Logic and Computer Design Fundamentals

Chapter 1 – Digital Systems and Information

Ming Cai

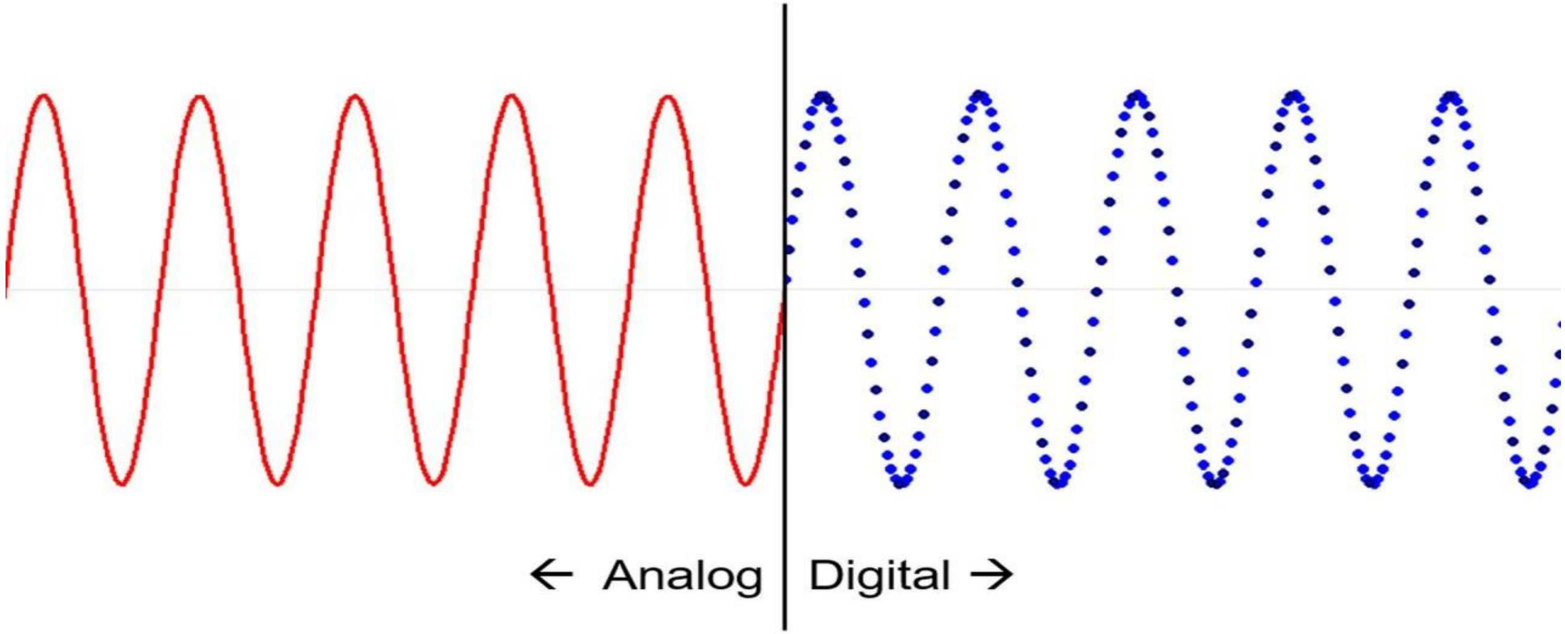
cm@zju.edu.cn

College of Computer Science and Technology
Zhejiang University

Overview

- **Digital Systems, Computers, and Beyond**
- **Information Representation**
- **Number Systems** [binary, octal and hexadecimal]
- **Arithmetic Operations**
- **Base Conversion**
- **Decimal Codes** [BCD (binary coded decimal)]
- **Alphanumeric Codes**
- **Parity Bit**
- **Gray Codes**

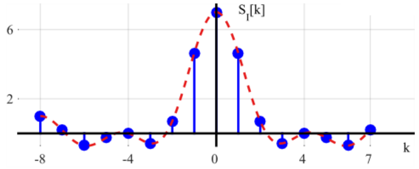
Analog and Digital Signals



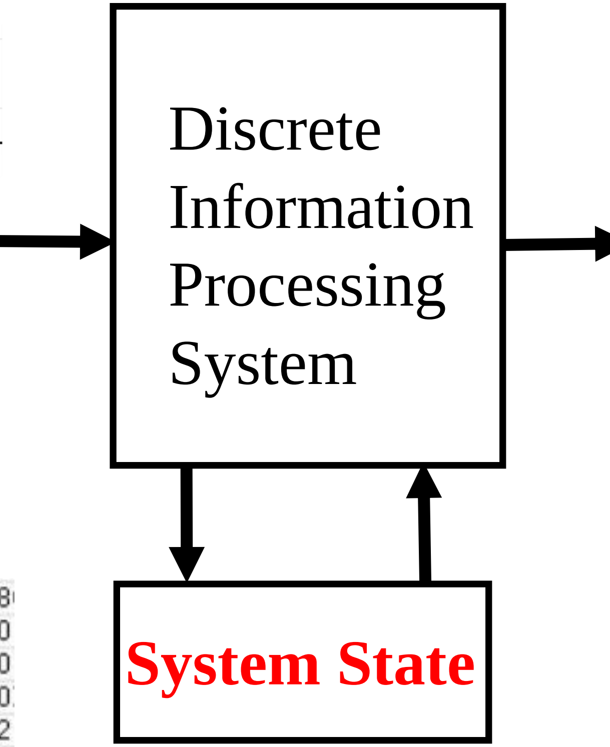
- Analog and digital signals are two types of signals carrying information.
- Analog signals are continuous in both values and time, while digital signals are discrete in value and time.

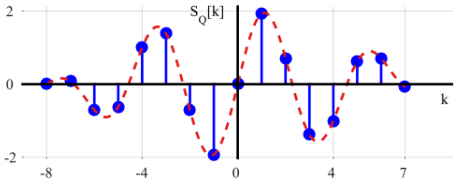
Digital System (1/2)

- **Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.**



Discrete Inputs



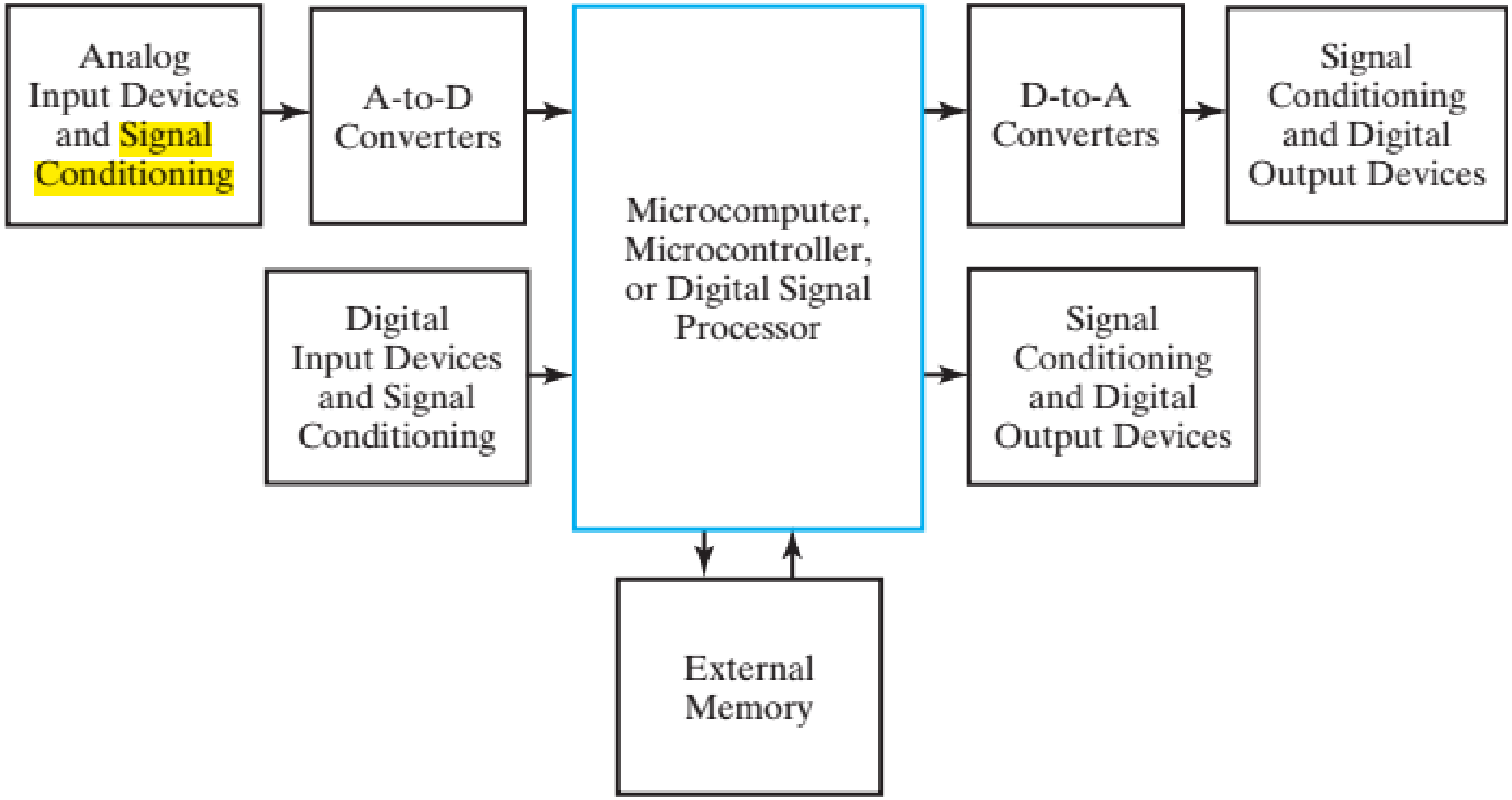


Discrete Outputs

```
US_MR    = 0x000008
US_IER   = 0x000000
US_IDR   = 0x000000
US_IMR   = 0x000000
US_CSR   = 0x000002
```

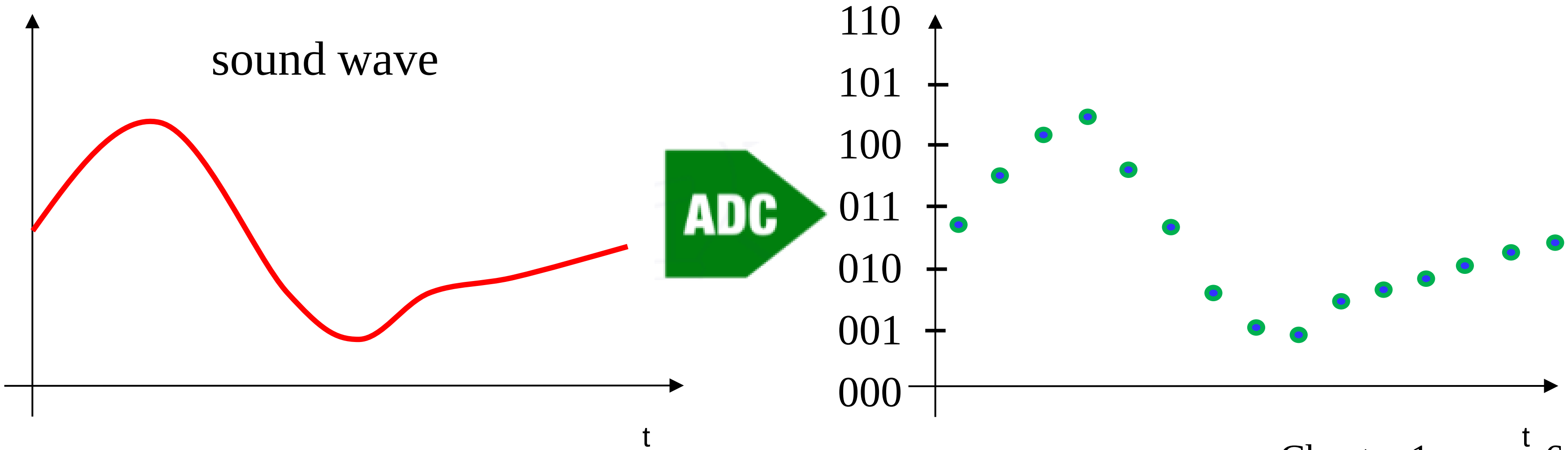
Digital System (2/2)

