Week 1- T1 2020

# Introduction/Number Systems/Combinational Circuits

ELEC2141: Digital Circuit Design

# Staff and consultation

**Course coordinator:**

Dr. Daniel Ssu-Han Chen
EE&T 346, ssu-han.chen@unsw.edu.au
Consultation hours:
Wednesday 1am-2pm

**Tutor:**

Dr. Daniel Ssu-Han Chen
EE&T 346, ssu-han.chen@unsw.edu.au
Dr. Aron Michael
EE&T 316, a.michael@unsw.edu.au

**Laboratory coordinator:**

Yen Nee Ho, yennee.ho@unsw.edu.au

# Course format

Formal face-to-face lectures (twice a week)
Lecture videos

Tutorials (begin Week 2)

Fortnight online quizzes

Pre-lab quizzes (need to be completed prior to lab)

Laboratory sessions (begin Week 3)

Midterm exam (Week 5)

Design Assignments (due Week 6 and 9)

# Assessment

| | |
|---|---|
| Fortnight online quizzes | 5% |
| Laboratory practical experiments | 15% |
| Lab examination | 5% |
| Assignments (I & II) | 20% |
| Midterm exam (1 hour) | 15% |
| Final Exam (2 hours) | 40% |

# Course introduction

Will cover the analysis & design of digital circuits

Digital circuits manipulate discrete or digital voltage instead of continuous or analog voltages

They are used in personal computers, mobile phone, embedded control system

# Overview

Information Representation

Introduction to digital circuits

Number systems

Binary numbers

Number conversion between bases

Binary codes

Logical gates

Boolean functions

**Reading: Mano - Chapter 1, Chapter 2, 2.1-2.2**

# Information representation

Information represents physical parameters or man-made parameters

Known as signals

Most physical parameters are continuous- can take all possible values over a defined range

Example – temperature, humidity

Man-made parameters are discrete – can take only finite values over a defined range

Example – currency

# Signal digitisation

Physical parameter

↓

Sensor

↓

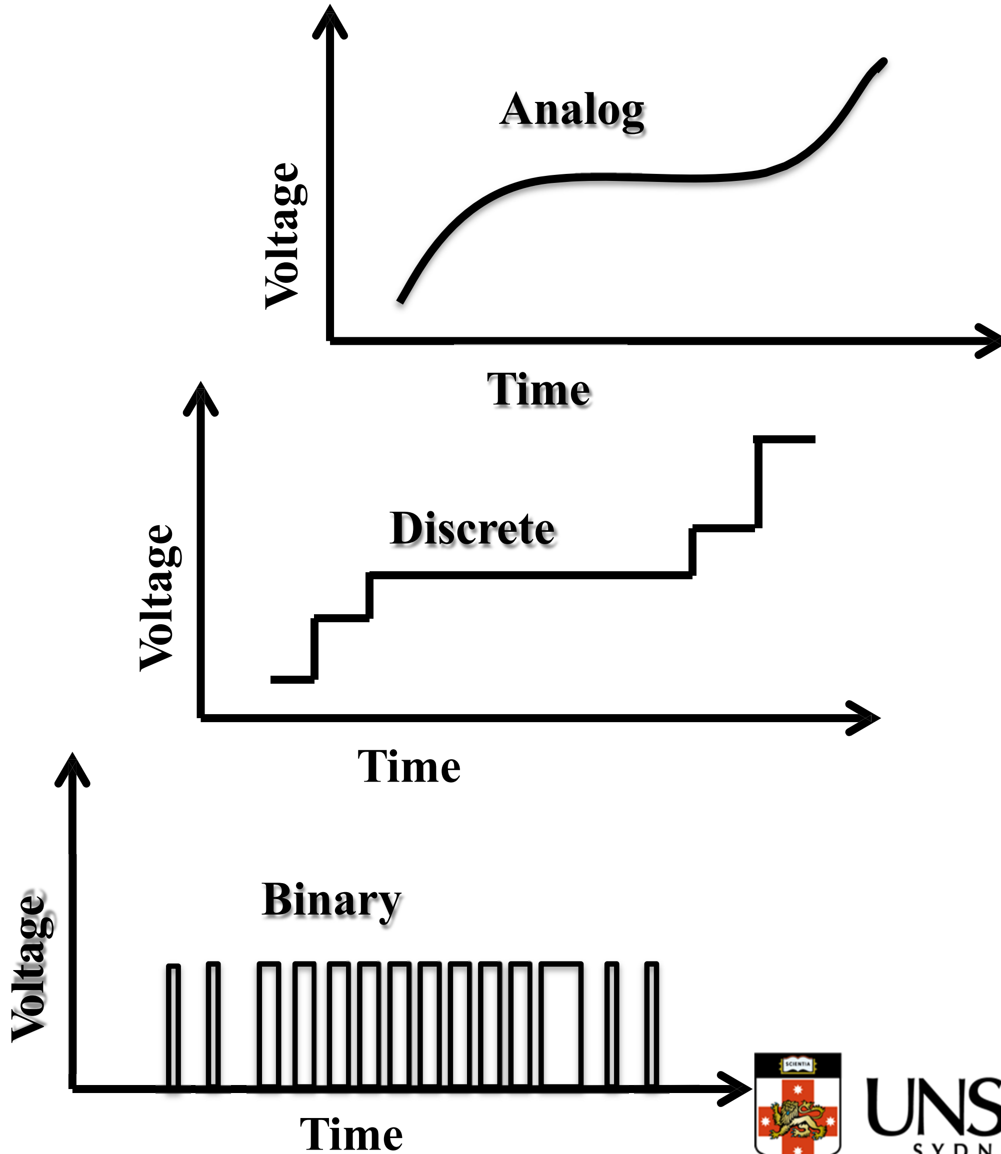
Analog voltage

↓

Discrete voltage

↓

Conversion to binary



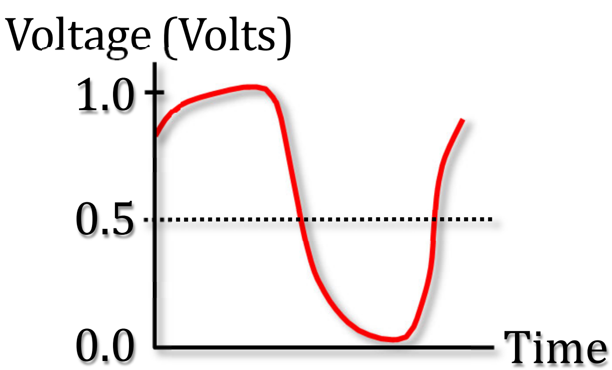
# Digital circuits

Digital Circuits can understand (manipulate and store) signals as only binary

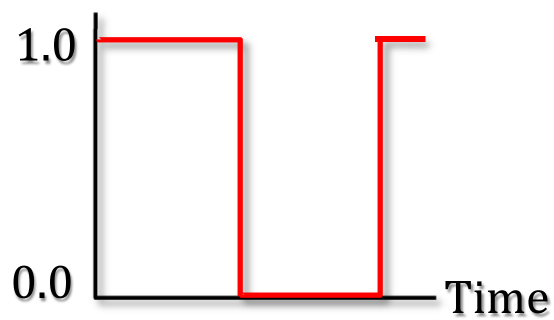
Signals in digital circuits are in fact analog, but interpreted as binary

