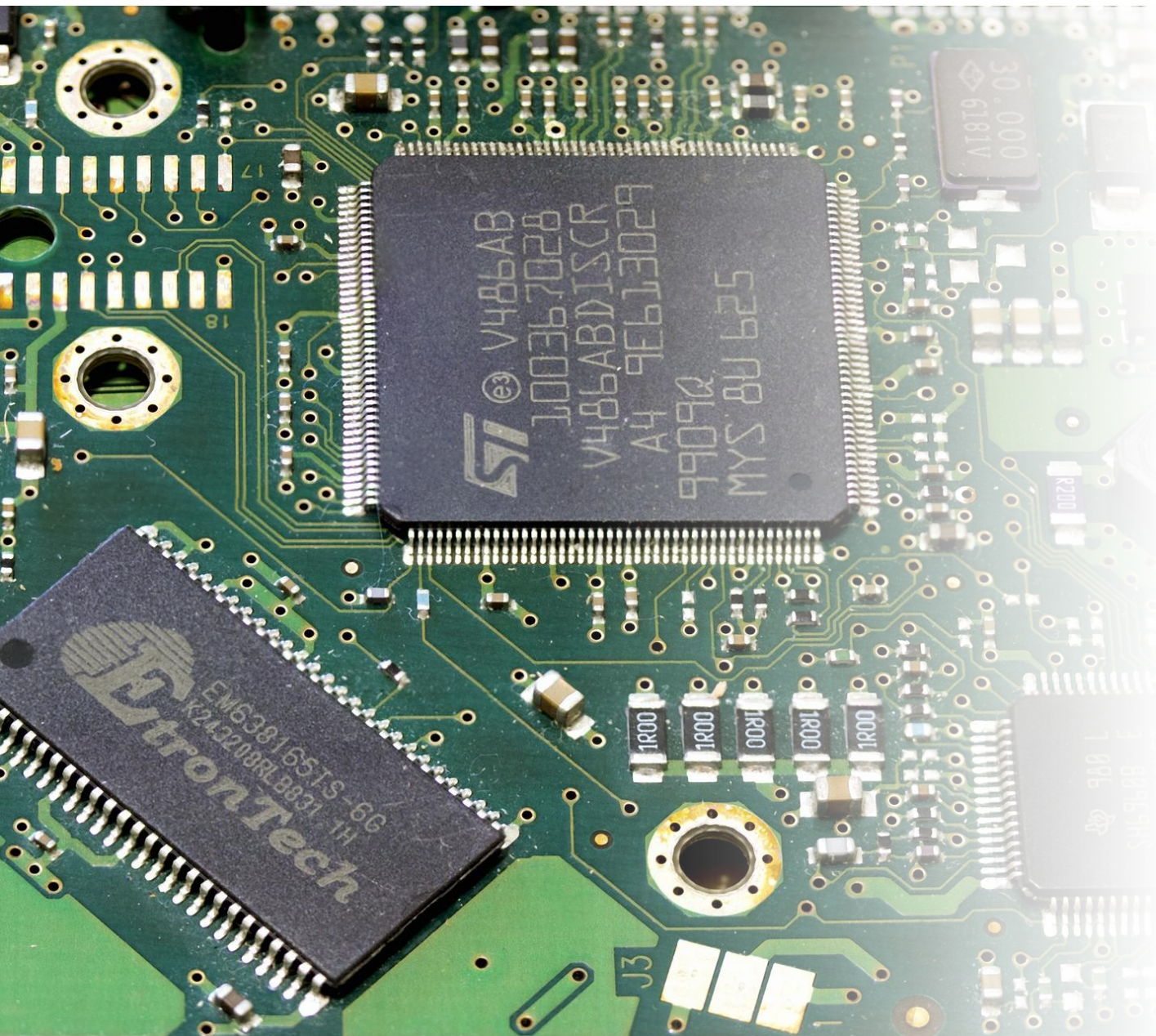




Digital Logic and System Design

2: Representation

COL215, I Semester 2025-2026

Venue: LHC 408

'E' Slot: Tue, Wed, Fri 10:00-11:00

Instructor: Preeti Ranjan Panda

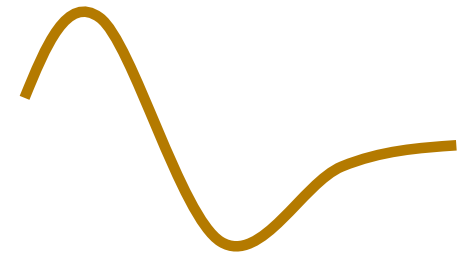
panda@cse.iitd.ac.in

www.cse.iitd.ac.in/~panda/

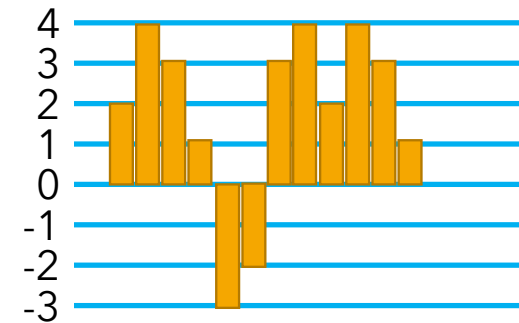
Dept. of Computer Science & Engg., IIT Delhi

A Digital System

- Represent and Manipulate **DISCRETE** Values
 - Instead of **CONTINUOUS** Values (Analog System)
- **FINITE** set of elements



Analog

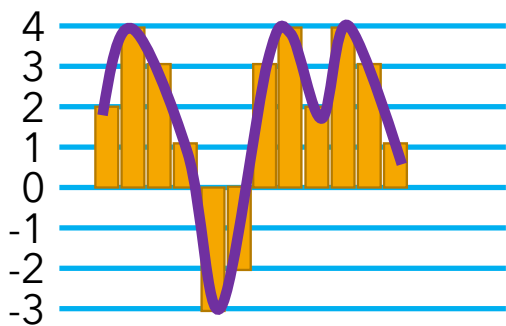


Digital

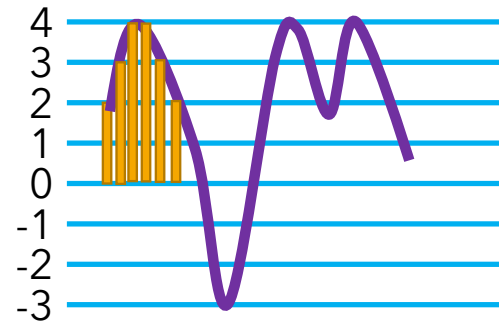
Why Digital?

- Information is lost! Why bother?
- Precise representation
- Reproducibility of results
 - E.g., fewer errors due to environmental conditions
- Ease of design
 - We'll see in this course!
- Sophisticated automation techniques
- High speed
- Low cost