- Block Diagram of Digital Systems


Analog Input Devices and Signal Conditioning
A-to-D Converters
Microcomputer, Microcontroller, or Digital Signal Processor
D-to-A Converters
Signal Conditioning and Digital Output Devices
Digital Input Devices and Signal Conditioning
Signal Conditioning and Digital Output Devices
External Memory

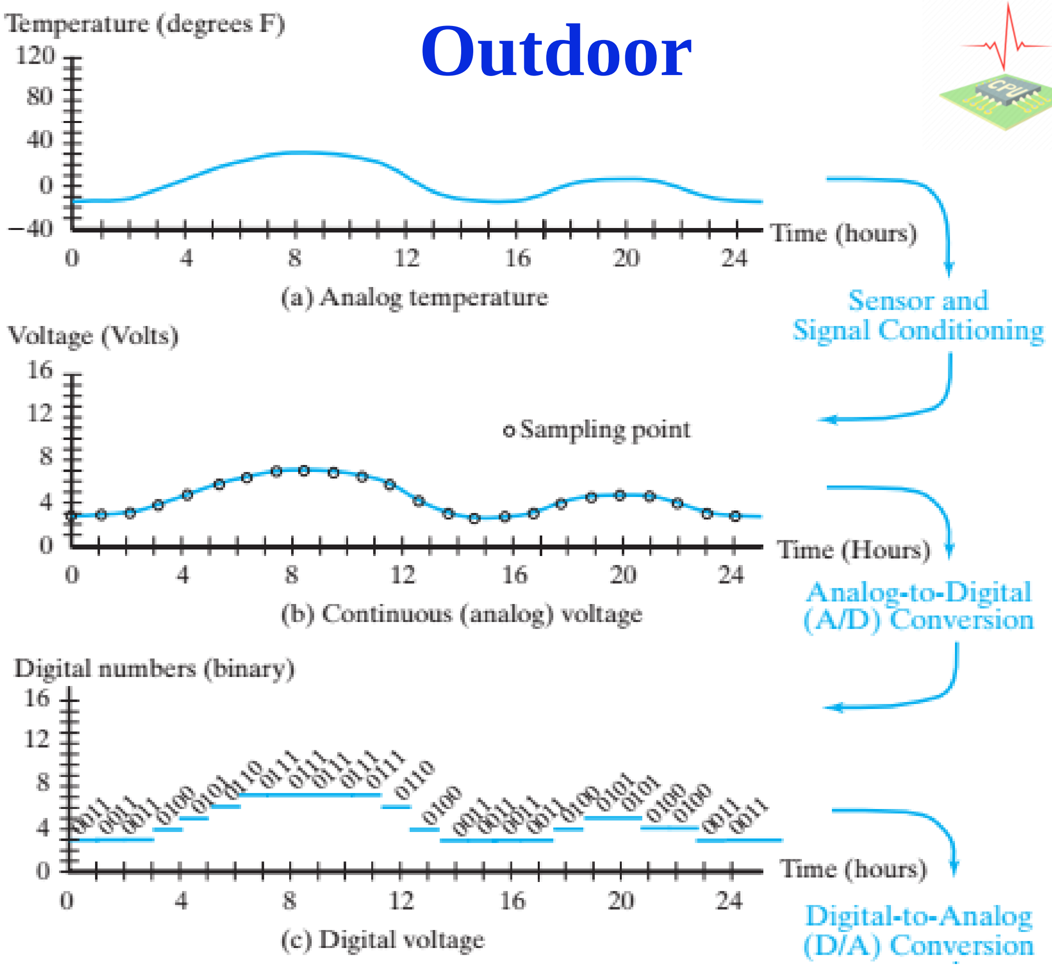
Analog-To-Digital (ADC) Converters

- ADC converts a signal from analog (continuous) to digital (discrete) form. It provides a link between the analog world of transducers and the digital world of signal processing.
- The result of ADC comprises a string of bytes, e.g., 010, 110,100, 001.



Example: Temperature Measurement and Display

Outdoor



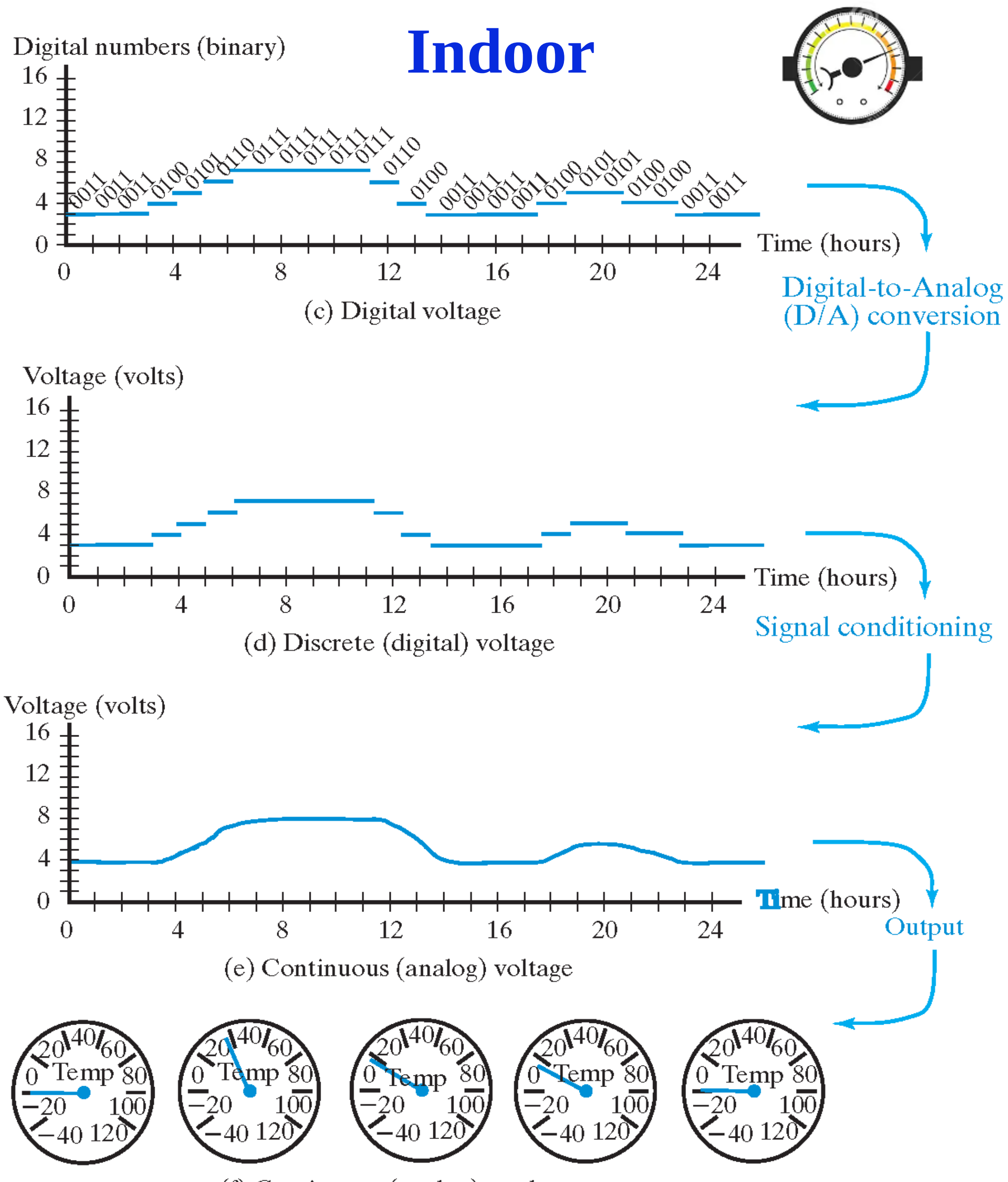
OUTDOOR DATA
328 ft/ 100m wireless transmitter in open area.
Transmitter is waterproof.

- Temperature Measurement

Example: Temperature Measurement and Display

- Temperature Display

Indoor



(c) Digital voltage

Digital-to-Analog (D/A) conversion

(d) Discrete (digital) voltage

Signal conditioning

(e) Continuous (analog) voltage

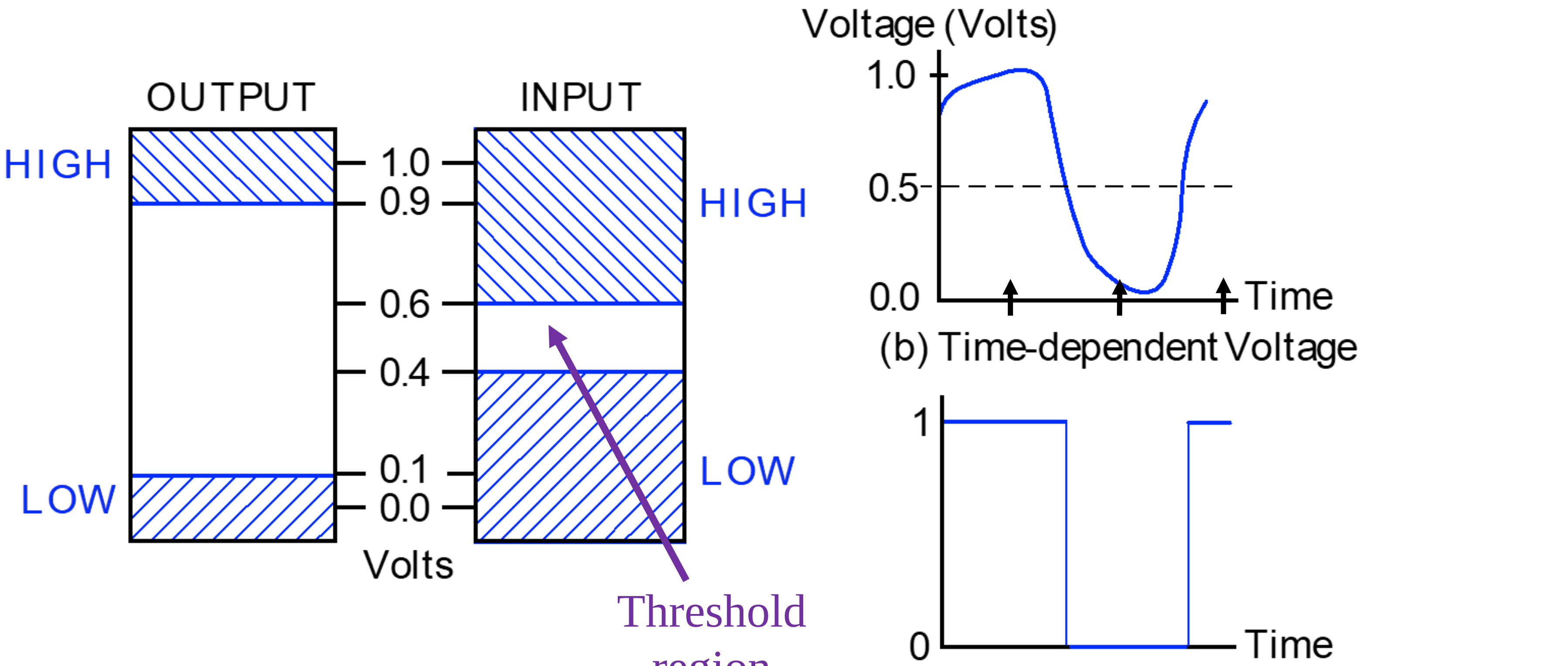
Output

(f) Continuous (analog) readout

INFORMATION REPRESENTATION - Signals

- Information variables represented by physical quantities.
- For digital systems, the variables take on discrete values.
- Two level, or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.
- Binary values are represented by values or ranges of values of physical quantities

