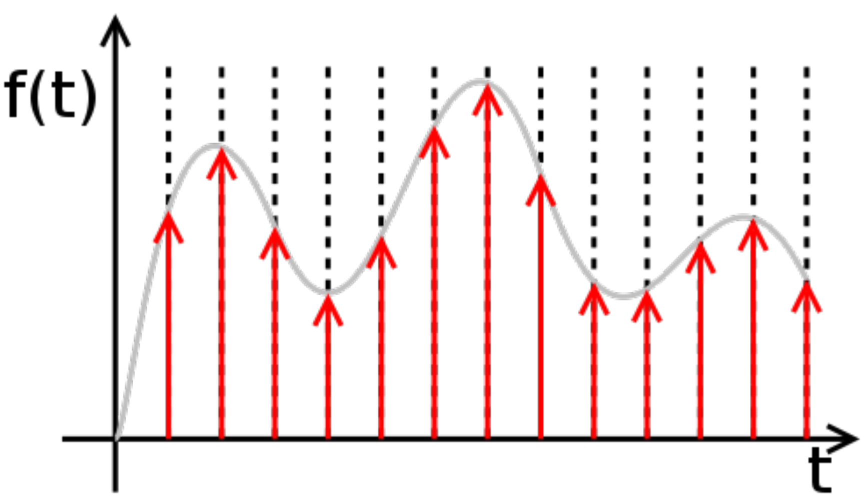
- A digital quantity changes in discrete steps.
- Analog quantities can be represented in digital format by using sampling and quantisation.

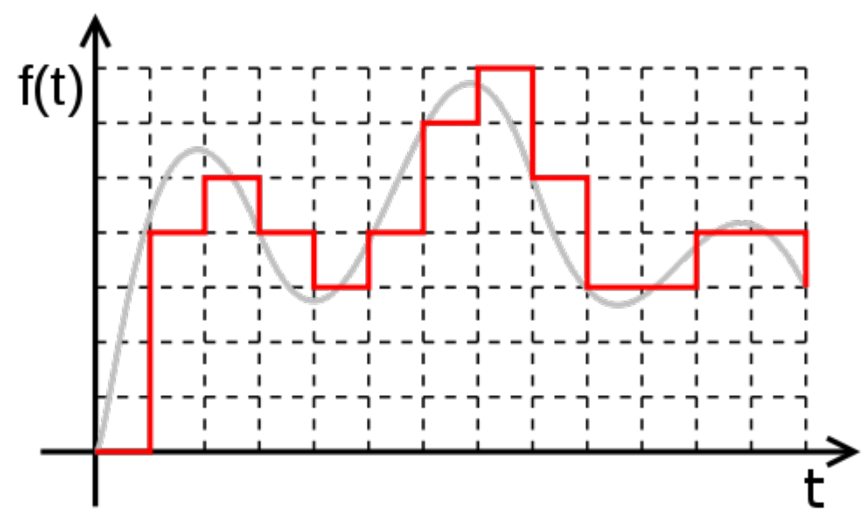
- Examples of analog quantities that are commonly digitised:
  - digital clock/watch (time is analog)
  - digital thermometer (temperature is analog)
  - digital camera (light intensity is analog)
  - digital audio/video content (light and sound intensities are analog)

f(t) is an analog signal continuously varying with time (gray curve)

Sampling f(t) at periodic intervals will generate the discrete time signal (red arrows)

Quantisation of the discrete time signal will produce the digital signal (red lines)



# What we should be aware of Quantisation

- A range of analog values is lumped together and assigned a representative digital value. For example, 0V to 0.8V is assigned logic 0; 2V to 5V is assigned logic 1
- A many-to-one mapping (using the above example, 2V is mapped to 1, similarly 3V, 3.8V, 5V are also mapped to 1)
- A finite amount of precision is lost in the process (logic 1 can mean 2V; logic 1 can also mean 5V)
- See illustration on next page

| Temperature range | Quantised value |
| --- | --- |
| 20 <= temp < 21 | $20^oC$ |
| 19 <= temp < 20 | $19^oC$ |
| 18 <= temp < 19 | $18^oC$ |
| 17 <= temp < 18 | $17^oC$ |
| 16 <= temp < 17 | $16^oC$ |
| 15 <= temp < 16 | $15^oC$ |
| 14 <= temp < 15 | $14^oC$ |
| 13 <= temp < 14 | $13^oC$ |

Any temperature variation within each range cannot be distinguished in the digital representation

# Advantages of Digital Techniques Over Analog Techniques

- Easier to design
- Information storage is easy
- Greater accuracy & precision
- Programmability
- Less susceptible to circuit noise
- VLSI (Very Large Scale Integration) technology

  - high speed
  - low cost
  - small size

# Limitations of Digital Technique

- The real world is mainly analog in nature, hence there is a need to
  - convert analog inputs to digital form
  - process the digital information
  - convert the digital result back to analog form
- The advantages of digital techniques usually outweigh the additional time, complexity and expenses involved in ADC (analog-to-digital conversion) and DAC (digital-to-analog conversion)

# Digital Number Systems

The decimal system is most commonly used in daily life because we have 10 fingers. But digital circuits prefer the binary system.

| Number system | Symbols |
|---|---|
| Decimal | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| Binary | 0, 1 |
| Octal | 0, 1, 2, 3, 4, 5, 6, 7 |
| hexadecimal | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |

You will learn more in the self-study material 2a_number_systems.pdf

# Electronic aspects

- In digital circuits, information is stored in binary digits
- A binary digit (commonly known as bit) has the value of 0 or 1
- In digital electronic circuits, the binary value is represented by electrical voltage or current
- Thus all digital gadgets require electrical power supply to work

- For example:

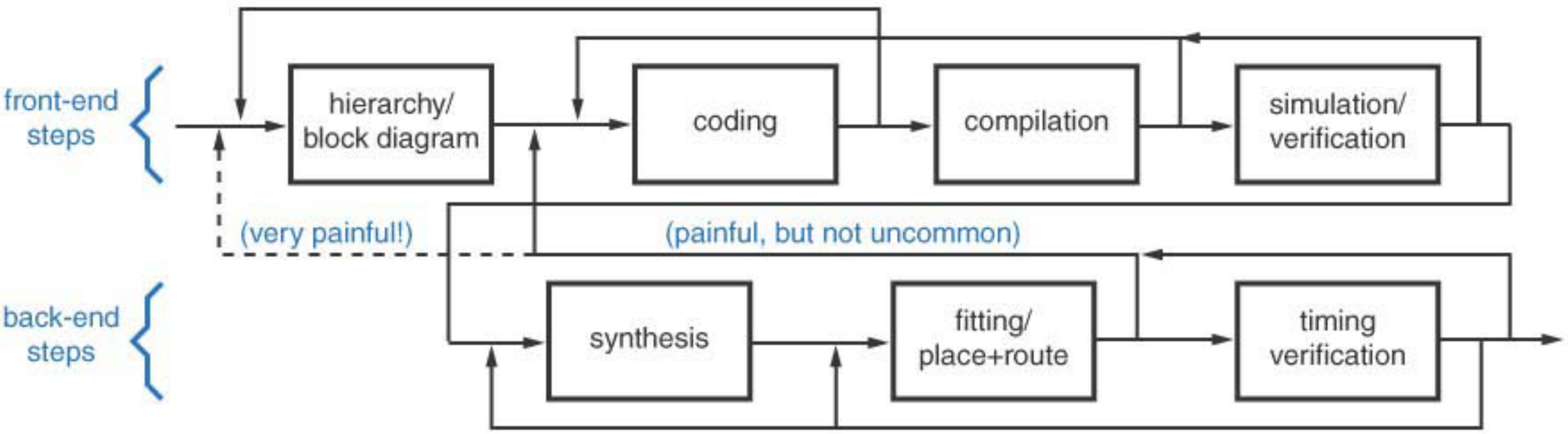
  0 volt – 0.8 volt may represent 0

  2 volts – 5 volts may represent 1
- The exact voltage level within each range is usually not important (e.g. 0.2 or 0.3 volt both represent 0)
- Usually more bits are required to represent useful information
- 4 bits make a nibble, e.g. 1001
- 8 bits make a byte, e.g. 1100 0101

# Software aspects

- Modern digital design involves Computer-Aided Design (CAD) software tools
- Schematic entry: use a software tool to draw circuit connections diagrams
- HDL: use Hardware Description Language to describe the logic circuit (e.g. Verilog)
- Synthesizer: creates a circuit realisation based on the above inputs

- Simulator: predicts the electrical and functional behaviour of a circuit without actually building it
- Test bench: a software environment to test the simulated circuit's functional and timing behaviour
- You will use some of these tools in the lab experiments
- Fig. 1.19 on the next page shows the typical design flow