Can we reduce the information loss?



Digital:
Large errors



Digital:
Smaller errors

Errors can be reduced by taking more data points

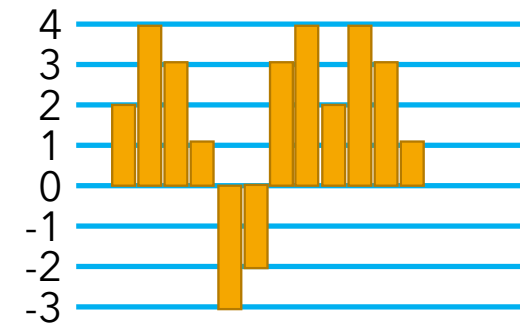
[Recall Fundamental Theorem of Integral Calculus]

Example Digital Systems

- Camera
 - Where is the digital element?
- Phone (over data connection)
 - What is digital about it?
- Computer
 - Was always digital
 - Analog components also present (interface with external world)

Representation

- Need ways to represent data
 - Store and Retrieve
 - Manipulate
- How do we represent the data on right?
- Sequence of **NUMBERS**



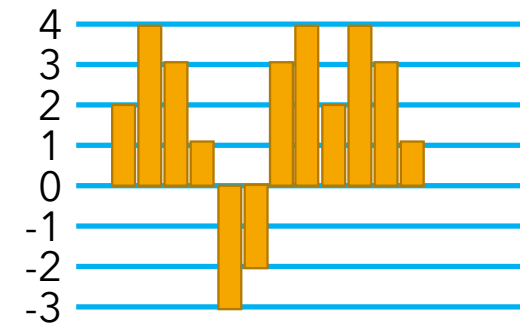
Digital

Representation:

[2, 4, 3, 1, -3, -2, 3, 4, 2, 4, 3, 1]

Representing a Number

- Can a number be represented **exactly**?
 - Integer?
 - Rational number?
 - Real number?
 - Complex number?
- Needs to be element of a **FINITE set**



Digital

Representation:

[2, 4, 3, 1, -3, -2, 3, 4, 2, 4, 3, 1]

Need to impose some restrictions

- Limited range
 - e.g., [-100, 200]
- Simple way:
 - **FIXED** number of digits
 - Each digit can take a **FIXED** number of values

Decimal Representation

- Number **3465** is a DECIMAL number
 - **base** is **10**
 - each **digit** of number $\in \{0,1,2,3,4,5,6,7,8,9\}$
- Interpretation:
$$\mathbf{3465} = \mathbf{3} \times 10^3 + \mathbf{4} \times 10^2 + \mathbf{6} \times 10^1 + \mathbf{5} \times 10^0$$

Other Bases

- We could represent the same number in a different **BASE** (also called **RADIX**)
 - E.g., Base **12**
 - in Base 12, each digit of number $\in \{0,1,2,3,4,5,6,7,8,9,10,11\}$
 - $3465_{10} = 2009_{12} = 2 \times 12^3 + 0 \times 12^2 + 0 \times 12^1 + 9 \times 12^0$
- ...or base **5**
 - in this base, each digit of number $\in \{0,1,2,3,4\}$
 - $3465_{10} = 102330_5 = 1 \times 5^5 + 0 \times 5^4 + 2 \times 5^3 + 3 \times 5^2 + 3 \times 5^1 + 0 \times 5^0$

Binary Numbers

- Binary number: **Base 2**
- n-digit number $a_{n-1}a_{n-2}a_{n-3}a_{n-4}a_{n-5}a_{n-6}a_{n-7}a_{n-8}a_{n-9}a_{n-10}a_{n-11}a_{n-12}a_{n-13}a_{n-14}a_{n-15}a_{n-16}a_{n-17}a_{n-18}a_{n-19}a_{n-20}a_{n-21}a_{n-22}a_{n-23}a_{n-24}a_{n-25}a_{n-26}a_{n-27}a_{n-28}a_{n-29}a_{n-30}a_{n-31}$
 - Digits $a_{n-1}, a_{n-2}, \dots, a_2, a_1, a_0 \in \{0, 1\}$
- Interpretation of number in base 2:
$$a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \dots + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$
- $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$$= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$$
$$= 8 + 4 + 1 = 13_{10}$$
- Thus, 1101_2 is another way to represent thirteen

Which base should we use?

- Need **reliable** way to:
 - **Store** numbers
 - **Manipulate** numbers
- Decimal system:
 - need to find a way to represent 10 different entities for each digit
- Binary system:
 - find a way to represent 2 different things
- Modern digital systems: 2 voltage levels
 - 1 V (or 1.2 V, etc.) represents '1'
 - 0 V represents '0'

13 or 15 or 1101?
Decimal Octal Binary
System System System

Choice based on engineering efficiency

- Should be easy/efficient to:
 - **Store/Retrieve** number
 - **Manipulate** numbers
- **Charge** stored on a capacitor
 - if capacitor is **charged**, a '**1**' is stored
 - if capacitor is **discharged**, a '**0**' is stored
 - Other physical phenomena could be used (e.g., magnetization direction)
- Since ANY number can be represented as a binary number, we have a way to store anything we want
- Binary is popular: **easier to distinguish between 2 values**
 - Exceptions: some memory types
 - Manipulation/computation usually in binary

Other Popular Bases

- Base **8** (Octal)
 - digits are: 0,1,2,3,4,5,6,7
- Base **16** (Hexadecimal)
 - digits are: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
 - 10=A, 11=B, 12=C, 13=D, 14=E, 15=F

Representing Real Numbers

- $3465.28 = 3 \times 10^3 + 4 \times 10^2 + 6 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 8 \times 10^{-2}$

- $1101.11_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

$$= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times 0.5 + 1 \times 0.25$$

$$= 8 + 4 + 1 + .5 + .25 = 13.75_{10}$$

Integral Part

Fractional Part

Conversion between Number Systems

- Conversion from some base to decimal system easy
 - use equation
- Conversion from decimal system to other base
 - repeated division by base
 - keep track of remainders

Example: Convert 35_{10} to Base 2

Repeated Division by Base

	Quotient	Remainder
$35 / 2 =$	17	1
$17 / 2 =$	8	1
$8 / 2 =$	4	0
$4 / 2 =$	2	0
$2 / 2 =$	1	0
$1 / 2 =$	0	1

$35_{10} \text{ to } = 100011_2$

When one base is a power of another...