1 Bit Signal Example – Physical Quantity: Voltage



Threshold region

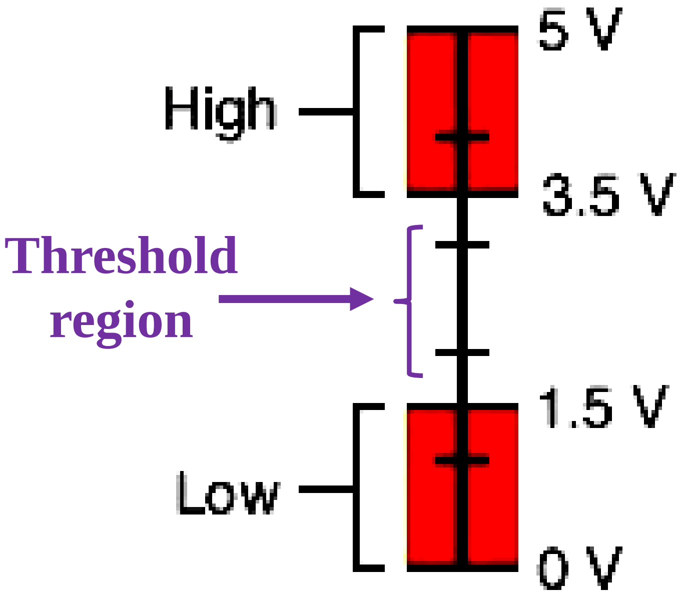
(a) Example voltage ranges

(b) Time-dependent Voltage

(c) Binary model of time-dependent voltage

- **What if falls within threshold region?**
- **Why different voltage tolerance?**
- **Why binary is most commonly used?**

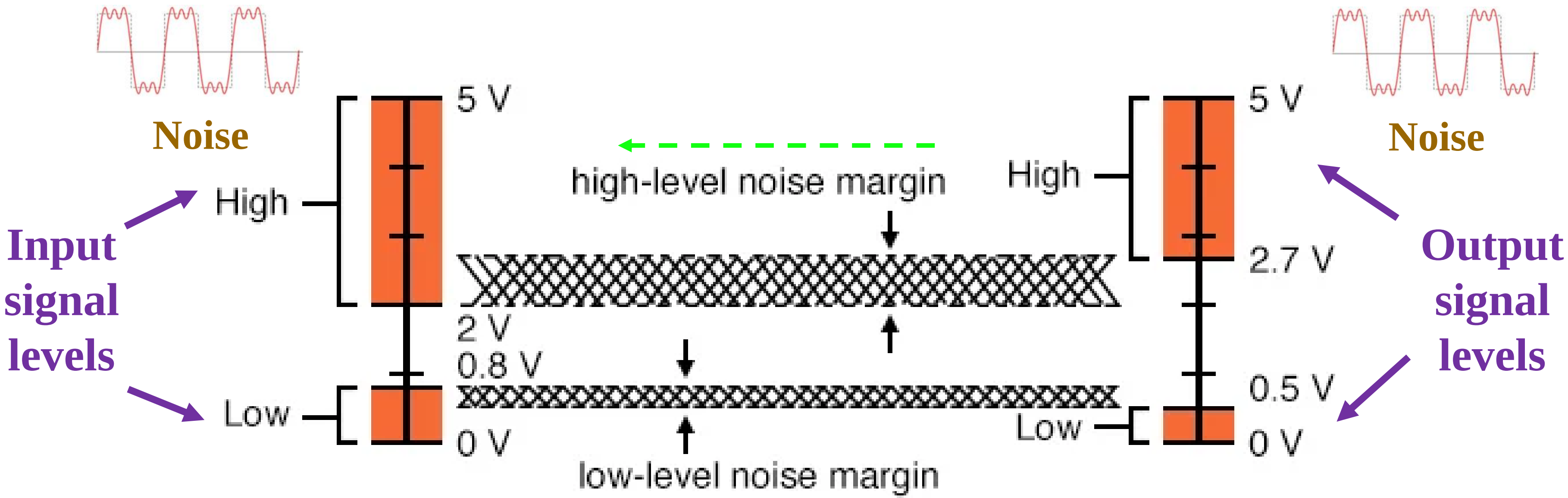
What if Falls within Threshold Region



- Voltages inside of the threshold region are undefined and result in an invalid state, often referred to as floating.

- If an output pin is "floating" in this range, there is no certainty with what the signal will result in, which may bounce arbitrarily between HIGH and LOW.

Why Different Voltage Tolerance



- The tolerable ranges for input signal levels are wider than for output signal levels, to ensure that the circuits to function correctly in spite of variations and undesirable “noise” voltages.
- The difference between the tolerable output and input ranges is called the noise margin.

Why Binary Numbers are Used

- **Two-state nature of electronic components:**
  - **A switch is either OFF (“0”) or ON (“1”).**
  - **A transistor is either conducting (“1”) or not conducting (“0”).**
- **Least amount of necessary circuitry, which results in the least amount of space, energy consumption, and cost.**

For example, to express a number less than 100

- **for decimal: 00 - 99, using 20 states**
- **for binary: 0000000 - 1111111, using 14 states**

NUMBER SYSTEMS – Representation

- Positive radix, positional number systems
- A number with *radix* ***r*** is represented by a string of digits:

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

  in which $\mathbf{0} \leq A_i < r$ and **.** is the *radix point*.
- The string of digits represents the power series:

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

(Integer Portion) + (Fraction Portion)

Number Systems – Examples

		General	Decimal	Binary
Radix (Base)		r	10	2
Digits		0 => r - 1	0 => 9	0 => 1
Powers of Radix	0	r^0	1	1
	1	r^1	10	2
	2	r^2	100	4
	3	r^3	1000	8
	4	r^4	10,000	16
	5	r^5	100,000	32
	-1	r^{-1}	0.1	0.5
	-2	r^{-2}	0.01	0.25
	-3	r^{-3}	0.001	0.125
	-4	r^{-4}	0.0001	0.0625
	-5	r^{-5}	0.00001	0.03125

Special Powers of 2

- 2^{10} (1024) 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"

ARITHMETIC OPERATIONS - Binary Arithmetic

- **Single Bit Addition with Carry**
- **Multiple Bit Addition**
- **Single Bit Subtraction with Borrow**
- **Multiple Bit Subtraction**
- **Multiplication**
- **BCD Addition**

Single Bit Binary Addition with Carry

Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):

Carry in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Multiple Bit Binary Addition

- Extending this to two multiple bit examples:

Carries	0	0
Augend	01100	10110
Addend	+10001	+10111
Sum	11101	101101

- Note: The 0 is the default Carry-In to the least significant bit.

Single Bit Binary Subtraction with Borrow

- Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):
- Borrow in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	0 0	1 1	0 1	0 0

- Borrow in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	11	1 0	0 0	1 1

Multiple Bit Binary Subtraction

- **Extending this to two multiple bit examples:**

Borrows	**0**	**0**
Minuend	**10110**	**10110**
Subtrahend	**- 10010**	**- 10011**
Difference	**00100**	**00011**

- **Notes: The 0 is a Borrow-In to the least significant bit. If the Subtrahend > the Minuend, interchange and append a – to the result.**

Binary Multiplication

The binary multiplication table is simple:

$$0 * 0 = 0 \;\mid\; 1 * 0 = 0 \;\mid\; 0 * 1 = 0 \;\mid\; 1 * 1 = 1$$

Extending multiplication to multiple digits:

$$\begin{array}{lr}
\text{Multiplicand} & 1011 \\
\text{Multiplier} & \underline{\times\ \ 101} \\
\text{Partial Products} & 1011 \\
 & 0000\ - \\
 & \underline{1011\ -\ -} \\
\text{Product} & 110111
\end{array}$$

BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

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

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 (in addition to the 10 integers) in hexadecimal represent: 10, 11, 12, 13, 14, 15

Numbers in Different Bases

- **Good idea to memorize!**

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Converting Binary to Decimal

- **To convert to decimal, use decimal arithmetic to form Σ (digit × respective power of 2).**
- **Example: Convert 11010_2 to N_{10}:**

$$
\begin{aligned}
&11010_2 \Rightarrow \\
&1 * 2^4 &= 16 \\
&+ 1 * 2^3 &= 8 \\
&+ 0 * 2^2 &= 0 \\
&+ 1 * 2^1 &= 2 \\
&+ 0 * 2^0 &= \underline{\ 0\ } \\
& & 26_{10}
\end{aligned}
$$

Converting Decimal to Binary

- **Method 1**
  - **Subtract the largest power of 2 that gives a positive remainder and record the power.**
  - **Repeat, subtracting from the prior remainder and recording the power, until the remainder is zero.**
  - **Place 1's in the positions in the binary result corresponding to the powers recorded; in all other positions place 0's.**
- **Example: Convert $(625.75)_{10}$ to N_2**

$625.75 - 2^{9} = 113.75 \Rightarrow 9$

$113.75 - 2^{6} = 49.75 \Rightarrow 6$

$49.75 - 2^{5} = 17.75 \Rightarrow 5$

$17.75 - 2^{4} = 1.75 \Rightarrow 4$

$1.75 - 2^{0} = 0.75 \Rightarrow 0$

$0.75 - 2^{-1} = 0.25 \Rightarrow -1$

$0.25 - 2^{-2} = 0 \Rightarrow -2$

Placing 1's in the result for the positions recorded and 0's elsewhere,

9	8	7	6	5	4	3	2	1	0	-1	-2
1	0	0	1	1	1	0	0	0	1	1	1

$N_2 = (1001110001.11)_2$

Conversion Between Bases

- **Method 2**
- **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

Conversion Details

- To Convert the Integral Part:

  Repeatedly divide the number by the new radix and save the remainders. The digits for the new radix are the remainders in *reverse order* of their computation. If the new radix is > 10, then convert all remainders > 10 to digits A, B, …

- To Convert the Fractional Part:

  Repeatedly multiply the fraction by the new radix and save the integer digits that result. The digits for the new radix are the integer digits in *order* of their computation. If the new radix is > 10, then convert all integers > 10 to digits A, B, …