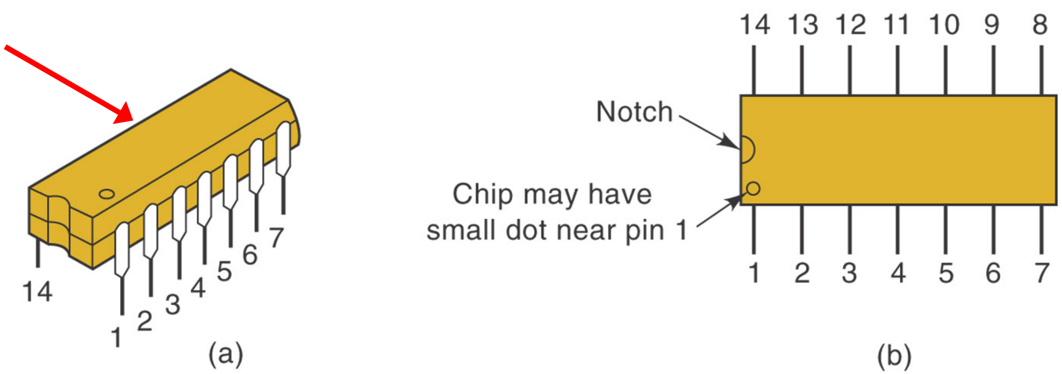

# Fig. 1.19 HDL-based design flow

# Integrated Logic Circuits

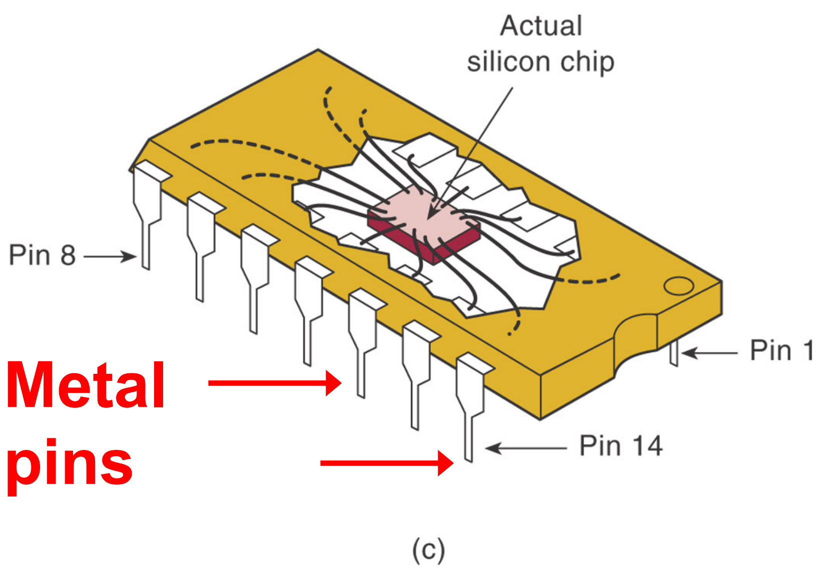
- Logic circuits are usually fabricated as Integrated Circuits (ICs) using various semiconductor technologies – see Fig. 4.29 on next page
- You will use some of these ICs in the lab experiments
- The circuit's logic can range from very simple to very complex

**Plastic or ceramic protective casing**

**Pin numbers**




**Top view**




**Metal pins**

The logic is built into the silicon chip. The metal pins are for connections.

**Figure 4.29: (Tocci 10th Ed) Dual-in-line Package**

# Programmable Logic Devices

- In some integrated circuits (ICs), the circuit's logic function can be easily changed, i.e. programmable
- This allows bugs to be fixed or circuit behaviour to be modified without physically replacing or rewiring the device
- An example is FPGA, Field-Programmable Gate Array
- You will be using it in the lab experiments

# Digital Data transmission

- Data (in bits) can be transmitted from one device to another in 2 ways: serial or parallel
- Parallel: think 4 checkout counters at the supermarket. 4 customers can be served at the same time
- Serial: think 1 checkout counter at the supermarket. Only 1 customer can be served at any time
- Trade off is Simplicity/Cost versus Speed

E.g. Fig. 1.10, Transmission of 8 bits of data

Parallel: all 8 bits transmitted at the same time

Advantage:

High speed

Serial: 8 bits transmitted one bit at a time

Advantage:

Low cost

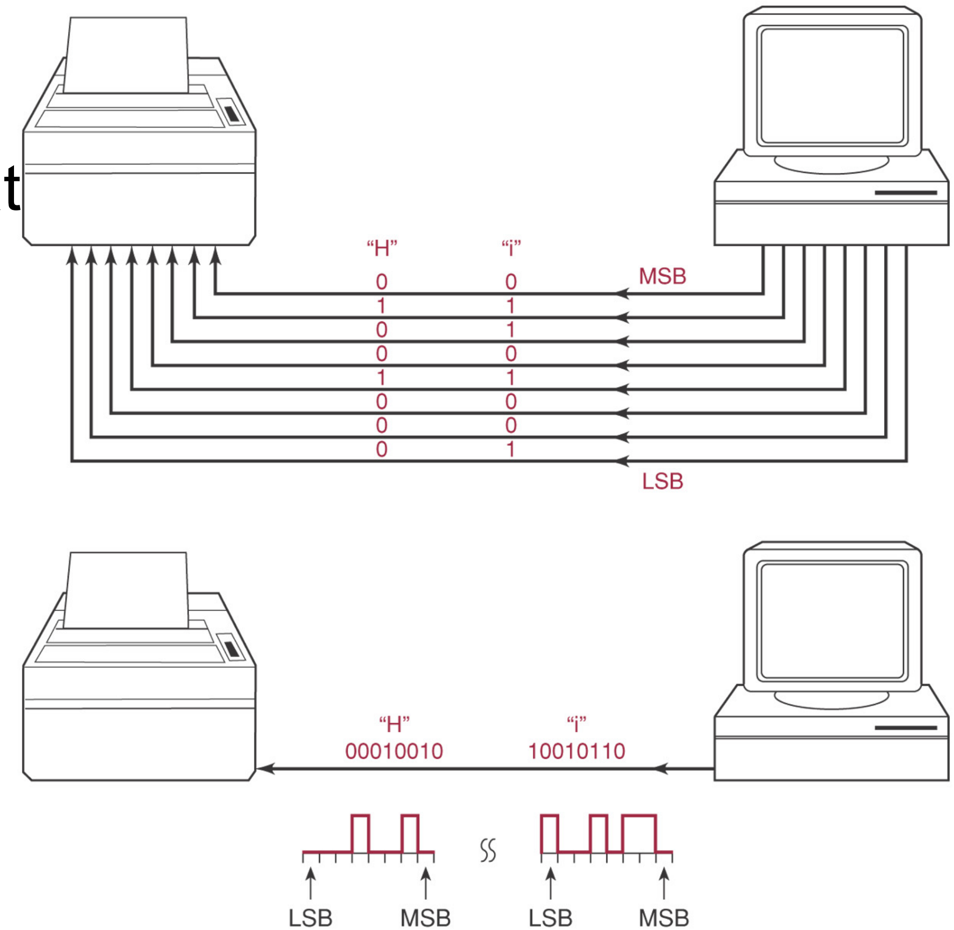



**Figure 1.10 (Tocci 10th Ed) Parallel and Serial Transfer**

**Example:**

- 1 serial line to transmit 8 bits, say at 1 bit per millisecond. Total time taken to transmit is 8 milliseconds. But 1 serial line costs only, say $1 (low cost option).