Binary signals can have amplitudes in only two ranges

The information associated with these ranges is represented by the voltage values of HIGH/LOW



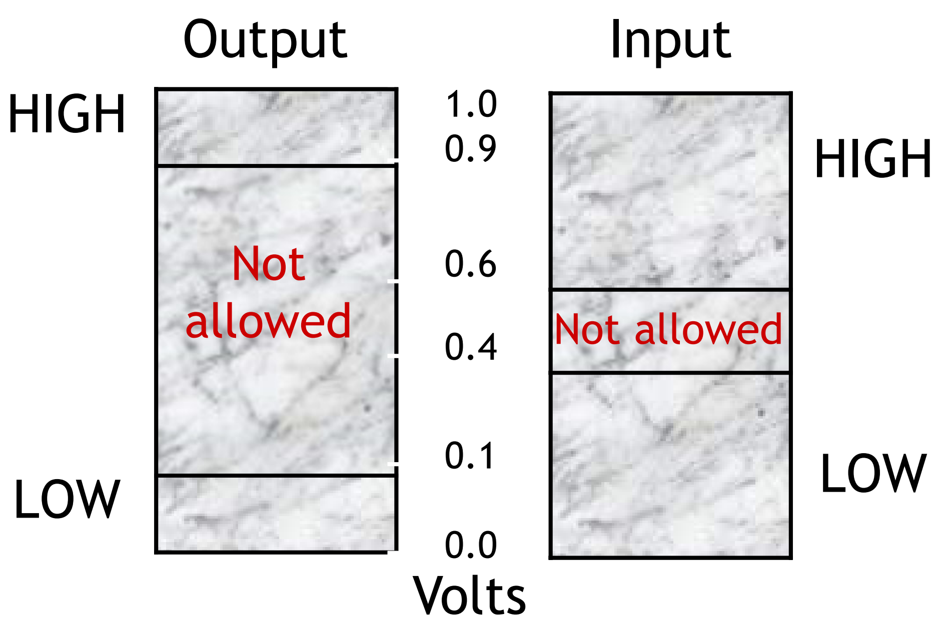

**Time-dependent voltage**




**Binary model of Time-dependent voltage**

# Binary signals

Binary signals are identified as HIGH/1/TRUE/On or LOW/0/FALSE/Off based on their range of values




**Example voltage ranges**

Output [0.9,1.0] – HIGH
Output [0,0.1]– LOW

Input [0.6 ,1.0] – HIGH
Input [0,0.4]- LOW

Noise margin – difference in range of values for input and output

UNSW SYDNEY

# Positive vs negative logic

Usually, the higher amplitude represents 1 and the lower amplitude represents 0 (called positive logic)

In negative logic, the lower amplitude represents 1 and the higher amplitude represents 0

The logic value 1 is also called high, true, or on

The logic value 0 is also called low, false, or off

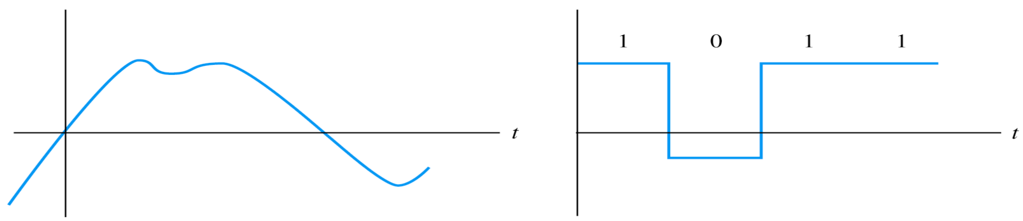
# Digital voltage values

| | TTL | | 3.3 V CMOS | | 5V CMOS | |
|---|---|---|---|---|---|---|
| | LOW | HIGH | LOW | HIGH | LOW | HIGH |
| INPUT | 0 - 0.8 | 2 - 5 | 0 - 0.8 | 2 - 3.3 | 0 - 1.5 | 2.7 - 5 |
| OUTPUT | 0 - 0.5 | 2.7 - 5 | 0 - 0.4 | 2.4 -3.3 | 0 - 0.5 | 3.5 - 5 |
| Noise Margin | 0.3 | 0.7 | 0.4 | 0.4 | 1.0 | 0.8 |

# Advantages of digital signals

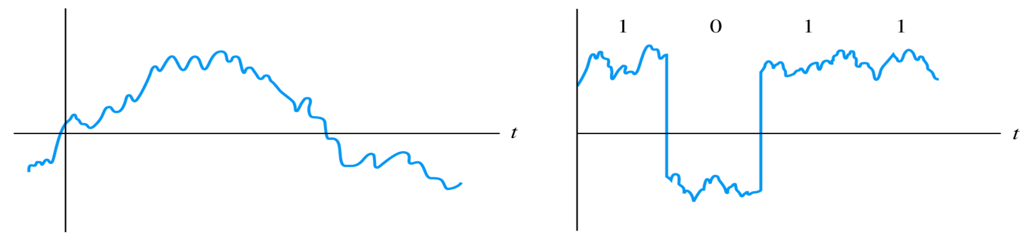
Even when noise in signal is large, logic values can still be determined in presence of noise unlike analog signals