Example: Convert 725.678_{10} To Base 2

- **Convert 725 to Base 2** **(Integral Part)**

  $725_{10} = 1011010101_2$

- **Convert 0.678 to Base 2** **(Fractional Part)**

  $0.678_{10} \approx 0.101011011001_2$

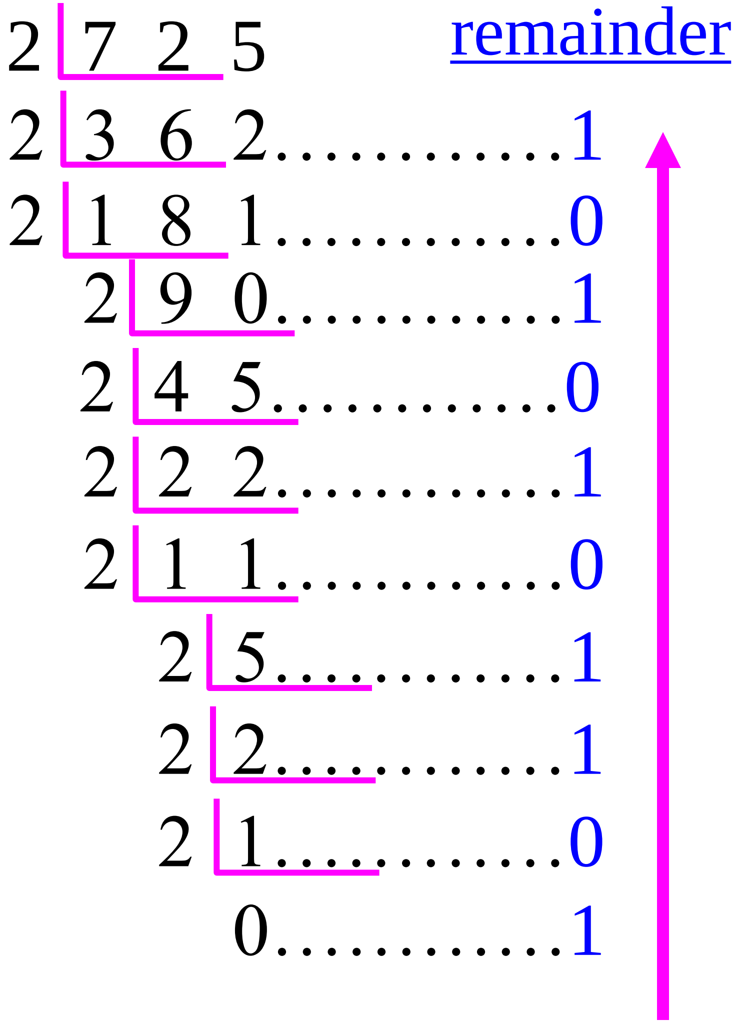
- **Join the results together with the radix point:**

  $725.678_{10} \approx 1011010101.\ 101011011001_2$

Example: Convert 725.678_{10} To Base 2

Integral Part: 725_{10}

$(725)_{10} = (10\ 1101\ 0101)_2$



2 | 7 2 5 remainder

2 | 3 6 2............1

2 | 1 8 1............0

2 | 9 0............1

2 | 4 5............0

2 | 2 2............1

2 | 1 1............0

2 | 5............1

2 | 2............1

2 | 1............0

0............1

Example: Convert 0.678_{10} To Base 2

Fractional Part:

0.678_{10}

$(0.678)_{10} \approx$

$(0.1010\ 1101\ 1001)_2$

overflow

$2\times 0.678 \ldots\ldots\ldots\ldots = 1.356$

$2\times 0.356 \ldots\ldots\ldots\ldots = 0.712$

$2\times 0.712 \ldots\ldots\ldots\ldots = 1.424$

$2\times 0.424 \ldots\ldots\ldots\ldots = 0.848$

$2\times 0.848 \ldots\ldots\ldots\ldots = 1.696$

$2\times 0.696 \ldots\ldots\ldots\ldots = 1.392$

$2\times 0.392 \ldots\ldots\ldots\ldots = 0.784$

$2\times 0.784 \ldots\ldots\ldots\ldots = 1.568$

$2\times 0.568 \ldots\ldots\ldots\ldots = 1.136$

$2\times 0.136 \ldots\ldots\ldots\ldots = 0.272$

$2\times 0.272 \ldots\ldots\ldots\ldots = 0.544$

$2\times 0.544 \ldots\ldots\ldots\ldots = 1.088$

Why Do Repeated Division and Multiplication Work?

- For example, consider the integral number 19_{10}:

$$19_{10} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$$= 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4$$

$$= \textcircled{1} + 2 \cdot (\textcircled{1} + 2 \cdot (\textcircled{0} + 2 \cdot (\textcircled{0} + 2 \cdot \textcircled{1})))$$

$19_{10} = 10011_2$ ← reverse direction

- You can see that the remainder 1 of 19 when divided by 2 is its first binary digit. Then we can perform division by 2 with remainder on (19-1)/2 to find the next digit, 1, and so on.
- The case of fractional numbers is similar:

$$0.abc\ldots_2 := a \cdot 2^{-1} + b \cdot 2^{-2} + c \cdot 2^{-3} + \cdots$$

→ forward direction

Additional Issue - Fractional Part

- Note that in repeated multiplications, the fractional part
  - can become 0 (precise value) or
  - becomes a recurring number (imprecise value)
- Example: Convert 0.1_{10} to N_2
  - 0.1 = 0.000110011001100110011 …
  - The fractional part begins repeating every 4 steps yielding repeating 0011 forever!
- Solution: Specify number of bits to right of radix point and round or truncate to this number.

Why Fractional Part of Decimal Cannot be Encoded Precisely in Binary? (1/3)

- For example:
  - The number 61 has an exact binary representation,
  - but the number 6.1 cannot be encoded precisely.
- Mathematically, there should be no intrinsic difference between the two numbers.
- In general, converting between decimal and binary depends on radix conversion.
- To convert the Integral Part:
  - $10^0 = a_1 \cdot 2^0 + a_2 \cdot 2^1 + a_3 \cdot 2^2 + \ldots\ldots$
  - $10^1 = b_1 \cdot 2^0 + b_2 \cdot 2^1 + b_3 \cdot 2^2 + \ldots\ldots$
  - $10^2 = c_1 \cdot 2^0 + c_2 \cdot 2^1 + c_3 \cdot 2^2 + \ldots\ldots$
  - ......

In this case we have to solve Diophantine equations.

Why Fractional Part of Decimal Cannot be Encoded Precisely in Binary? (2/3)

- **The Diophantine equation**
  $c = a_1 \times i_1 + a_2 \times i_2 + \ldots + a_n \times i_n$
  exists solutions if and only if $\gcd(i_1, i_2, \ldots, i_n)$ evenly divides c.
- **To convert the Integral Part:**
  - **$10^0 = a_1 \cdot 2^0 + b_1 \cdot 2^1 + c_1 \cdot 2^2 + \ldots\ldots$ has solutions (gcd=1)**
  - **$10^1 = a_2 \cdot 2^0 + b_2 \cdot 2^1 + c_2 \cdot 2^2 + \ldots\ldots$ has solutions (gcd=1)**
  - **$10^2 = a_3 \cdot 2^0 + b_3 \cdot 2^1 + c_3 \cdot 2^2 + \ldots\ldots$ has solutions (gcd=1)**
  - **……**
- **Therefore, the integral part of decimal can be encoded precisely in binary.**

Why Fractional Part of Decimal Cannot be Encoded Precisely in Binary? (3/3)

- To convert the Fractional Part:
  - $10^{-1} = a_1 \cdot 2^{-1} + b_1 \cdot 2^{-2} + c_1 \cdot 2^{-3} + \ldots\ldots$
  - $10^{-2} = a_2 \cdot 2^{-1} + b_2 \cdot 2^{-2} + c_2 \cdot 2^{-3} + \ldots\ldots$
  - $10^{-3} = a_3 \cdot 2^{-1} + b_3 \cdot 2^{-2} + c_3 \cdot 2^{-3} + \ldots\ldots$
  - ......

  have solutions ?

- For example:
  - $10^{-1} \overset{?}{=} a_1 \cdot 2^{-1} + b_1 \cdot 2^{-2} + c_1 \cdot 2^{-3} + \ldots\ldots \quad \Leftrightarrow$
  - $\frac{1}{2 \cdot 5} \neq a_1 \cdot \frac{1}{2} + b_1 \cdot \frac{1}{4} + c_1 \cdot \frac{1}{8} + \ldots\ldots \longrightarrow$

  Denominators with radix 2 do not contain factor 5. No solutions!

- Therefore, the fractional part of decimal cannot always be represented exactly in binary.

Loss of Significance——The Patriot Missile Failure

- **On 25 February 1991, a loss of significance in a Patriot missile prevented it from intercepting an incoming Iraqi Scud missile, killing 28 soldiers.**
  - The binary number of 0.1 is 0.0001100110011001100110011001100....,
  - but the 24 bit register in the Patriot stored instead 0.00011001100110011001100 introducing an error of 0.0000000000000000000000011001100... binary, or about 0.000000095 decimal,
  - resulting a time error of 0.34 seconds over 100 hours.

Octal (Hexadecimal) to Binary and Back

- **Octal (Hexadecimal) to Binary:**
  - **Restate the octal (hexadecimal) as three (four) binary digits starting at the radix point and going both ways.**
- **Binary to Octal (Hexadecimal):**
  - **Group the binary digits into three (four) bit groups starting at the radix point and going both ways, padding with zeros as needed in the fractional part.**
  - **Convert each group of three bits to an octal (hexadecimal) digit.**

Octal (Hexadecimal) to Binary and Back

- **Example:**

(67.731)$_{8}$ ＝ （110 111 . 111 011 001)$_{2}$

(312.64)$_{8}$ ＝(011 001 010 . 110 100)$_{2}$ ＝ (11001010 . 1101)$_{2}$

(11 111 101 . 010 011 11)$_{2}$＝(375.236)$_{8}$

(10 110.11)$_{2}$＝(26.6)$_{8}$

(3AB4.1)$_{16}$ ＝ （0011 1010 1011 0100 . 0001)$_{2}$

(21A.5)$_{16}$＝(0010 0001 1010 . 0101)$_{2}$

(1001101.01101)$_{2}$= (0100 1101. 0110 1000)$_{2}$=(4D.68)$_{16}$

(110 0101.101)$_{2}$＝(65.A)$_{16}$

Octal to Hexadecimal via Binary

- Convert octal to binary.
- Use groups of four bits and convert as above to hexadecimal digits.
- Example: Octal to Binary to Hexadecimal

6 3 5 . 1 7 7 $_{8}$

Restate : 110|011|101 . 001|111|111 $_{2}$

Regroup: 1|1001|1101 . 0011|1111|1000$_{2}$

Convert : 1 9 D . 3 F 8$_{16}$

- Why do these conversions work?

Binary Numbers and Binary Coding

- **Information Types**
  - **Numeric**
    - **Must represent range of data needed**
    - **Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted**
    - **Tight relation to binary numbers**
  - **Non-numeric**
    - **Greater flexibility since arithmetic operations not applied.**
    - **Not tied to binary numbers**
- **Flexibility of representation**
  - **We can assign any binary combination (called a code word) to any data as long as data is uniquely encoded.**

Non-numeric Binary Codes

- Given *n* binary digits (called 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 Number
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 minimum number of bits, *n*, satisfies the following inequality:**

  $$2^{(n-1)} < M \leq 2^n$$

  $n = \lceil \log_2 M \rceil$ **where** $\lceil x \rceil$**, called the *ceiling function*, is the integer greater than or equal to *x*.**

- **Example: How many bits are required to represent <u>decimal digits</u> with a binary code?**

- **answer: n =4, because the ceiling function for $\log_2 10$ is 4, $2^{(4-1)} < 10 \leq 2^4$**

DECIMAL CODES - Binary Codes for Decimal Digits

- There are over 8,000 ways that you can chose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

Decimal	8,4,2,1	Excess3	8,4,-2,-1	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0011
3	0011	0110	0101	0010
4	0100	0111	0100	0110
5	0101	1000	1011	0111
6	0110	1001	1010	0101
7	0111	1010	1001	0100
8	1000	1011	1000	1100
9	1001	1100	1111	1101

Binary Coded Decimal (BCD)

- **The BCD code is the 8,4,2,1 code.**
- **8, 4, 2, and 1 are weights**
- **BCD is a *weighted* code**
- **This code 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.**
- **Example: 1001 (9) = 1000 (8) + 0001 (1)**
- **How many "invalid" code words are there?**
- **What are the "invalid" code words?**
  - **Invalid code: 1010, 1011, 1100, 1101, 1110, 1111**

Excess 3 Code and 8, 4, –2, –1 Code

Decimal	Excess 3	8, 4, –2, –1
0	0011	0000
1	0100	0111
2	0101	0110
3	0110	0101
4	0111	0100
5	1000	1011
6	1001	1010
7	1010	1001
8	1011	1000
9	1100	1111

- **What interesting property is common to these two codes?** **complement code**

Number of Elements Represented

- Given n digits in radix r, there are r^n distinct elements that can be represented.
- But, you can represent M elements where M < r^n
- Examples:
  - You can represent 4 elements in radix $r = 2$ with $n = 2$ digits: (00, 01, 10, 11).
  - You can represent 4 elements in radix $r = 2$ with $n = 4$ digits: (0001, 0010, 0100, 1000).
  - This second code is called a "One-hot code".
  - One-hot encoding consists in using one bit representing each state.