- 8 parallel lines can transmit all 8 bits simultaneously in one millisecond. But 8 lines may cost $8 (high speed option).
- Data Transfer Methods - YouTube

# 2a. Number Systems

- Students are required to handle these number systems confidently.
- Essential concepts will be discussed in Tutorial 1.

# Quick links to each section

# Common Number Systems

- **Decimal - base 10**

  10 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

  Examples of decimal numbers:

  $48_{10}$ , $915_{10}$ , $607_{10}$ , $23_{10}$

- **Binary - base 2**

  2 symbols: 0, 1

  Examples of binary numbers:

  $10110_{2}$ , $111000010_{2}$ , $101011111_{2}$

Digit other than 0 or 1 cannot appear in a binary number

- **The subscript 10 or 2 shows the *base* or *radix***

- **Octal - base 8**

  8 symbols: 0, 1, 2, 3, 4, 5, 6, 7

  e.g. $417_8$ , $26_8$ , $530_8$

Digit 8 or 9 cannot appear in an octal number

- **Hexadecimal - base 16**

  16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

  e.g. $F019_{16}$ , $43127C_{16}$ , $85_{16}$ , $BEAD_{16}$

Refer to Table 2-1 on next page:

$1011_2 = 11_{10} = 13_8 = B_{16}$

# Table 2.1 Binary, decimal, octal and hex

| Binary | Decimal | Octal | 3-Bit String | Hexadecimal | 4-Bit String |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 000 | 0 | 0000 |
| 1 | 1 | 1 | 001 | 1 | 0001 |
| 10 | 2 | 2 | 010 | 2 | 0010 |
| 11 | 3 | 3 | 011 | 3 | 0011 |
| 100 | 4 | 4 | 100 | 4 | 0100 |
| 101 | 5 | 5 | 101 | 5 | 0101 |
| 110 | 6 | 6 | 110 | 6 | 0110 |
| 111 | 7 | 7 | 111 | 7 | 0111 |
| 1000 | 8 | 10 | — | 8 | 1000 |
| | | | | | 1001 |
| | | | | | 1010 |
| 1011 | 11 | 13 | — | B | 1011 |
| 1100 | 12 | 14 | — | C | 1100 |
| 1101 | 13 | 15 | — | D | 1101 |
| 1110 | 14 | 16 | — | E | 1110 |
| 1111 | 15 | 17 | — | F | 1111 |

$$\boxed{1011_2 = 11_{10} = 13_8 = B_{16}}$$

- The number of symbols is equal to the *base (or radix)*
- Octal - base *8*, it has *8* symbols
- Hexadecimal - base *16*, it has *16* symbols
- Binary – base *2*, it has only *2* symbols
- The lower the base, the larger number of digits is required to represent a given value
- Thus $11_{10}$ requires 2 decimal digits, 2 octal digits, 4 binary digits, but only 1 hexadecimal digit to represent its value:

$$11_{10} = 13_8 = 1011_2 = B_{16}$$

- The binary system is most commonly used in digital systems
- Typing/writing a long string of 0's and 1's is error-prone for human
- Hexadecimal is a shorthand for human to type/write binary numbers

Examples:

$1011_2 = B_{16} = 0xB$

0x prefix signifies a Hex number

$$\underbrace{1100}_{C}\ \underbrace{0001}_{1}\ \underbrace{1001}_{9}\ \underbrace{1010}_{A}{}_2 = 0xC19A$$

# Position-value system

- Each digit carries a weight.
- The LSD carries the least weight. The MSD carries the most weight.

$ 9 999 99 9

MSD: most significant digit

LSD: least significant digit

- The weight (expressed in decimal) carried by a base-N digit of position *p* (*p*=0, 1, 2, …) is given by $N^P$ (i.e. N raised to the power of *p*; or N multiplied by itself for *p*-number of times)

- The corresponding weights of a base-N number are thus

$$N^3 \; N^2 \; N^1 \; N^0 . \; N^{-1} \; N^{-2} \; N^{-3}$$

- Note that $N^0 = 1$ for $N \neq 0$

- The weights of a **Decimal number**

  $10^{3}$ $10^{2}$ $10^{1}$ $1 \bullet 10^{-1}$ $10^{-2}$ $10^{-3}$

  Decimal point

- The weights of a **Binary number**

  $2^{3}$ $2^{2}$ $2^{1}$ $1 \bullet 2^{-1}$ $2^{-2}$ $2^{-3}$

  Binary point

- The weights of an **Octal number**

  $8^{3}$ $8^{2}$ $8^{1}$ $1 \bullet 8^{-1}$ $8^{-2}$ $8^{-3}$

  Octal point

- The weights of a **Hex number**

  $16^{3}$ $16^{2}$ $16^{1}$ $1 \bullet 16^{-1}$ $16^{-2}$ $16^{-3}$

  Hexadecimal point

# 4-bit binary system

$2^2 + 2^0 = 5_{10}$

$2^3 + 2^2 + 2^1 = 14_{10}$

| Weights | | | | Decimal |
|---|---|---|---|---|
| $2^3=8$ | $2^2=4$ | $2^1=2$ | $2^0=1$ | equivalent |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | 10 |
| 1 | 0 | 1 | 1 | 11 |
| 1 | 1 | 0 | 0 | 12 |
| 1 | 1 | 0 | 1 | 13 |
| 1 | 1 | 1 | 0 | 14 |
| 1 | 1 | 1 | 1 | 15 |