Also significantly more economical




(a) Analog signal

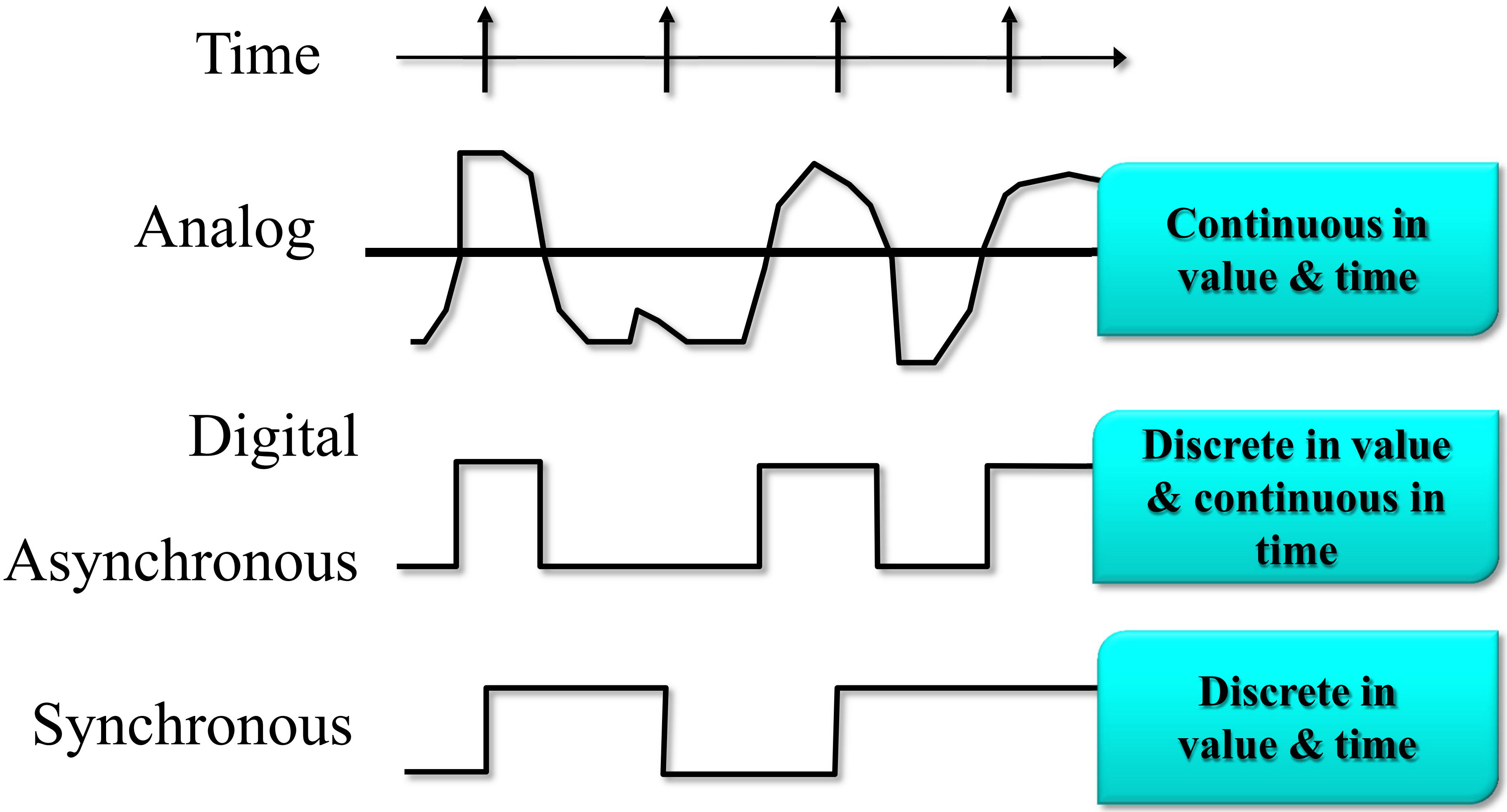
(b) Digital signal




(c) Analog signal plus noise

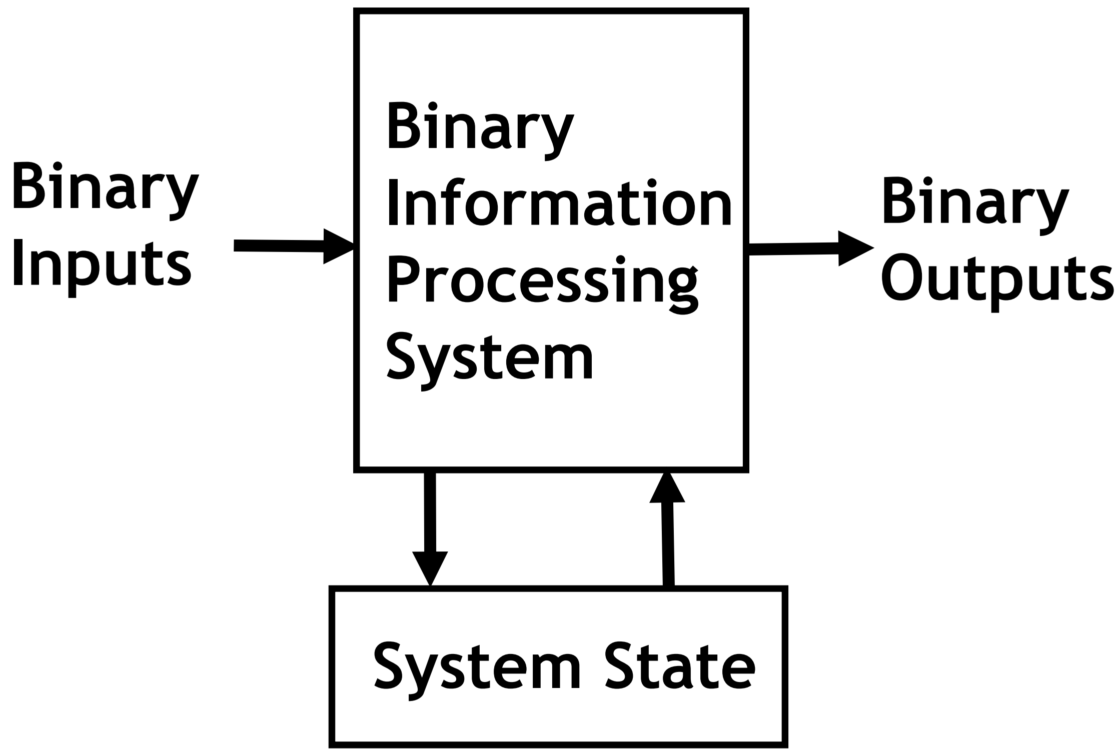
(d) Digital signal plus noise

# Signal over time example



# Digital circuits

Digital circuits have binary inputs, process binary signals, store binary signals (states), and generate binary outputs.