Warning: Conversion or Coding?

- Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a BINARY CODE.
- $13_{10} = 1101_2$ (This is conversion)
- 13 ⇔ 0001|0011 (This is coding)

BCD Arithmetic

- **Given a BCD code, we use binary arithmetic to add the digits:**

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**

- **Note that the result is MORE THAN 9, so must be represented by two digits!**
- **To correct the digit, subtract 10 by adding 6 modulo 16.**

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**
	+0110	**so add 6**
	carry = 1 0011	**leaving 3 + cy**
	0001 | 0011	**Final answer (two digits)**

- **If the digit sum is > 9, add one to the next significant digit**

BCD Addition Example

- Add 2905_{BCD} to 1897_{BCD} showing carries and digit corrections.

$$\begin{array}{rrrrl}
1 & 1 & 1 & 0 & \\
0001 & 1000 & 1001 & 0111 & 1897_{BCD} \\
+\ \underline{0010} & \underline{1001} & \underline{0000} & \underline{0101} & 2905_{BCD} \\
0100 & 10010 & 1010 & 1100 & \\
+\ \underline{0000} & +\ \underline{0110} & +\ \underline{0110} & +\ \underline{0110} & \text{add 6} \\
0100 & 1000 & 0000 & 0010 &
\end{array}$$

$1897_{BCD} + 2905_{BCD} = 4802_{BCD}$

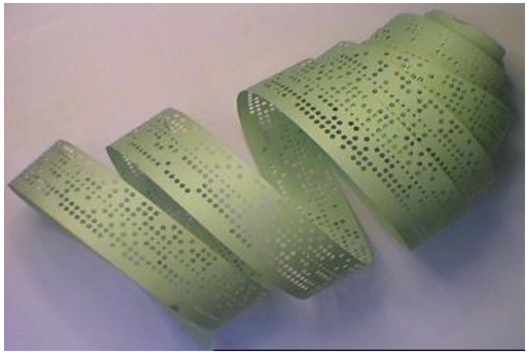
ALPHANUMERIC CODES - ASCII Character Codes

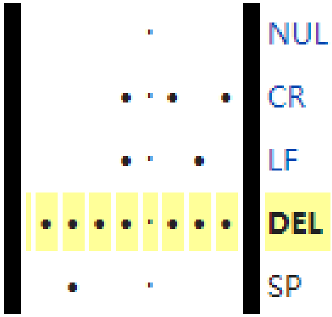
- **American Standard Code for Information Interchange** (Refer to Table 1 -5 in the text)
- This code is used to represent information sent as character-based data. It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters
- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return)
- Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).
- **The eighth and most significant bit in ASCII are used to hold parity.**

ASCII Properties

ASCII has some interesting properties:

- Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}.
- Upper case A - Z span 41_{16} to $5A_{16}$.
- Lower case a - z span 61_{16} to $7A_{16}$.
  - Lower to upper case translation (and vice versa) occurs by flipping bit 5.
- Delete (DEL) is all bits set, a carryover from when punched paper tape was used to store messages.
- Punching all holes in a row erased a mistake!

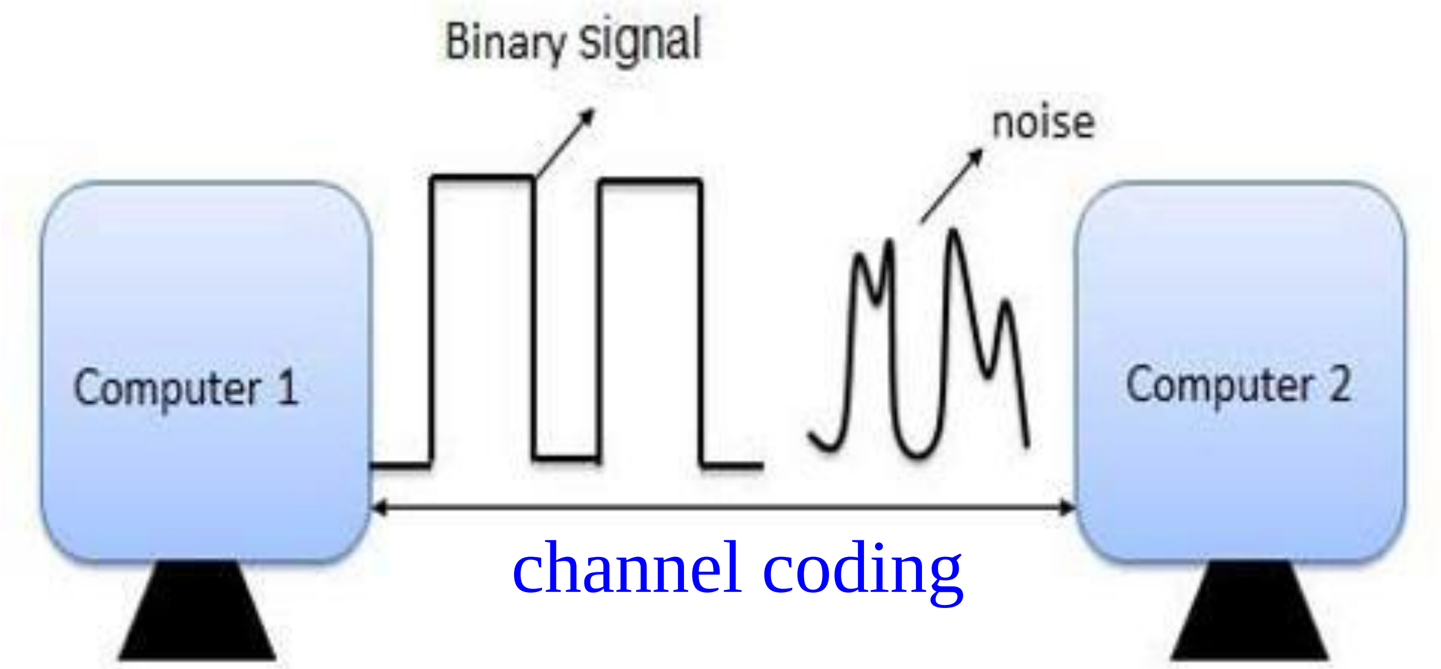




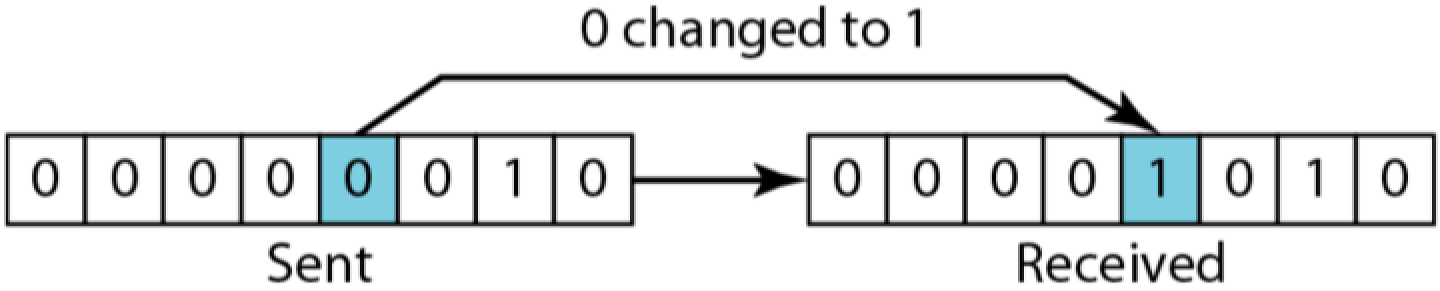
7 BIT ASCII CODE TABLE

b6b5b4 b3b2b1b0	000	001	010	011	100	101	110	111
0 0 0 0	NUL	DLE	SP	0	@	P	,	p
0 0 0 1	SOM	DC	!	1	A	Q	a	q
0 0 1 0	STX	DC	“	2	B	R	b	r
0 0 1 1	ETX	DC	#	3	C	S	c	s
0 1 0 0	EOT	DC	$	4	D	T	d	t
0 1 0 1	ENQ	NAA	%	5	E	U	e	u
0 1 1 0	ACA	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	,	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	A	[	k	
1 1 0 0	FF	FS	,	<	L	\	l	|
1 1 0 1	CR	GS	—	=	M	]	m	
1 1 1 0	SO	RS	.	>	N		n	~
1 1 1 1	SI	US	/	?	O	←	o	DEL