# Conversion from base-N to base-10:

**1.** Multiply each digit of the base-N number by its positional weight.

2. Sum together the products obtained in step 1.

**Examples**

$100.001_2 = (1 \times 2^2) + (1 \times 2^{-3}) = 4.125_{10}$

$5.7_8 = (5 \times 8^0) + (7 \times 8^{-1}) = 5.875_{10}$

$AF.2_{16} = (10 \times 16^1) + (15 \times 16^0) + (2 \times 16^{-1})$
$= 175.125_{10}$

# Conversion from base-10 to base-N:

**1.** Divide the base-10 number repeatedly by N until a quotient of 0 is obtained.

2. Write down the remainder after each division.

3. The first remainder is the LSD and the last remainder is the MSD of the base-N number. The rest of the remainders fall sequentially between the LSD and the MSD.

Examples: conversion from decimal to base-N

Convert

- 13 to binary

- 25 to octal

- 59 to hex

- 5.3 to binary (repeat division for integer, repeat multiplication for fraction)

Octal and Hex numbers are usually used as “short form” by human for binary numbers.

## $13_{10}$ to binary

$13 \div 2 = 6$ R 1

$6 \div 2 = 3$ R 0

$3 \div 2 = 1$ R 1

$1 \div 2 = 0$ R 1

$\boxed{13_{10} = 1101_2}$

## $25_{10}$ to octal

$25 \div 8 = 3$ R 1

$3 \div 8 = 0$ R 3

$\boxed{25_{10} = 31_8}$

## $59_{10}$ to hex

$59 \div 16 = 3 \text{ R } 11$

$3 \div 16 = 0 \text{ R } 3$

$$\boxed{59_{10} = 3B_{16}}$$

## $5.3_{10}$ to binary

$5 \div 2 = 2 \text{ R } 1$

$2 \div 2 = 1 \text{ R } 0$

$1 \div 2 = 0 \text{ R } 1$

$5_{10} = 101_2$

$0.3 \times 2 = 0.6$

$0.6 \times 2 = 1.2$

$0.2 \times 2 = 0.4$

$0.4 \times 2 = 0.8$

$0.8 \times 2 = 1.6$

$0.6 \times 2 = 1.2$

$$\boxed{5.3_{10} = 101.010011\ldots_2}$$

# Explanation of conversion

e.g. a base-10 number: $d_2\ d_1\ d_0 \bullet d_{-1}\ d_{-2}\ d_{-3}$

It has the value of

$(d_2 \times 10^2) + (d_1 \times 10^1) + (d_0 \times 10^0)$ - integer

$+ (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + (d_{-3} \times 10^{-3})$ - fraction

It can be represented by the binary number

$b_m \ldots b_1\ b_0 \bullet b_{-1}\ b_{-2} \ldots b_{-n}$

which has the value of

$(b_m \times 2^m) + \ldots + (b_1 \times 2^1) + (b_0 \times 2^0)$ - integer

$+ (b_{-1} \times 2^{-1}) + (b_{-2} \times 2^{-2}) + \ldots + (b_{-n} \times 2^{-n})$ - fraction

# Explanation of conversion (integer)

$(d_2 \times 10^2) + (d_1 \times 10^1) + (d_0 \times 10^0)$

has the same value as

$(b_m \times 2^m) + \ldots + (b_1 \times 2^1) + (b_0 \times 2^0)$ - integer

Divide by 2, we get

$$\underbrace{(b_m \times 2^{m-1}) + \ldots + (b_1 \times 2^0)}_{\text{Quotient: integer}} + \underbrace{(\mathbf{b_0} \times 2^{-1})}_{\text{fraction}}$$

We get $\mathbf{b_0}$ which is the remainder.

# Explanation of conversion (cont)

Divide the quotient by 2 again, we get

$$\underbrace{(b_m \times 2^{m-2}) + \ldots + (b_2 \times 2^0)}_{\text{Quotient: integer}} + \underbrace{(\mathbf{b_1} \times 2^{-1})}_{\text{fraction}}$$

We get $\mathbf{b_1}$ which is the remainder.

Thus by repeated division, the bits
$b_0$ , $b_1$ , $b_2$ , …, $b_m$ are obtained in sequence.

# Explanation of conversion (fraction)

$(d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + (d_{-3} \times 10^{-3})$

has the same value as

$(b_{-1} \times 2^{-1}) + (b_{-2} \times 2^{-2}) + \ldots + (b_{-n} \times 2^{-n})$ - fraction

Multiply by 2, we get

$$\underbrace{(\mathbf{b}_{\mathbf{-1}} \times 2^{0})}_{\text{integer}} + \underbrace{(b_{-2} \times 2^{-1}) + \ldots + (b_{-n} \times 2^{-n+1})}_{\text{fraction}}$$

We get $\mathbf{b}_{\mathbf{-1}}$ which is the integer.