# Types of digital circuits

Combinational logic circuits

No state present

Sequential logic circuits

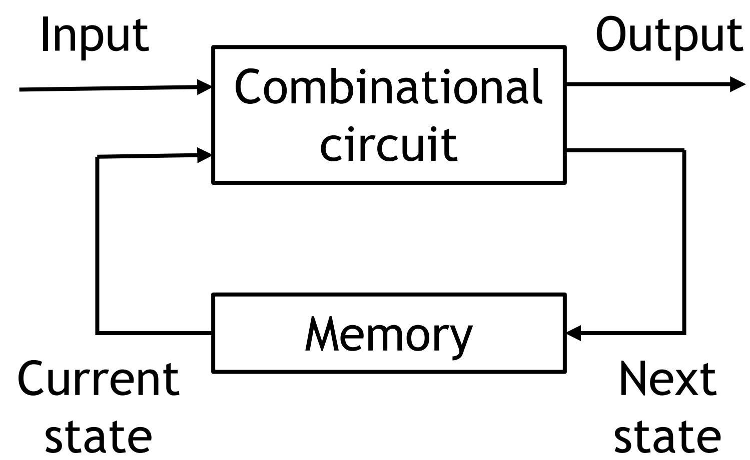
State present

State updated at discrete times

=> Synchronous Sequential circuits

State updated at any time

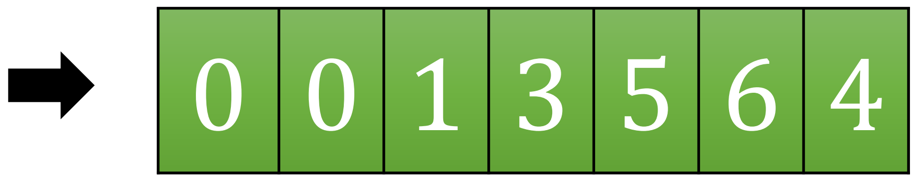
=> Asynchronous Sequential circuits



# Digital system example

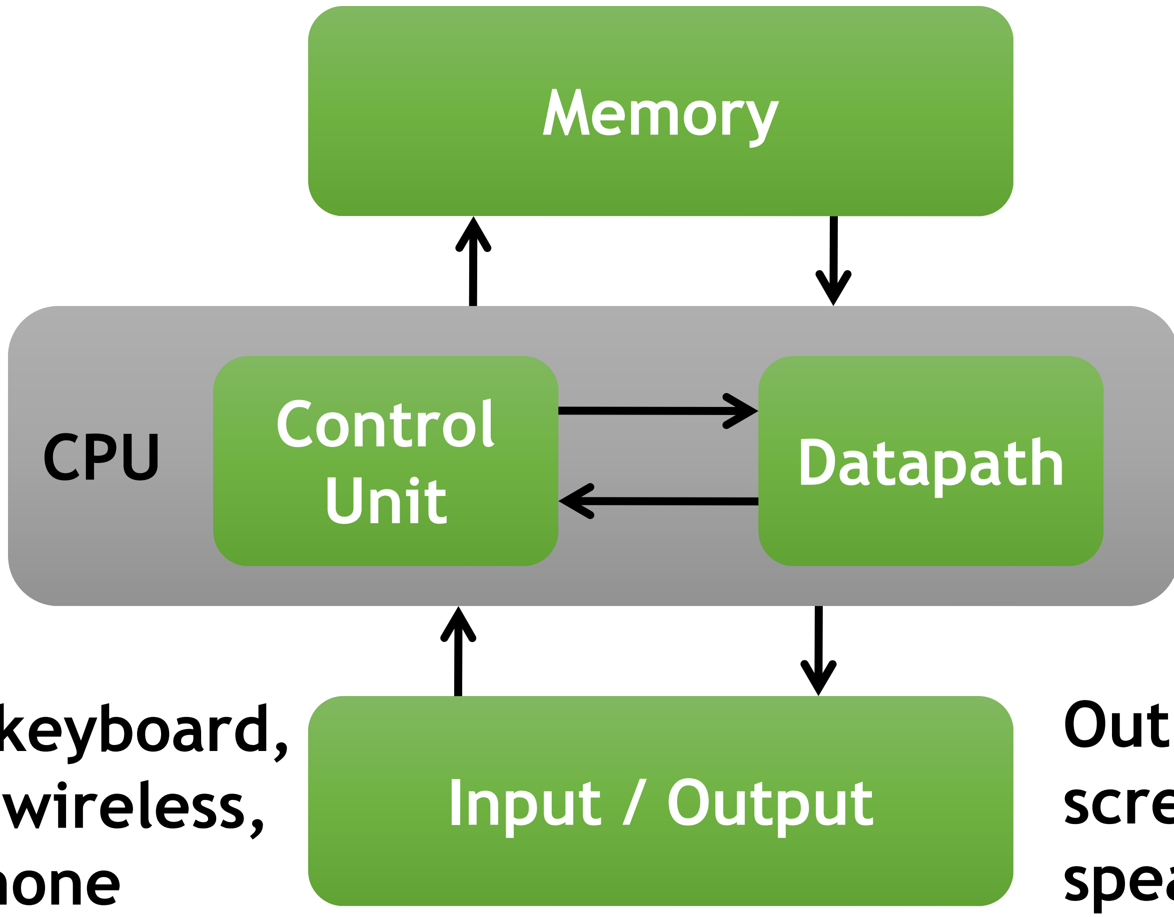
Digital Counter

Count Up
Reset




Inputs: Count Up, Reset
Outputs: Visual Display
State: “Value” of stored digits

# Digital computer




Inputs: keyboard, mouse, wireless, microphone

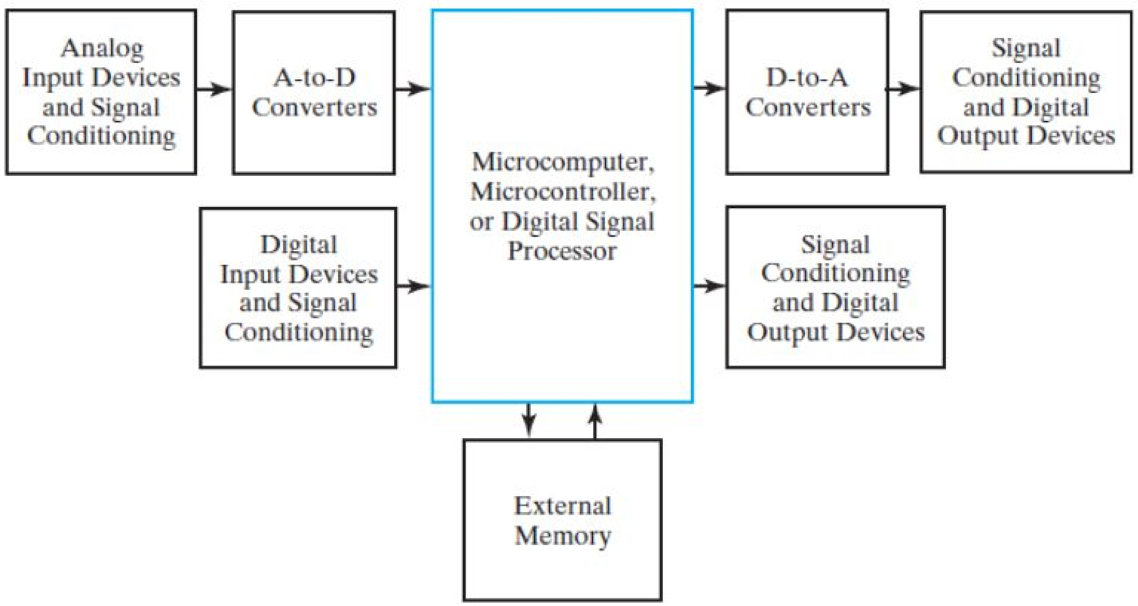
Outputs: LCD screen, wireless, speakers

# And beyond - embedded systems

Computers as integral parts of other products

Examples of embedded computers

- Microcomputers, microcontrollers, digital signal processors



# Binary physical quantities

Examples of some physical representations for binary 0 and 1 include

| | |
|---|---|
| CPU: | *Voltage* |
| Disk: | *Magnetic Field Direction* |
| CD: | *Surface Pits* |
| Dynamic RAM: | *Electrical Charge* |

# Hierarchy in a digital system

Digital system

Digital modules:
Adders, Memories, Registers, state machines

Logic gates:
AND, OR, NAND, NOR, NOT

Transistors:
BJT (TTL) and CMOS, …

IC Technology (VLSI technology):
Bipolar, CMOS, …

Semiconductor foundry:
Intel, GF, TSMC, IBM, …

# Number systems

A decimal number $724.5_{10}$ can be represented as

$$724.5_{10} = 7x10^{2} + 2x10^{1} + 4x10^{0} + 5x10^{-1}$$

A decimal number with n digits to the left of the decimal point and m digits to the right is represented as

$$A_{n-1}A_{n-2}\ldots A_1A_0.A_{-1}A_{-2}\ldots A_{-m+1}A_{-m}$$

$A_i$ = 0,1,2,3,4,5,6,7,8 or 9

A number in base or radix r with n-digits to the left of the radix point and m-digits to the right is represented by a string of coefficients and expressed as a power series in r

$$(number)_r = \left(\sum_{i=0}^{i=n-1} A_i \cdot r^i\right) + \left(\sum_{i=-m}^{j=-1} A_j \cdot r^j\right)$$