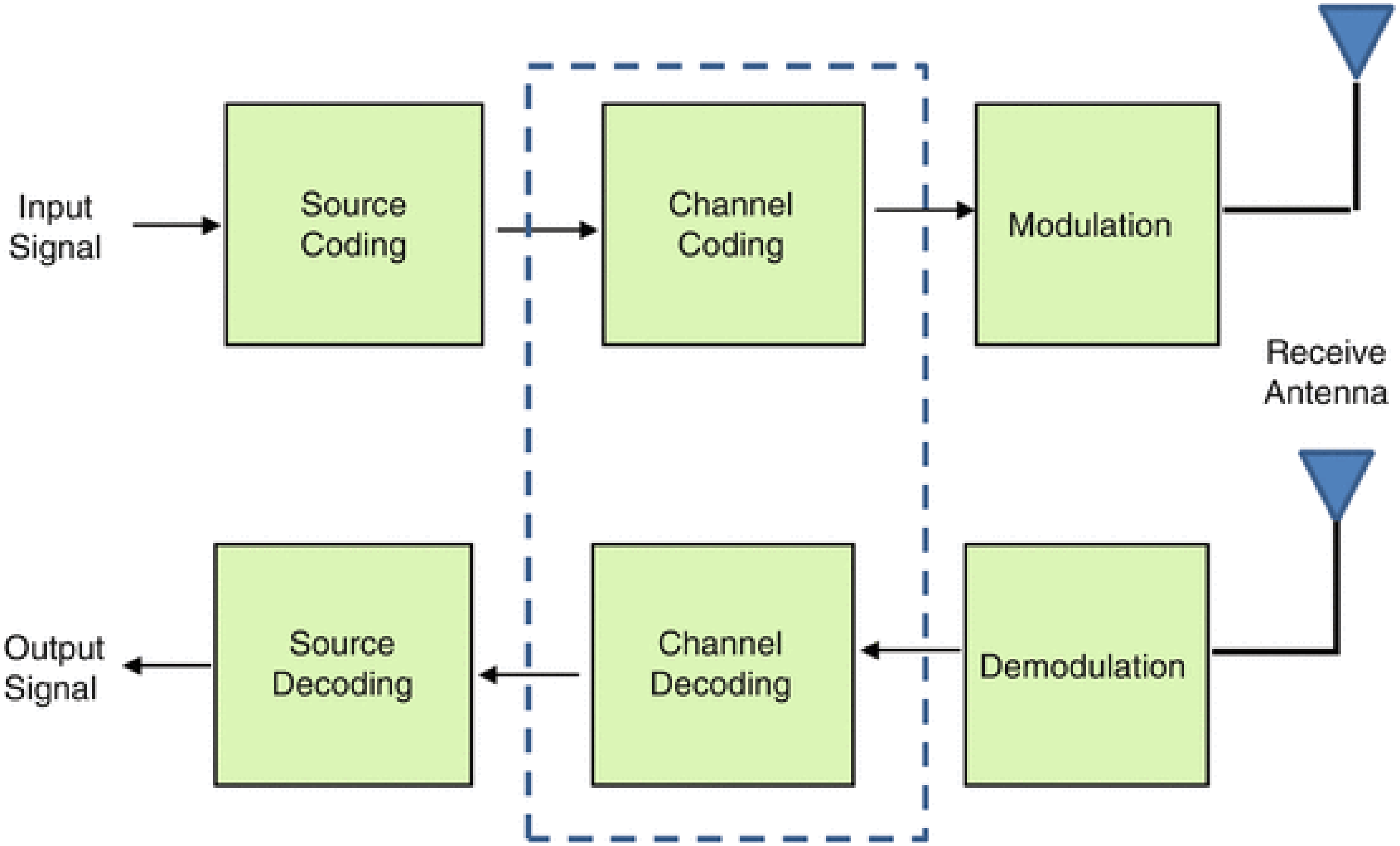
Error-Detection



channel coding



Source Coding and Channel Coding



An example of a communication system

Shannon Capacity Theorem/Limit

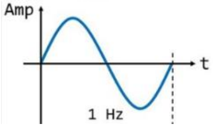
- The Shannon capacity theorem defines the maximum rate at which information can be transmitted over a communication channel of a specified bandwidth in the presence of noise.

$$C = B\log_2\left(1 + \frac{S}{N}\right)$$

signal power (Watts)

maximum data rate (bits/sec)

channel bandwidth (Hz)



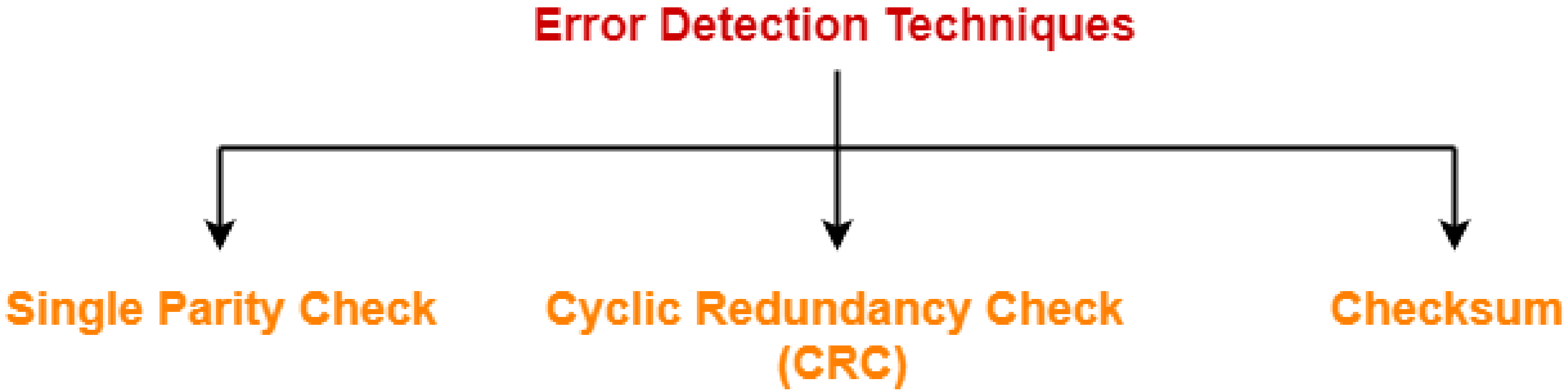
noise power (Watts)

Shannon Capacity/Limit Equation

- C is the channel capacity in bits per second.
- B is the bandwidth of the channel in hertz.
- S and N are the average received signal power and noise power over the bandwidth in watts.
- S/N is the signal-to-noise ratio (SNR).

Error-Detection Methods

- Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.



Error Detection Techniques

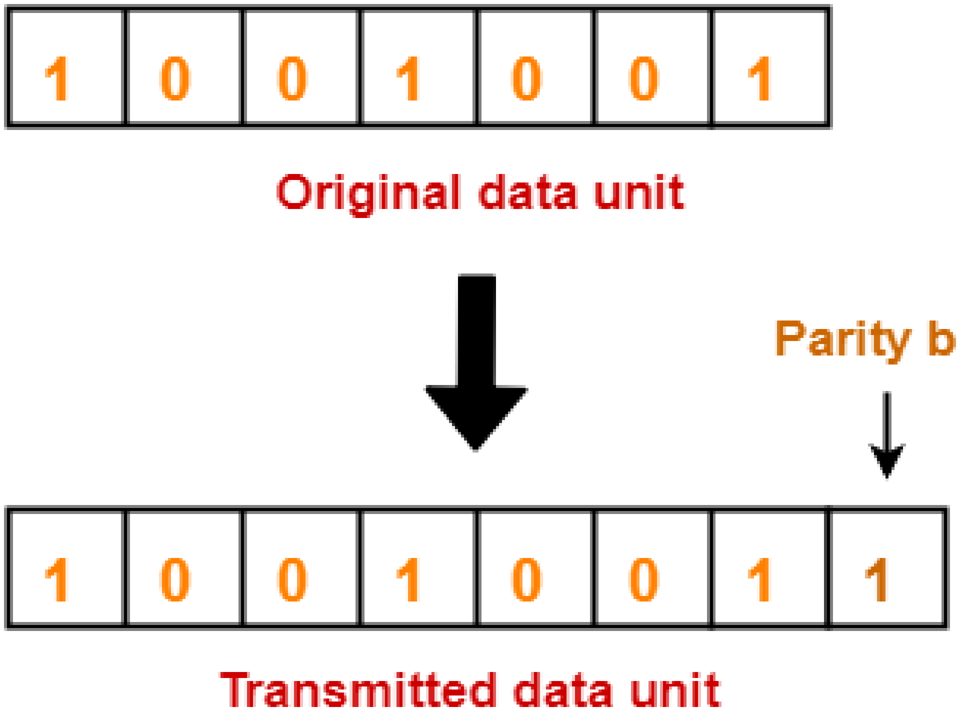
Single Parity Check

Cyclic Redundancy Check (CRC)

Checksum

- Parity is an extra bit appended onto the code word to make the number of 1's odd or even:
  - Even parity: the number of 1's in the code word is even.
  - Odd parity: the number of 1's in the code word is odd.

Parity Bit Error-Detection Codes



1 0 0 1 0 0 1

Original data unit

Parity bit

1 0 0 1 0 0 1 1

Transmitted data unit

odd or even parity?

- **Parity can detect all single-bit errors and some multiple-bit errors.**

4-Bit Parity Code Example

- Fill in the even and odd parity bits:

Even Parity	Odd Parity
Message _ Parity	Message _ Parity
000 _	000 _
001 _	001 _
010 _	010 _
011 _	011 _
100 _	100 _
101 _	101 _
110 _	110 _
111 _	111 _

- The codeword "1111" has even parity and the codeword "1110" has odd parity. Both can be used to represent 3-bit data.

GRAY CODE – Decimal

binary reflected
Gray code

Decimal	Binary	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

refection

binary addition

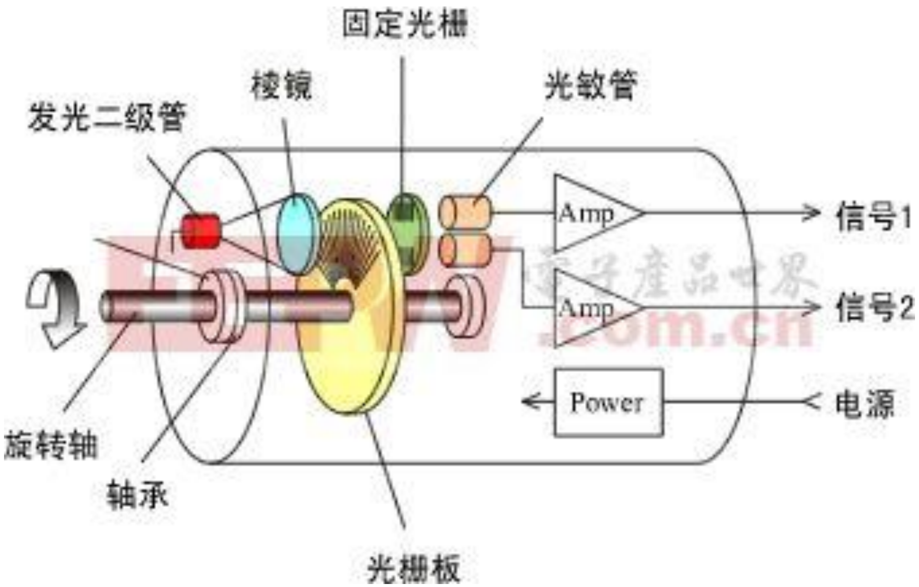
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0

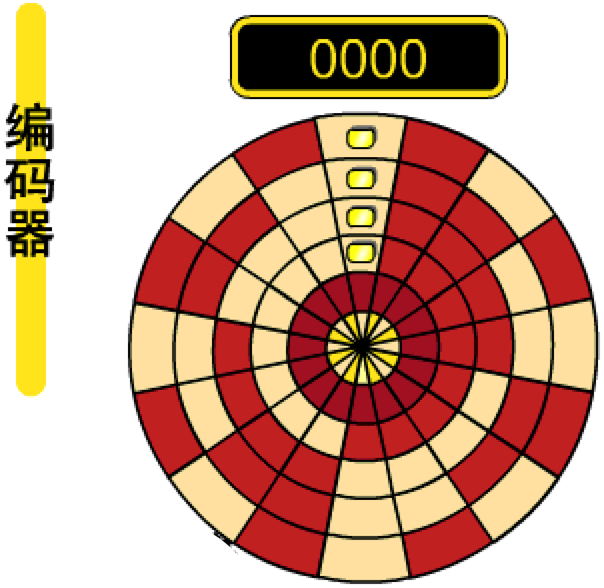
2^4	2^3	2^2	2^1	2^0	
16	8	4	2	1	
0	0	0	0	0	0 0

- **What special property does the Gray code have in relation to adjacent numbers?**
  Two successive values differ in only one bit!