# Explanation of conversion (cont)

Multiply the fraction by 2 again, we get

$$\underbrace{(b_{-2} \times 2^{0})}_{\text{integer}} \quad + \quad \underbrace{(b_{-3} \times 2^{-1}) + \ldots + (b_{-n} \times 2^{-n+2})}_{\text{fraction}}$$

We get $\mathbf{b_{-2}}$ which is the integer.

Thus the bits $b_{-1}$ , $b_{-2}$ , $b_{-3}$ , …, $b_{-n}$ are obtained <u>in sequence</u> by repeated multiplication

## Conversion from hex (octal) to binary

- replace each hex (octal) digit by the corresponding 4-bit (3-bit) binary equivalent

## Conversion from binary to hex (octal)

- Starting from the LSB, replace every 4 bits (3 bits) by the corresponding hex (octal) digit
- Pad MSB with 0's if necessary

Each octal digit represents a group of 3 bits.

| Binary | | | Octal |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

**Examples**

$\underbrace{110}\ \underbrace{011}\ \underbrace{100}_2$

$= 634_8$

correct:

$\underbrace{10}\ \underbrace{100}_2$

$= 24_8$

**Wrong!**

$\underbrace{101}\ \underbrace{00}_2$

$= 50_8$

Each hexadecimal digit represents 4 bits.

| Binary | | | | Hex (Dec) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | A (10) |
| 1 | 0 | 1 | 1 | B (11) |
| 1 | 1 | 0 | 0 | C (12) |
| 1 | 1 | 0 | 1 | D (13) |
| 1 | 1 | 1 | 0 | E (14) |
| 1 | 1 | 1 | 1 | F (15) |

Learners should not fear hexadecimal numbers.

Just treat a hex number as a short form. Each hex digit simply replaces 4 bits.

**Examples:**

$Abc_{16}$ = $1010\ 1011\ 1100_2$

$CAFE_{16}$ = $1100\ 1010\ 1111\ 1110_2$

$C130_{16}$ = $1100\ 0001\ 0011\ 0000_2$

$d24_{16}$ = $1101\ 0010\ 0100_2$

Either upper or lower case may be used for the hex digits a-f

A space is usually inserted between every 4 bits to improve readability

More examples:

| Binary | Octal | Hex |
|---|---|---|
| 101010001 | 521 | 151 |
| 10000001 | 201 | 81 |
| 11011 | 33 | 1B |
| 111001 | 71 | 39 |
| 111111111 | 777 | 1FF |
| 1110111 | 167 | 77 |
| 10010011 | 223 | 93 |

# Exercise

1. Convert $1011001111_2$ to hexadecimal

2. Convert $19.25_{10}$ to binary

Work on these before checking the answers on the next page

# Answers

1. Convert $1011001111_2$ to Hex

   10 1100 1111 = 0010 1100 1111

   $= 2CF_{16}$

2. Convert $19.25_{10}$ to binary

   $19_{10} = 2^4 + 2^1 + 2^0$

   $= 10011_2$

   $0.25_{10} = 2^{-2}$

   $= 0.01_2$

   Thus $19.25_{10} = 10011.01_2$

**Try this online tool.
It provides explanation for the conversion.**

https://www.mathportal.org/calculators/numbers-calculators/decimal-binary-hexadecimal-converter.php

# 2b. Codes

- Students are required to do self-study for this topic.
- Essential concepts will be discussed in Tutorial 1.

# Quick links to each section

## Encoding

Numbers, letters or words are represented by a special group of symbols. A group of symbols is called a code.

For example, you may use the following binary code with your friend:

- 00: let's go eat lunch
- 01: let's go play basketball
- 10: let's go to the library
- 11: let's study for tomorrow's quiz

Both you and your friend must agree what each code word means for it to work.

## Straight binary coding

In digital systems, numbers are probably the most common type of information that need to be represented.

It is very common to represent a numerical value in binary, i.e. base-2.

e.g. the decimal value 35 is simply represented as 100011 in binary. This is called straight binary coding or simply binary coding.

Note: $35_{10}$ = 2^5 + 2^1 + 2^0

There are other commonly used codes for representing numbers.

# Binary-Coded-Decimal Code (BCD)

- Encode decimal numbers; combine some features of decimal and binary systems
- Each digit of a decimal number is represented by its 4-bit binary equivalent
- The legitimate digits are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9
- Since the largest decimal digit is 9, 4 bits are required for each digit.

| **Decimal digit** | **BCD equivalent** | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |

**Notice that the following bit patterns are illegal in BCD code:**

1010

1011

1100

1101

1110

1111

BCD code is used in digital machines whenever decimal information is either applied as inputs or displayed as outputs

# Representation in BCD

**E.g. Represent decimal 957 in BCD**

- Decimal 9 = 1001 in BCD
- Decimal 5 = 0101 in BCD
- Decimal 7 = 0111 in BCD
- Thus decimal 957 = **1001 0101 0111** in **BCD**
- Contrast this with

  decimal 957 = **11 1011 1101** in **straight binary**

# Characteristics of BCD

- Relative ease of conversion
- Consists of **groups of 4-bit codes** for decimal digits 0-9
- Important from hardware standpoint – logic circuits perform conversion to and from decimal digits, all digits can be converted simultaneously
- E.g. converting 957 to binary requires repeated division, but converting it to BCD does not.