UNSW SYDNEY

# Binary numbers

Binary (base 2) numbers are widely used in digital systems

Base 2 system only has two possible digits:
0 and 1

Digits in binary numbers are called *bits* (Binary Digits)

The rightmost bit in a binary number is referred to as the *Least Significant Bit* (LSB)

Similarly, the leftmost bit is referred to as the *Most Significant Bit* (MSB)

# Binary to decimal example

Example:

$11010_2 =$

In practice, when calculating binary values, omit all 0-bits and add values for 1-bits

Example (with fractions):

$110101.11_2 =$

# Problem

Determine the decimal values of

$0111011110_2 =$

$100101.101_2 =$

# Positive powers of 2

Useful for base conversion

Important to remember!

| Exponent | Value |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1,024 |

| Exponent | Value |
|---|---|
| 11 | 2,048 |
| 12 | 4,096 |
| 13 | 8,192 |
| 14 | 16,384 |
| 15 | 32,768 |
| 16 | 65,536 |
| 17 | 131,072 |
| 18 | 262,144 |
| 19 | 524,288 |
| 20 | 1,048,576 |
| 21 | 2,097,152 |

Memorize if you can (and have too much time on your hands)

# Special powers of 2

$2^{10}$ (1,024) is Kilo, denoted *"K"*

$2^{20}$ (1,048,576) is Mega, denoted *"M"*

$2^{30}$ (1,073,741,824) is Giga, denoted *"G"*

$2^{40}$ (1,099,511,627,776) is Tera, denoted *"T"*

# Decimal to binary integer

Repeatedly divide integers by 2 to obtain quotient and remainder

Read remainders in reverse order

Example – convert $41_{10}$ to binary:

# Another conversion example

Can convert decimal to any other base

Example – Convert $388_{10}$ to base 7:

# Problems

Determine the binary value of $455_{10}$

Determine the octal (8) value of $641_{10}$

# Decimal fractions to binary

Repeatedly multiply fractions by 2 to obtain integer product and fraction

Read integer products in order

Example – convert $0.6875_{10}$ to binary:

# Additional issue in fractional part

Note that in this conversion, the fractional part can become 0 as a result of the repeated multiplications

In general, it may take many bits to get this to happen or it may never happen

Example: Convert $0.65_{10}$ to binary

- 0.65 = 0.1010011001001…
- The fractional part begins repeating every 4 steps, yielding repeating 1001 forever!

Solution: Specify number of bits to right of radix point and round/truncate to this number

# Problem

Convert $341.23_{10}$ to binary with 6 bits in fractional part

341.23 = 101010101.001110

0.23 x2 = 0.46 0

0.46x2 = 0.92 0

0.92x2 = 1.84 1

0.84 x 2 = 1.68 1

0.68x2 = 1.36 1

0.36 x 2 = 0.72 0

UNSW SYDNEY

# Conversion between bases

In general, to convert between bases, you need to convert to decimal first, and then from decimal to the target base (exceptions follow)

So, to convert from one base to another:

1) Convert the integer part
2) Convert the fraction part
3) Join the two results with a radix point

# Commonly occurring bases

| Name | Radix | Digits |
|---|---|---|
| Binary | 2 | 0, 1 |
| Octal | 8 | 0, 1, 2, 3, 4, 5, 6, 7 |
| Decimal | 10 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| Hexadecimal | 16 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |

The six letters A, B, C, D, E, and F in hexadecimal represent the digits for values 10, 11, 12, 13, 14, 15 respectively

Usually, the prefix 0x is added to indicate a hexadecimal number – e.g. 0xA5

# Numbers in different bases

| Decimal (Base 10) | Binary (Base 2) | Octal (Base 8) | Hexadecimal (Base 16) |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

# Hexadecimal to binary and back

## Hexadecimal (or Octal) to Binary:

Restate each hexadecimal (octal) digit as the corresponding four (three) bits starting at the radix point and going both ways

## Binary to Hexadecimal (or Octal):

Group the binary digits into four (three) bit groups starting at the radix point and going both ways, padding with zeros as needed in the fractional part

Convert each group of four (three) bits to a hexadecimal (octal) digit

| | |
|---|---|
| 010 111 011 001 | 2731 - octal |
| 0101 1101 1001 | 5D9 - HEXA |

UNSW SYDNEY

# Octal to hexadecimal via binary

Example: Convert the octal number $26153.7406_8$ to binary and therefore to hexadecimal

# Problem

Convert $2B.5_{16}$ to binary and octal

# Binary coding

Flexibility of representation

- Within the constraints below, can assign any binary combination (called a code word) to any data as long as data is uniquely encoded

Information Types

- Numeric
- Non-numeric

# Non-numeric binary codes

Given $n$ bits, a binary code is a mapping from a set of represented elements to a subset of the $2^n$ binary numbers

Example:

A binary code for the seven colors of the rainbow

Code 100 is not used

| Color | Binary Code |
|---|---|
| Red | 000 |
| Orange | 001 |
| Yellow | 010 |
| Green | 011 |
| Blue | 101 |
| Indigo | 110 |
| Violet | 111 |

# Number of bits required

Given $M$ elements to be represented by a binary code, the $n$ minimum number of bits needed satisfies the following relationship:

$2^{(n-1)} < M \leq 2^n$ *or* $n \geq \lceil \log_2 M \rceil$

Where this is the *ceiling function* – the integer greater than or equal to the argument

Example: How many bits are required to represent decimal digits with a binary code?

UNSW SYDNEY

# Number of elements represented

Given $n$ digits in radix $r$, there are $r^n$ distinct elements that *can* be represented

But can represent less elements, $m$, such that:

$m < r^n$

Examples:

- You can represent 4 elements in radix $r = 2$ with $n = 2$ digits:
- You can represent 4 elements in radix $r = 2$ with $n = 4$ digits:

# Decimal codes - BCD and gray code

Two useful ways to code decimal digits into binary are *Binary Coded Decimal* (BCD) and *Gray Code*

| Decimal | BCD | Gray Code |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0100 |
| 2 | 0010 | 0101 |
| 3 | 0011 | 0111 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0010 |
| 6 | 0110 | 0011 |
| 7 | 0111 | 0001 |
| 8 | 1000 | 1001 |
| 9 | 1001 | 1000 |

# Binary coded decimal (BCD)

BCD is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9

Every digit in a decimal number is encoded separately and then combined together

Example (BCD *coding*):

$$185_{10} = 0001\ 1000\ 0101$$

Compare with (binary *conversion*):

$$185_{10} = 10111001$$

# Gray code

Gray code is a binary code where two successive values differ in only one bit change