Optical Shaft Encoder

- Example: rotary encoder for angle-measuring



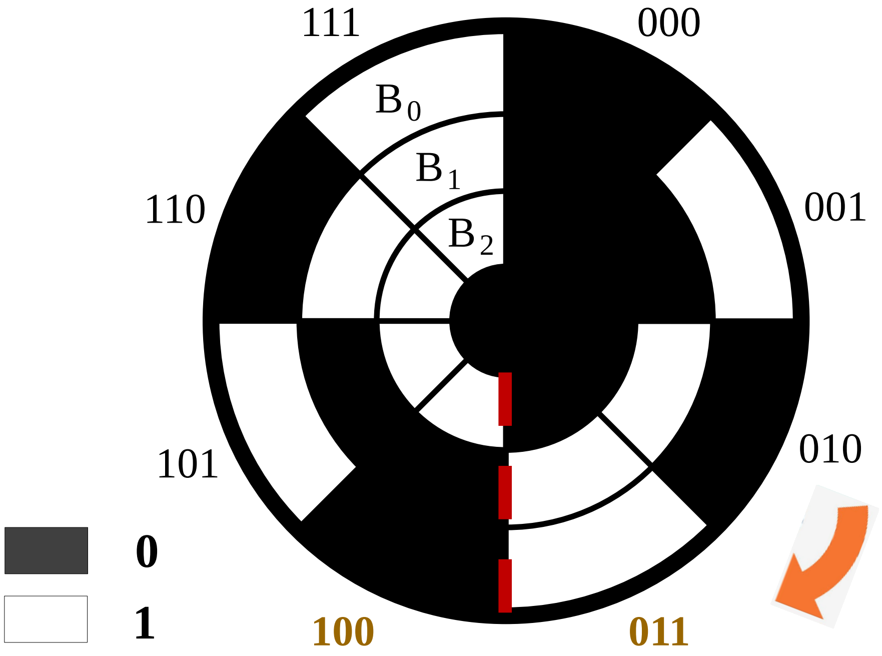


🟥 clear: 1

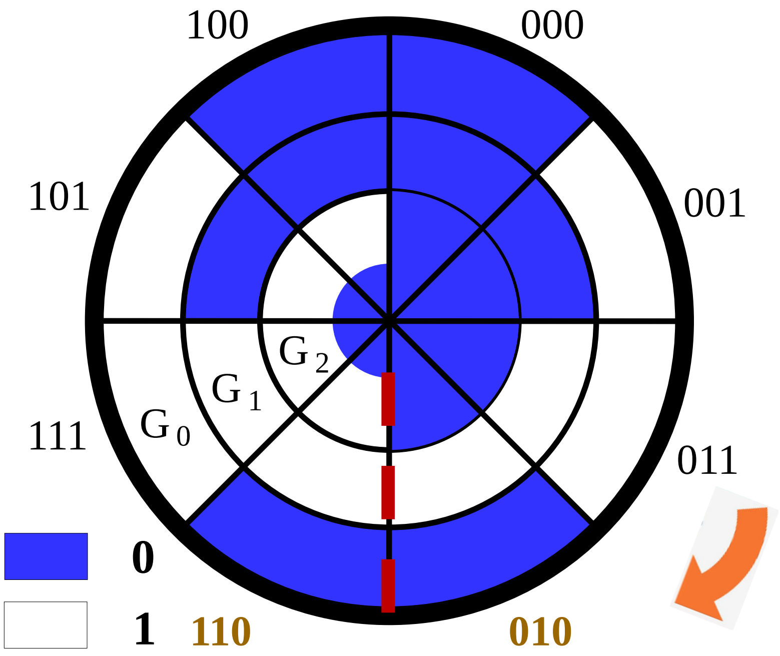
🟨 opaque: 0

Optical Shaft Encoder (continued)

- **An Example: Optical Shaft Encoder**



(a) Binary Code for Positions 0 through 7



(b) Gray Code for Positions 0 through 7

$$100 \xrightarrow[001,\ 110,\ 101]{000,\ 111,\ 011,\ 010} 011$$

This problem is called skew

Gray codes are widely used to prevent spurious output.

Shaft Encoder (Continued)

- **How does the shaft encoder work?**

The encoder disk contains opaque and clear areas.

- **For the binary code, what codes may be produced if the shaft position lies between codes for 3 and 4 (011 and 100)?**

The codes 011,100, 000, 010, 001, 110, 101, or 111 can be produced.

- **Is this a problem?**

The shaft position can be completely UNKNOWN!

Shaft Encoder (Continued)

- **For the Gray code, what codes may be produced if the shaft position lies between codes for 3 and 4 (010 and 110)?**

  Only the correct codes, either 010 or 110

- **Is this a problem?**

  No

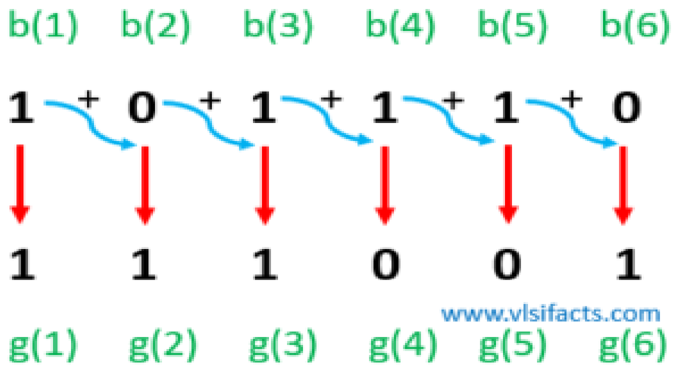
- **Does the Gray code function correctly for these borderline shaft positions for all cases encountered in octal counting?**

  Yes

Conversion from Binary Code to Gray Code

- Converting a binary code to a specific Gray code (called **binary reflected Gray code**) can be obtained by adding each adjacent pair of binary code bits to get the next Grey code bit. (Discard carry). e.g.,

- Binary code: 1 0 1 1 1 0
- Gray code: 1 1 1 0 0 1



Binary

Gray

note: binary addition: 0+0=0, 1+0=1, 0+1=1, 1+1=0

Assignment

Reading

- 1.1-1.7

Problem assignment

- 1-3, 1-9, 1-12, 1-13, 1-16, 1-18, 1-19, 1-28

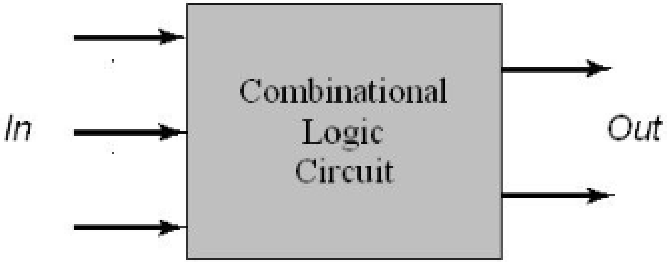
Appendix A: Types of Digital Systems

- No state present
  - Combinational Logic System
  - Output = Function(Input)
- State present
  - State = Function (State, Input)
  - State updated at discrete times

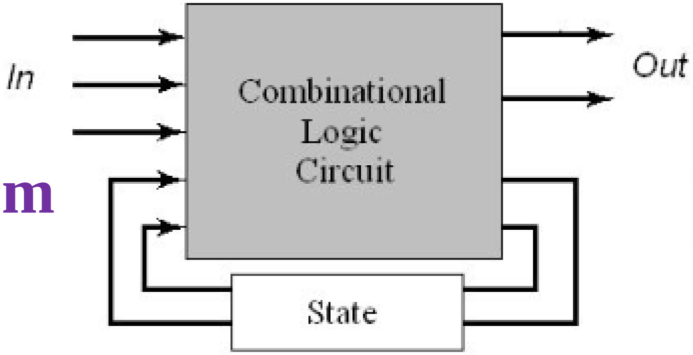
    => Synchronous Sequential System
  - State updated at any time

    =>Asynchronous Sequential System
  - Output = Function (State) or

    Output = Function (State, Input)



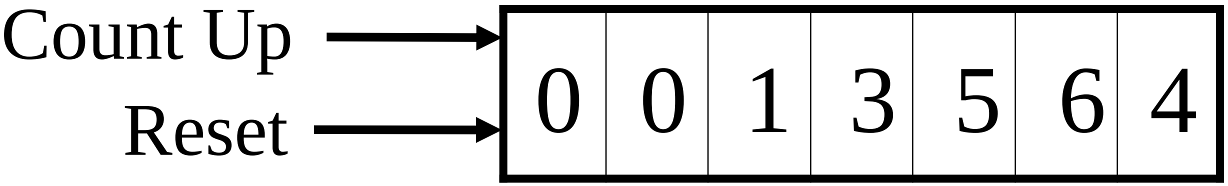
combinational system



sequential system

Digital System Example:

A Digital Counter (e. g., odometer):