BCD code is <u>not</u> used in

- **High speed computers**
  - it requires more bits than binary, and is therefore less efficient
  - E.g. decimal 3 in binary is 11, but decimal 3 in BCD is 0011
  - **arithmetic** processes represented in BCD code are more complicated and slower

# Exercise

1. Convert $34_{10}$ to BCD

2. Convert $19.25_{10}$ to BCD

Answers:

0011 0100

0001 1001. 0010 0101

# Gray code

- Belongs to a class of codes called **minimum-change** codes
- Only 1 bit in the code group changes when going from 1 step to the next
- **Unweighted** code: Bit positions do not have any specific weight (contrast with position-value numbers)
- Usually **cyclical**: the last codeword and the first codeword only has 1 bit difference

| Decimal | 4-bit Gray code | | | |
|---|---|---|---|---|
| **0** | 0 | 0 | 0 | **0** |
| **1** | 0 | 0 | **0** | 1 |
| **2** | 0 | 0 | 1 | **1** |
| **3** | 0 | **0** | 1 | 0 |
| **4** | 0 | 1 | 1 | **0** |
| **5** | 0 | 1 | **1** | 1 |
| **6** | 0 | 1 | 0 | **1** |
| **7** | **0** | 1 | 0 | 0 |
| **8** | 1 | 1 | 0 | **0** |
| **9** | 1 | 1 | **0** | 1 |
| **10** | 1 | 1 | 1 | **1** |
| **11** | 1 | **1** | 1 | 0 |
| **12** | 1 | 0 | 1 | **0** |
| **13** | 1 | 0 | **1** | 1 |
| **14** | 1 | 0 | 0 | **1** |
| **15** | **1** | 0 | 0 | 0 |

Only 1 bit is changed from one value to the next

## Example - Error occurring while using BCD

- what happens when a number increments from 1 to 2?

| | BCD code | | | |
|---|---|---|---|---|
| **Dec** | **b3** | **b2** | **b1** | **b0** |
| **1** | **0** | **0** | **0** | **1** |
| | | | | |
| **2** | **0** | **0** | **1** | **0** |

# Example - Error occurring while using BCD

- what happens when a number increments from 1 to 2?

Ideally, the bits b1 and b0 change at the same time:

| Dec | b3 | b2 | b1 | b0 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 |
| | | | | |
| 2 | 0 | 0 | 1 | 0 |

**No error**

- possible actual case, bit b0 <u>changes before</u> b1:

| Dec | b3 | b2 | b1 | b0 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | <u>1</u> |
| | | | | |
| 0 | 0 | 0 | <u>0</u> | 0 |
| | | | | |
| 2 | 0 | 0 | 1 | 0 |

Transition error due to different speeds of bit change – **race condition**

**Solution** - Error occurring while using BCD

- Use Gray code instead and there will be no such problem since only 1 bit (b1) is changed when the number increments from 1 to 2

| | Gray code | | | |
|---|---|---|---|---|
| **Dec** | **b3** | **b2** | **b1** | **b0** |
| **1** | **0** | **0** | **<u>0</u>** | **1** |
| | | | ⇩ | |
| **2** | **0** | **0** | **1** | **1** |

Gray code is useful in situations where multiple bit change may lead to error.

Gray code is **not** suitable for **arithmetic operations**.

You may read section 2.11 of the textbook by Wakerly for details.

# Alphanumeric Codes

- **Codes that represent**
  - alphabet (e.g. a, b, c, …, z)
  - punctuation marks
  - special characters and numbers
- **A complete set of alphanumeric code must include**
  - 26 lowercase letters (a – z)
  - 26 uppercase letters (A – Z)
  - 10 numeric digits (0 – 9)
  - 7 punctuation marks
  - 20 – 40 other characters such as +, -, /, <, #, %, …

# ASCII Code

- Most widely used alphanumeric code
- 7-bit code, hence 128 (=$2^7$) possible code symbols
- There is also the 8-bit extended ASCII code
- Used for transferring alphanumeric data between digital devices
- Used in digital computers to store alphanumeric characters

Students are **NOT required** to memorize the ASCII table

## American Standard Code for Information Interchange (ASCII)

| | $b_7b_6b_5$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $b_4b_3b_2b_1$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 0000 | NUL | DLE | SP | 0 | @ | P | ` | p |
| 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | STX | DC2 | " | 2 | B | R | b | r |
| 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | BEL | ETB | ' | 7 | G | W | g | w |
| 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 1001 | HT | EM | ) | 9 | I | Y | i | y |
| 1010 | LF | SUB | * | : | J | Z | j | z |
| 1011 | VT | ESC | + | ; | K | [ | k | { |
| 1100 | FF | FS | , | < | L | \ | l | \| |
| 1101 | CR | GS | – | = | M | ] | m | } |
| 1110 | SO | RS | . | > | N | ^ | n | ~ |
| 1111 | SI | US | / | ? | O | _ | o | DEL |

**b7 is MSB, b1 is LSB**

Example:

**Mp3** encoded into ASCII, will become

$$\mathbf{\underbrace{1001101}_{M}\underbrace{1110000}_{p}\underbrace{0110011}_{3}}$$

Note: Lower case and upper case alphabets have different codes

Often times, hexadecimal digits are used to represent ASCII codes.