Example – using the gray code assignments given earlier for decimal digits, only one bit changes on the transition from $3_{10}$ to $4_{10}$:

$0111_{GC} \rightarrow 0110_{GC}$

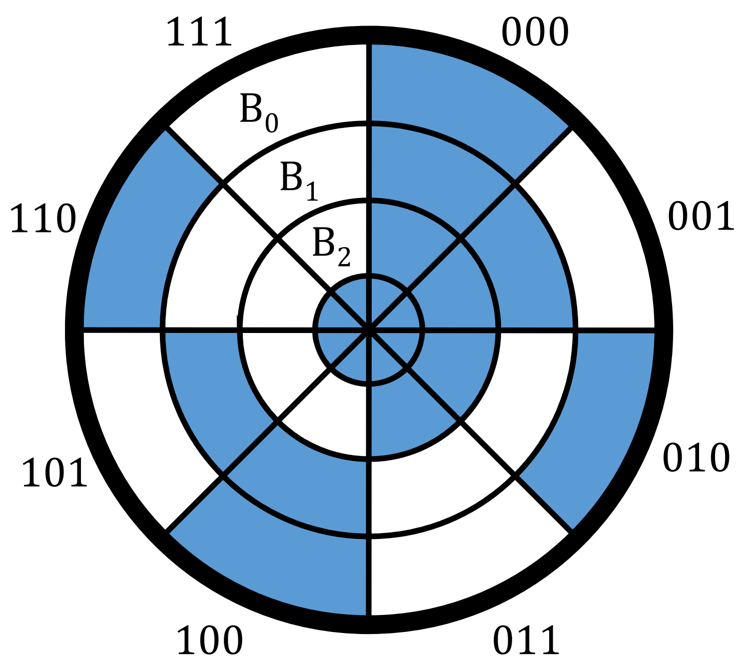
Compare with regular binary code, three bits change on the same transition:

$011_2 \qquad 100_2$

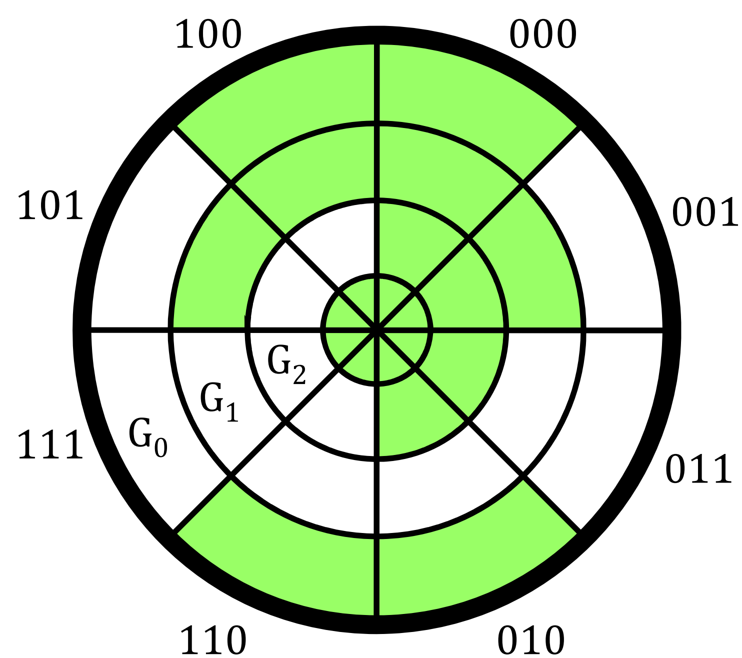
# Optical shaft encoder using gray code

What is it good for?

An example: Optical Shaft Encoder




Binary Code




Gray Code

# Alphanumeric codes

Digital systems need to handle both numeric and non-numeric ( characters or symbols) data.

It is necessary to formulate a binary code for letters of the alphabets, numerals and special characters.

Alphanumeric character set is a set of elements that include the 10 decimal digits, the 26 letters of the alphabet ( lower and upper case), and a number of special characters.

# Alphanumeric codes - ASCII Code

64 to 128 elements in this character set.

Standard binary code for the alphanumeric characters is the American Standard Code for Information Interchange (ASCII), which uses seven bits to code 128 characters.

ASCII includes 10 numerals, 26 upper case letters, 26 lower case letters, 32 special printable characters such as %,@, and $, and 34 non-printing characters.



UNSW SYDNEY

# Alphanumeric codes – ASCII Code

**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 | NULL | 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 |

**Control Characters**

| | | | |
|---|---|---|---|
| NULL | NULL | DLE | Data link escape |
| SOH | Start of heading | DC1 | Device control 1 |
| STX | Start of text | DC2 | Device control 2 |
| ETX | End of text | DC3 | Device control 3 |
| EOT | End of transmission | DC4 | Device control 4 |
| ENQ | Enquiry | NAK | Negative acknowledge |
| ACK | Acknowledge | SYN | Synchronous idle |
| BEL | Bell | ETB | End of transmission block |
| BS | Backspace | CAN | Cancel |
| HT | Horizontal tab | EM | End of medium |
| LF | Line feed | SUB | Substitute |
| VT | Vertical tab | ESC | Escape |
| FF | Form feed | FS | File separator |
| CR | Carriage return | GS | Group separator |
| SO | Shift out | RS | Record separator |
| SI | Shift in | US | Unit separator |
| SP | Space | DEL | Delete |

# Binary logic and gates

Interconnected transistors form logic gates

Each gate has inputs and an output. It performs a specific logical operation on its binary inputs and provides a binary value at the output

Inputs and output of a logic gate are designated by alphabetical variables

These variables can assume only 1 or 0 values and are known as binary variables

Three basic logical operations : AND, OR, and NOT

# AND gate

AND is represented by a dot or an absence of an operator

$$Z = X \cdot Y = XY = X \wedge Y$$

$Z$ is 1 if and only if $X = 1$ and $Y = 1$; otherwise $Z = 0$

| $X \cdot Y$ |
| --- |
| 0·0 = 0 |
| 0·1 = 0 |
| 1·0 = 0 |
| 1·1 = 1 |

Can be extended to more than two input binary variables

Also called logical multiplication

# OR gate

OR is represented by a plus symbol or "v"

$$Z = X + Y = X \vee Y$$

$Z$ is 1 if $X = 1$ or $Y = 1$ ; $Z$ is 0 if $X = 0$ and $Y = 0$

| $X + Y$ |
|---|
| 0+0 = 0 |
| 0+1 = 1 |
| 1+0 = 1 |
| 1+1 = 1 |

OR logical operation can be extended to more than two input binary variables

Also called logical addition

UNSW SYDNEY

# NOT gate

NOT is represented by a bar over the variable

$$Z = \bar{X}$$

$Z$ is 1 if $X = 0$ and $Z$ is 0 if $X = 1$

| $X$ |
|---|
| $\bar{0}$ = 1 |
| $\bar{1}$ = 0 |

Also called an inverter

# Truth tables

A *Truth Table* is a tabular form that uniquely represents the relationship between the input variables of a function and its output