Count Up

Reset

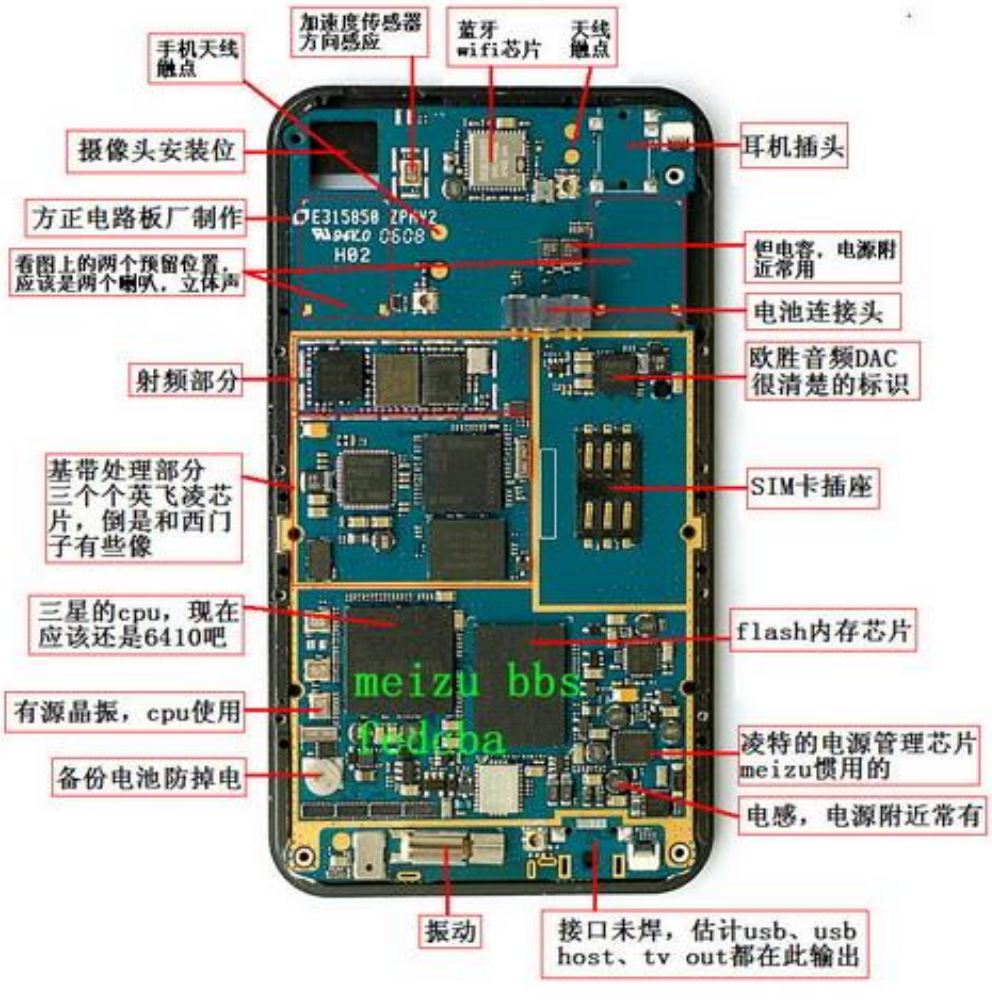
0	0	1	3	5	6	4

Inputs: **Count Up, Reset**

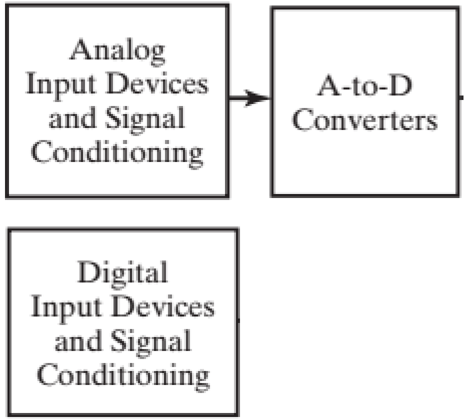
Outputs: **Visual Display**

State: **"Value" of stored digits**

Appendix B: Embedded Systems



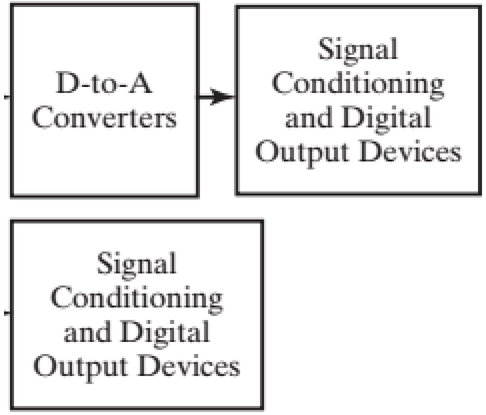
- Input


Analog Input Devices and Signal Conditioning

A-to-D Converters

Digital Input Devices and Signal Conditioning

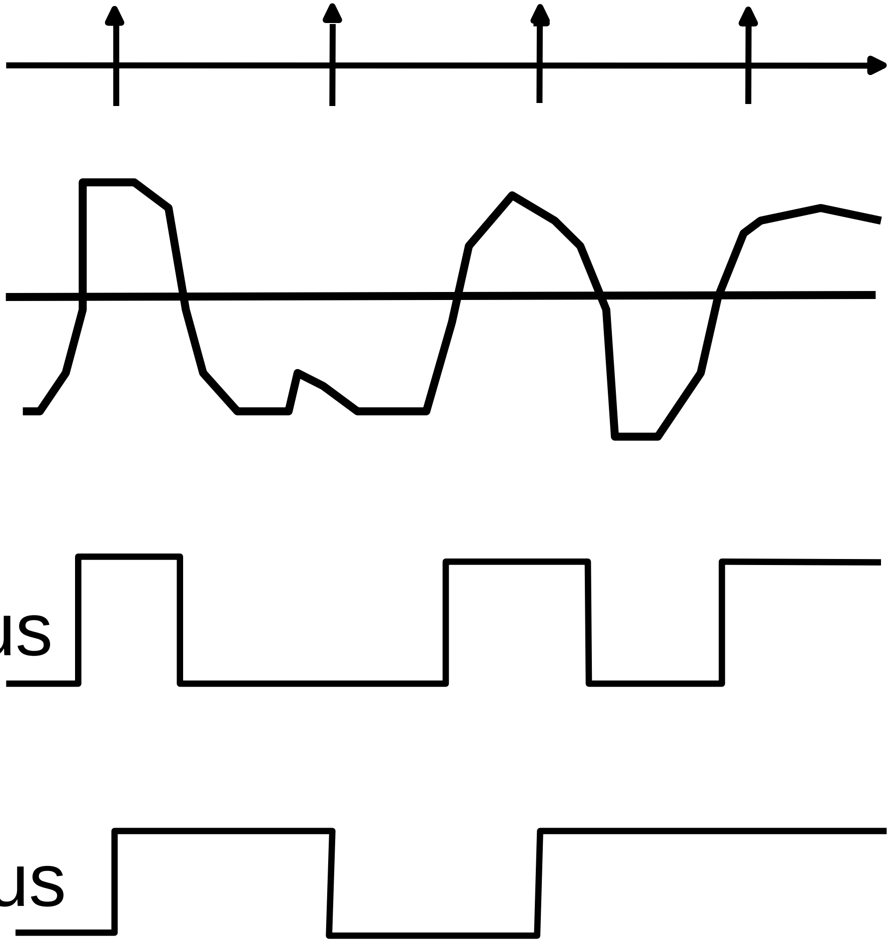
- Output


D-to-A Converters

Signal Conditioning and Digital Output Devices

Signal Conditioning and Digital Output Devices

Signal Examples Over Time



Time

Analog

Continuous in value & time

Digital

Asynchronous

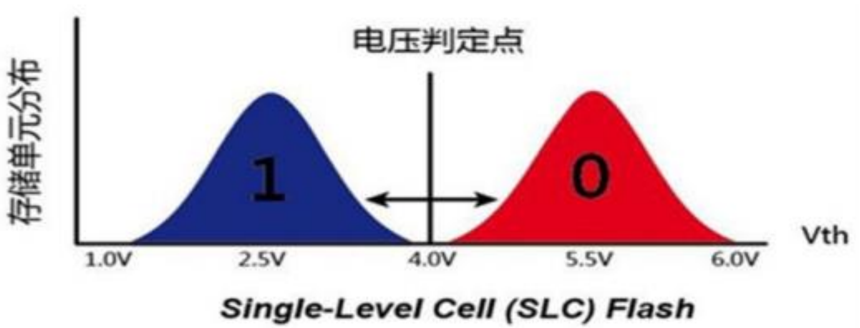
Discrete in value & continuous in time

Synchronous

Discrete in value & time

Appendix C: Binary Values: Other Physical Quantities

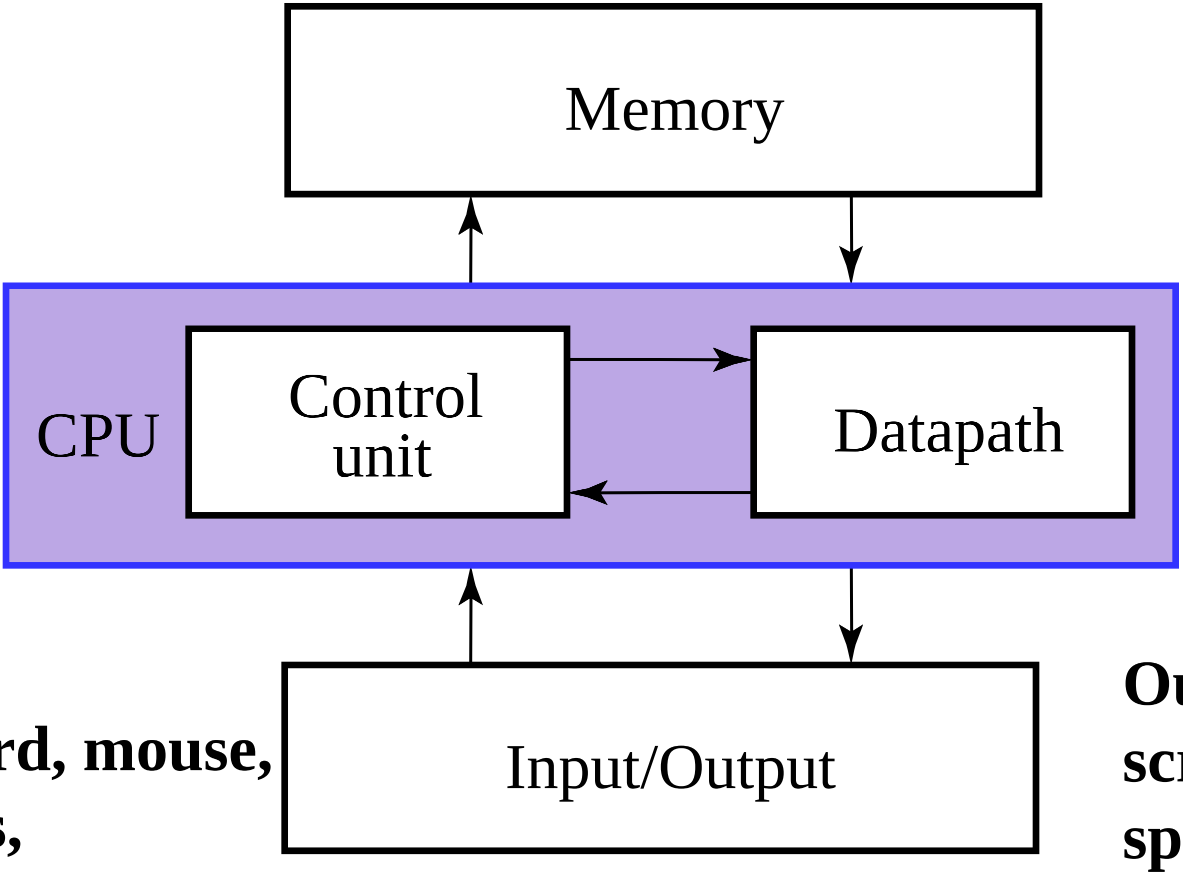
- What are other physical quantities represent 0 and 1?



Single-Level Cell (SLC) Flash

  - Flash Voltage
  - Disk Magnetic Field Direction
  - CD Surface Pits/Light
  - Dynamic RAM Capacitor Charge

Digital Computer Example



Inputs: keyboard, mouse, wireless, microphone

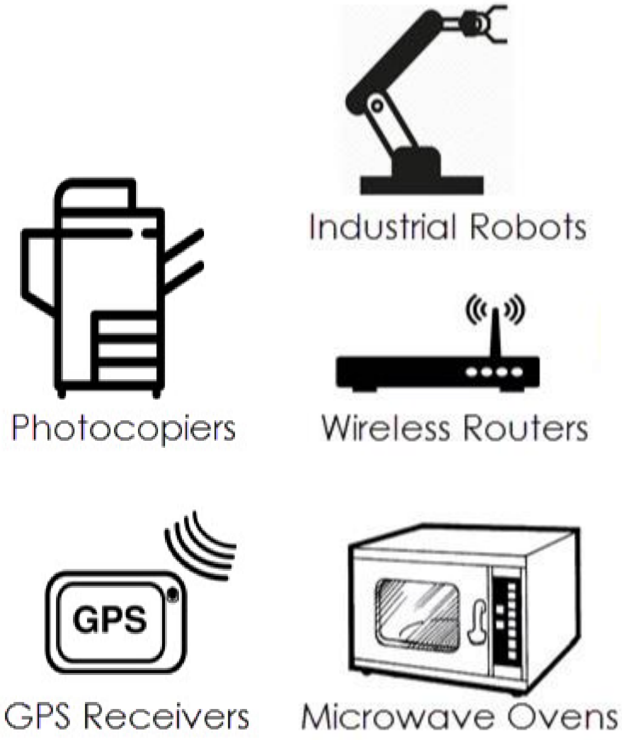
Outputs: LCD screen, wireless, speakers

And Beyond – Embedded Systems

- Computers as integral parts of other products
- Examples of embedded computers
  - Microcomputers
  - Microcontrollers
  - Digital signal processors

Embedded Systems

- Examples of Embedded Systems Applications
  - Smart phones
  - Wireless routers
  - Flat Panel TVs
  - Industrial robots
  - Video games
  - Copiers
  - Dishwashers
  - Global Positioning Systems



A Final Conversion Note

- **You can use arithmetic in other bases if you are careful:**
- **Example: Convert 101110_2 to Base 10 using binary arithmetic:**

Step 1 101110 / 1010 = 100 r 0110

Step 2 100 / 1010 = 0 r 0100

Converted Digits are 0100_2 | 0110_2

or 4 6 $_{10}$

UNICODE

- UNICODE extends ASCII to 65,536 universal characters codes
  - For encoding characters in world languages
  - Available in many modern applications
  - 2 byte (16-bit) code words
  - See Reading Supplement – Unicode on the Companion Website http://www.prenhall.com/mano