- Convert **1100110011110011**₂ into Hexadecimal (Base 16)

Convert 0.6875_{10} to Binary

- Repeated Multiplication
- $$\begin{aligned}0.6875 \times 2 &= \mathbf{1}.3750 \\0.3750 \times 2 &= \mathbf{0}.7500 \\0.7500 \times 2 &= \mathbf{1}.5000 \\0.5000 \times 2 &= \mathbf{1}.0000\end{aligned}$$
- $$0.6875_{10} = 0.\mathbf{1011}_2$$

Representing Negative Numbers

5:

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

-5:

1	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

Sign-magnitude representation
Separate digit for sign

5:

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

-5:

1	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

One's complement representation
Invert representation of positive number

5:

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

-5:

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

Two's complement representation
1 + one's complement

Representing Integers in 2's Complement

- Range of integers for n-bit binary number: -2^{n-1} to $+2^{n-1}-1$
- With 3 bits we can represent: -4 to +3
- Leftmost bit represents sign
 - 0: positive
 - 1: negative

0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

$$\begin{array}{cccccc}
 0 & 0 & 0 & 1 & 1 & 0 \\
 0 & 0 & 1 & 0 & 1 & 1
 \end{array}
 \begin{array}{l}
 6 \\
 11
 \end{array}$$

$$= \begin{array}{cccccc}
 0 & 1 & 0 & 0 & 0 & 1
 \end{array}
 17$$

Multiplication

The diagram shows the addition of two 11-bit numbers using a ripple-carry adder. The first number is 6 (00000110) and the second number is 11 (00001011). The sum is 66 (0000000110). The carry is shown as a ripple from right to left.

Bit Position	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number 1 (6)	0	0	1	1	0	0	0	0	0	0	0
Number 2 (11)	0	0	0	1	0	1	1	0	0	0	0
Sum (66)	0	0	1	0	0	0	0	1	0	0	0

Binary Arithmetic: Subtraction

Subtraction: Convert to negative, then add

	0	0	0	1	1	0	6
-	0	0	1	0	1	1	11
=	0	0	0	1	1	0	6
+	1	1	0	1	0	1	-11 (2's complement)
=	1	1	1	0	1	1	-5 (2's complement)

Confirm:

0	0	0	1	0	1	5
↓ 1's complement						
1	1	1	0	1	0	
↓ + 1 2's complement						
1	1	1	0	1	1	-5

Why 2's complement?

- For negative numbers, 2's complement is popular in digital systems
- Only one encoding for zero
 - Compare: Sign magnitude **+0** and **-0**
- Arithmetic is simpler in digital implementation
 - same circuit: just **include sign digit in addition**
 - don't need additional logic for sign
- Fixed #digits: careful about **Overflow**
 - result of operation might not fit
- To get familiar, work out exercises (Textbook, Chapter 1)

Bits and Bytes

- **Bit** - a single binary digit (0 or 1)
- **Byte** - 8 bits
- 1 integer usually 32 bits (4 bytes) or 64 bits (8 bytes)
- **Boolean** values (**true/false**)
 - 1 bit is sufficient
- Other computing mechanisms: **Qubits**
 - A qubit can be in State $|0\rangle$ or $|1\rangle$
 - ...or a **superposition** of the two states: $\alpha|0\rangle + \beta|1\rangle$

Encodings: ASCII/Unicode

- Characters ('a', 'b',...) can be **encoded** into numbers
 - 1 byte in **ASCII** code (American Standard Code for Information Interchange)
 - Other codes (**Unicode** (All languages): up to 4 bytes)

Gray Code

- **Only 1 bit changing** between consecutive numbers
- Application coming up soon...

	Gray Code
0:	000
1:	001
2:	011
3:	010
4:	110
5:	111
6:	101
7:	100

Parity Code

- Add **Parity** bit to ensure **even number of 1's**
- Application: helps detect data **corruption**
 - One bit flipping
- Store/send Data + Parity bit

Data	Parity Bit
01100011	0
11001101	1
01010101	0

Conclusion

- Digital systems use **Binary Logic**
 - **2 values** (true/false, 0/1, etc.)
 - Binary operations (we will define them)
- **Voltage** value represents **Logic value**
 - Others: **magnetization** direction, **state** of material,...
- Interface with real world is often **analog**
 - Convert to digital, process, and convert back
- This is a simplification. Ongoing research:
 - Sometimes it is efficient to stay in **analog domain**
 - Different formalism needed for **Quantum Machines**