Example:

| **Character** | **ASCII** | **expressed in Hex** |
|---|---|---|
| **M** | 0100 1101 | 4D |
| **p** | 0111 0000 | 70 |
| **3** | 0011 0011 | 33 |

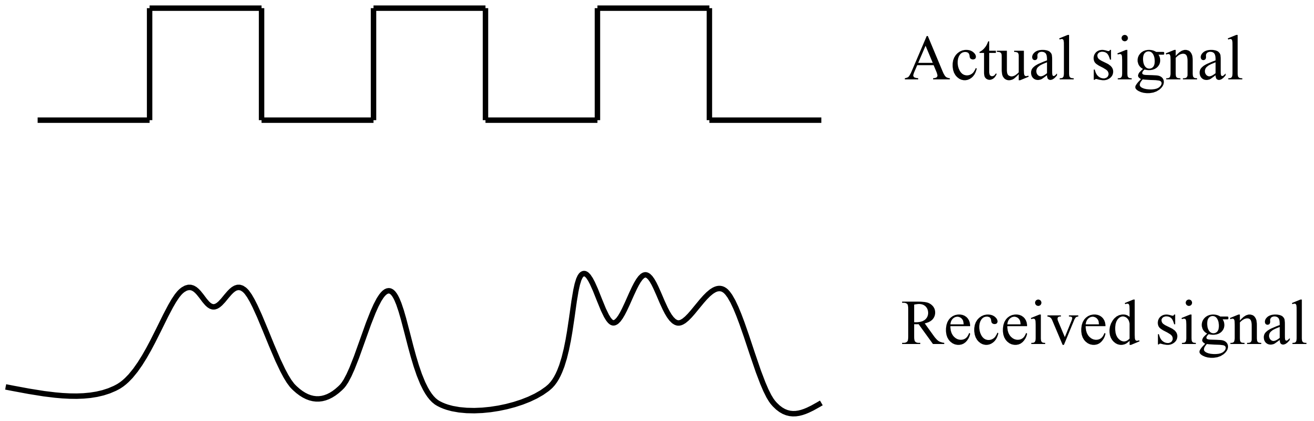
0 is padded to MSB before converting to Hex

# Different ways to represent $14_{10}$

- Straight binary: 1110
- Hexadecimal: E or e
- Octal: 16
- BCD: 0001 0100
- ASCII: 0110001 0110100
- Gray code: 1001
- Note the importance of knowing which representation is being used

# Parity Method for Error Detection

- Transfer of binary data from one location to another can be corrupted by noise.



- **Result of error:**

  transmitted 0 becomes 1 at the receiver

  transmitted 1 becomes 0 at the receiver

- **e.g. 1010** wrongly received as **1011,** or

  **1100** wrongly received as **1000**

- Multiple bit errors cannot be detected by this simple parity method. They are much less likely to happen than single bit errors.

# Parity Bit

- An extra bit attached to a code group
- It forms part of the code being transmitted
- The parity bit is made either 0 or 1
- **Even parity** make total no. of ‘1’ bits even BEFORE transmitting

$0 \quad \underbrace{0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1}$

**Parity bit**

**4 bits of 1 in data** (4 is an even number)

- **Odd parity** make total no. of ‘1’ bits <u>odd</u> BEFORE transmitting

| 1 | 1 1 0 1 1 1 1 |
|---|---|
| **Parity bit** | **6 bits of 1 in data** (7 is an odd number) |

- Is able to detect single bit error only

- Receiver and transmitter must **agree** on odd/ even parity scheme

## Examples:

| 7-bit data to be transmitted | 8-bits are transmitted after adding parity bit | |
|---|---|---|
| | **Odd parity** | **Even parity** |
| 1001000 | 11001000 | 01001000 |
| 0011100 | 00011100 | 10011100 |
| 0101110 | 10101110 | 00101110 |
| 1010111 | 01010111 | 11010111 |

# Example: Limitation of parity method

- Transmitter and receiver agree on even parity system
- Data to be transmitted: 1010111
- Data transmitted with parity bit: <u>1</u>1010111
- Actual data received corrupted by noise: 11000111 – one bit error
- Receiver checks parity: odd
- Receiver correctly concludes data in error
- If actual data received: 01010110 – two bit error
- Receiver checks parity: even
- Receiver wrongly concludes data no error!

# Commonly Used Prefixes

**SI units**

- k (kilo) = $10^3$
- M (mega) = $10^6$
- G (giga) = $10^9$

**JEDEC**

- K (kilo) = $2^{10}$
- M (mega) = $2^{20}$
- G (giga) = $2^{30}$
- T (tera) = $2^{40}$

**IEC**

- Ki (kibi) = $2^{10}$
- Mi (mebi) = $2^{20}$
- Gi (gibi) = $2^{30}$
- Ti (tebi) = $2^{40}$

Binary prefix - Wikipedia, the free encyclopedia

# Commonly Used Prefixes (cont)

**Metric system**

- m (milli) = $10^{-3}$
- μ (micro) = $10^{-6}$
- n (nano) = $10^{-9}$
- p (pico) = $10^{-12}$

**Example:**

- 0.1 μs = 100 ns = $10^{-7}$ second
- 200 mV = 0.2 V

# About significant figures

The value of pi is 3.1415926…

It can also be written as

- 3.14159 (6 significant figures)
- 3.1416 (5 sf)
- 3.142 (4 sf)
- 3.14 (3 sf)
- 3.1 (2 sf)
- 3 (1 sf)