A function *F* that depends on *n* variables will have $2^n$ rows

AND

| $X$ | $Y$ | $Z = X \cdot Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR

| $X$ | $Y$ | $X + Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT

| $X$ | $Z = \overline{X}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Problem

Building a Truth Table for 3 and 4 input variables

| $X$ | $Y$ | $Z$ | $F$ |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

| $W$ | $X$ | $Y$ | $Z$ | $F$ |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# Logic gates

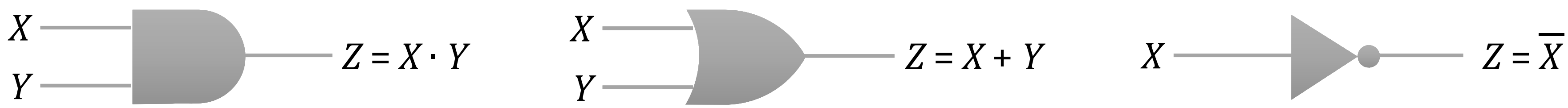
Logical gates are electrical circuits that implement logical operations

The input terminals accept voltage signals within allowable range (as a binary signal) and gives out at the output terminal a binary signal that also falls with in the allowable range

Intermediate values are crossed only during transitions from 0 to 1 or otherwise

# Graphical gate representation

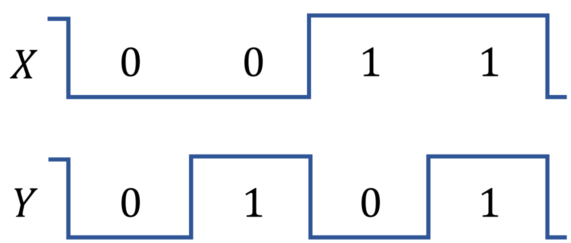
Graphical (symbolical) representations of the three types of gates: AND, OR, NOT




AND gate

OR gate

NOT gate (inverter)




Time diagram shows waveform behavior

(AND) $X \cdot Y$

(OR) $X + Y$

(NOT) $\overline{X}$

# Boolean algebra

*Boolean Algebra* is an algebra dealing with binary variables and logic operations.

A *Boolean expression* is an algebraic expression formed by using binary variables, the constants 0 and 1, the logic operation symbols, and parentheses

The order of evaluation in a Boolean expression:

( ), NOT, AND, OR

# Boolean functions

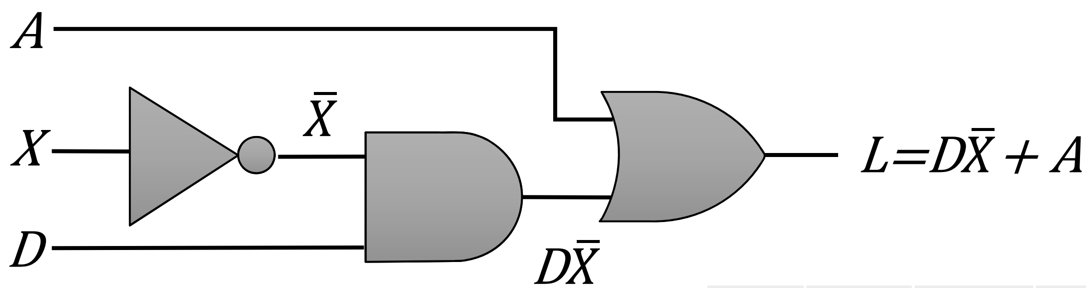
A Boolean function is a Boolean equation consisting of a binary variable identifying the function followed by an equal sign and Boolean expression

$$\underbrace{L(D,X,A) = \overbrace{D\bar{X} + A}^{\text{Boolean expression}}}_{\text{Boolean function or Boolean equation}}$$

A Boolean function can be transformed into a circuit diagram (logic diagram) composed of logic gates and interconnected by wires

# Boolean functions

Logic Diagram




A Boolean function can be represented by a truth table

It uniquely represents the relationship between the input variables of a function and its output

A function $F$ that depends on $n$ variables will have $2^n$ rows

| $D$ | $X$ | $A$ | $L = D\bar{X} + A$ |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

UNSW SYDNEY

# Boolean functions

The Boolean function, however, can be expressed by various Boolean equations, which are not the same but equivalent. e.g. $L$ and $F$ have the same function and truth table

$$L(D, X, A) = D\bar{X} + A$$
$$F(D, X, A) = D\bar{X} + AD + A\bar{D}$$

The Boolean equation dictates the interconnection of gates in the logic circuit diagram

Simpler expression reduces the number of gates and the number of inputs into the gates

# Problem

Use a truth table to find $F$ where

$$F(A,B,C) = A\bar{C} + C(A+B) + \overline{BC}$$

| $A$ | $B$ | $C$ | $A\bar{C}$ | $A+B$ | $C(A+B)$ | $BC$ | $\overline{BC}$ | $\boldsymbol{F}$ |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

# Combinational logic circuit

$$F(D, X, A) = D\bar{X} + AD + A\bar{D}$$

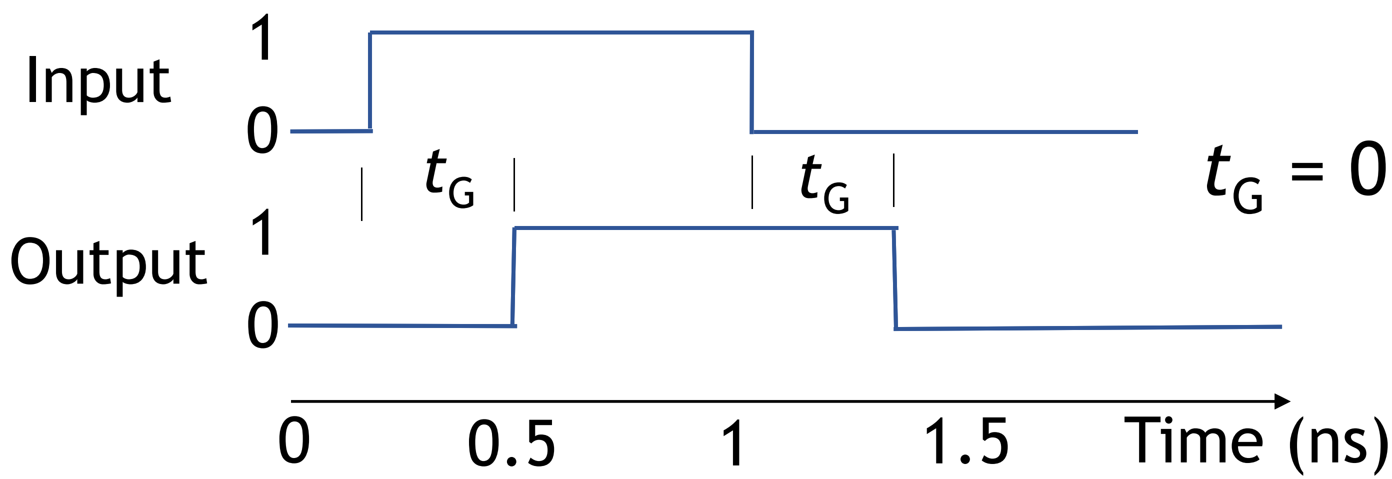
A combinational logic circuit can be constructed to implement a Boolean function $F$, by appropriately connecting input signals and logic gates:

- Circuit input signals → from function variables $(D, X, A)$
- Circuit output signal → function output $F$
- Logic gates → from logic operations

# Gate delay

In reality, there is always a gate delay

Gate delay – the length of time it takes for an input changes to result in the corresponding output change




$t_G$ = 0.3 ns

Gate delay is a function of gate type, number of inputs, underlying technology, and circuit design of the gate.

# Basic identities of Boolean algebra

| | | |
|---|---|---|
| 1. $X + 0 = X$ | 2. $X \cdot 1 = X$ | Identity |
| 3. $X + 1 = 1$ | 4. $X \cdot 0 = 0$ | Null |
| 5. $X + X = X$ | 6. $X \cdot X = X$ | Idempotence |
| 7. $X + \overline{X} = 1$ | 8. $X \cdot \overline{X} = 0$ | Complementarity |
| 9. $\overline{\overline{X}} = X$ | | Involution |

AND

| $X$ | $Y$ | $Z = X \cdot Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR

| $X$ | $Y$ | $X + Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# Basic identities of Boolean algebra

| | | |
|---|---|---|
| 10. $Y+X = X+Y$ | 11. $Y \cdot X = X \cdot Y$ | Commutative |
| 12. $X + (Y + Z) = (X+Y) + Z$ | 13. $(XY)Z = X(YZ)$ | Associative |
| 14. $X(Y + Z) = XY + XZ$ | 15. $X + YZ = (X + Y)(X + Z)$ | Distributive |
| 16. $\overline{X + Y} = \overline{X}\,\overline{Y}$ | 17. $\overline{XY} = \overline{X} + \overline{Y}$ | DeMorgan's law |

AND

| $X$ | $Y$ | $Z = X \cdot Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR

| $X$ | $Y$ | $X + Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# Basic identities of Boolean algebra

Any expression can replace the variable X in all Boolean identities

e.g. $X + 1 = 1$ then with $X = AB + C$

$AB + C + 1 = 1$

Identity 10 -14 are similar to ordinary algebra. However, Identity 15 does not hold in ordinary algebra. Identity 16 and 17 are the very important DeMorgan's rules

| $X$ | $Y$ | $\overline{X} + \overline{Y}$ | $\overline{XY}$ | $\overline{X + Y}$ | $\overline{X}\,\overline{Y}$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | | 1 | |
| 1 | 0 | 1 | | 0 | |
| 0 | 1 | 1 | | 0 | |
| 1 | 1 | 0 | | 0 | |

# Basic identities of boolean algebra

DeMorgan's law can be extended to three or more variables. The general DeMorgan's theorem can be expressed as

$$\overline{X_1 + X_2 + \cdots + X_n} = \overline{X_1}\;\overline{X_2} \cdots \overline{X_n}$$

$$\overline{X_1 X_2 \ldots X_n} = \overline{X_1} + \overline{X_2} + \cdots + \overline{X_n}$$

# Duality

The dual of an algebraic expression is obtained by interchanging + and · and interchanging 0's and 1's

Any Boolean theorem that can be proven is thus also proven for its dual!

Previous identities appear in dual pairs

Example:

$$XY + \bar{X}Z + YZ = XY + \bar{X}Z$$

Dual $(X + Y)(\bar{X} + Z)(Y + Z) = (X + Y)(\bar{X} + Z)$

# Problem

Use a truth table to prove that the dual of $XY + \bar{X}Z + YZ$ is $(X + Y)(\bar{X} + Z)(Y + Z)$

# Useful theorems of Boolean algebra

| | | |
|---|---|---|
| $X + XY = X$ | $X(X + Y) = X$ | Absorption |
| $XY + X\overline{Y} = X$ | $(X + Y)(\overline{X} + Y) = X$ | Minimization |
| $X + \overline{X}Y = X + Y$ | $X(\overline{X} + Y) = XY$ | Simplification |
| $XY + \overline{X}Z + YZ = XY + \overline{X}Z$ | $(X + Y)(\overline{X} + Z)(Y + Z) = (X + Y)(\overline{X} + Z)$ | Consensus |

# Proof of absorption theorem

| | |
|---|---|
| $X+XY = X$ | $X(X+Y) = X$ |

# Proof of minimization theorem

| | |
|---|---|
| $XY + X\overline{Y} = X$ | $(X + Y)(\overline{X} + Y) = X$ |

# Proof of simplification theorem

| | |
|---|---|
| $X + \overline{X}Y = X + Y$ | $X(\overline{X} + Y) = XY$ |

# Proof of consensus theorem

| | |
|---|---|
| $XY + \bar{X}Z + YZ = XY + \bar{X}Z$ | $(X+Y)(\bar{X}+Z)(Y+Z) = (X+Y)(\bar{X}+Z)$ |

$$XY + \bar{X}Z + YZ = XY + \bar{X}Z + YZ \cdot 1 \qquad (2)$$

$$= XY + \bar{X}Z + YZ(X + \bar{X}) \qquad (7)$$

$$= XY + \bar{X}Z + XYZ + \bar{X}YZ \qquad (14)$$

$$= XY + XYZ + \bar{X}Z + \bar{X}YZ \qquad (10)$$

$$= XY(1 + Z) + \bar{X}Z(1 + Y) \qquad (14)$$

$$= XY + \bar{X}Z \qquad (3)$$

# Expression simplification

A *literal* is a complemented or uncomplemented variable in a term

Simplify the following expression to contain the smallest number of literals:

$$F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$$

# Expression Simplification

Simplify the following expression to contain the smallest number of literals:

$$AB + ACD + \bar{A}BD + AC\overline{D} + \bar{A}BCD$$

# Problem

Simplify the following expression to use the minimum number of literals

$$\bar{X}\bar{Y} + XYZ + \bar{X}Y$$

# Problem

Show that

$$\bar{X}\bar{Z} + YZ + X\bar{Y} = \bar{X}Y + XZ + \bar{Y}\bar{Z}$$

# Complement of a function

The complement of a function $F$, $\bar{F}$,can be obtained in two ways:

- By applying DeMorgan's theorem
- By taking the dual of the function and complementing each literal

Example: $F = \bar{X}Y\bar{Z} + \bar{X}\bar{Y}Z$

# Problem

Find the complement of the function below using both DeMorgan's theorem and the dual of the function

$$D = AC + \bar{B}C$$

# Problem

Find the complement of the function below using both DeMorgan's theorem and the dual of the function

$$D = AC + \bar{B}C + \bar{A}(\bar{B} + BC)$$

# Expression Simplification

Simplify the following expression to contain the smallest number of literals:

$$AB + ACD + \bar{A}BD + AC\bar{D} + \bar{A}BCD$$

$$= AB + \overline{A}BCD + ACD + AC\overline{D} + \overline{A}BD$$

$$= AB + \overline{A}BCD + AC(D + \overline{D}) + \overline{A}BD$$

$$= AB + AC + \overline{A}BD = B(A + \overline{A}D) + AC$$

$$= B(A + D) + AC$$

Only 5 literals!