Lecture 1: Binary Numbers

CS207: Digital Logic

Jialin Liu

Department of Computer Science and Engineering (CSE)
Southern University of Science and Technology (SUSTech)

9 September 2022

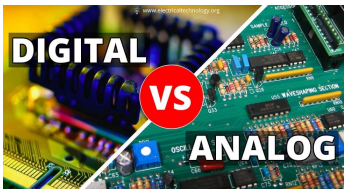
Acknowledgement

These slides were prepared based on the slides by Dr. Jianqiao Yu and the ones by Prof. Georgios Theodoropoulos of the Department of CSE at the SUSTech, as well as the contents of the following book:
M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

Electronics

- **Analog circuits** (模拟电路):
  Analog signals (usually continuous values) → analog circuits → analog systems

- **Digital circuits** (数字电路):
  Digital signal (usually discrete values) → digital circuits → digital systems
  - Signal input is given with the help of the switches (`HIGH` and `LOW`).



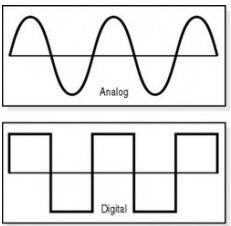


Image source: `https://www.electricaltechnology.org`

Digital Circuits

- Applications in computers, telephony, radar navigation, data processing, and other digital electronic devices.
  - Flexible in implementation.
  - Easy to manipulate.
  - Less expensive.
  - ...
- If you are interested in digital circuits in computers, you can find two videos by *Sebastian Lague* on *Sakai* (CS207-02-fall2022 -> Resource -> OtherResources):
  - *Exploring How Computers Work* (18 minutes)
  - *How Do Computers Remember* (19 minutes)

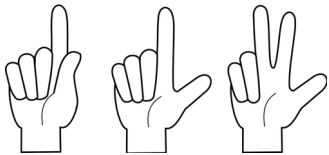
Lecture 1: Binary Numbers

Digital Systems

- All things computing have a special-purpose digital computer embedded.
- *Digital system*s represent/manipulate discrete elements of information.
  - 10 decimal digits;
  - 26 letters of alphabet.
- In a digital system, input is given with the help of switches.
  - Usually have two distinct discrete levels or values: `HIGH` and `LOW`.
- Such signals are called digital signals and the circuit within the device is called a digital circuit.
- Digital circuits find applications in computers, telephony, radar navigation, data processing, and many other applications.
  - We first learn the general properties of number systems.

Outline of This Lecture

- It is natural for human to use **decimal system** (十进制).



- In a digital world, we think in **binary**
  - A light is either off or on.
  - Signals in most present-day digital systems use binary digit: **bit** (0 and 1)

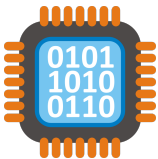


Image source: https://pixabay.com/vectors/counting-fingers-first-hand-one-154152/, https://freesvg.org/digital-signal-processor-vector-illustration

Outline

Number Systems (记数制)

- Commonly used **number systems**:
  - Decimal system (十进制): 0, 1, 2, . . . , 9 ← 10 different digits
  - Binary system (二进制): 0, 1 ← 2 different digits
  - Octal system (八进制): 0, 1, 2, . . . , 7 ← 8 different digits
  - Hexadecimal system (十六进制): 0, 1, 2, . . . , 9, A, B, . . . , F ← 16 different digits/letters

- Example:
  - A decimal number $2048 = 2 \cdot 10^3 + 0 \cdot 10^2 + 4 \cdot 10^1 + 8 \cdot 10^0$
  - A binary number 1011 in decimal system is $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1 = 11$

- It is natural for human to use **decimal system**.
- In a digital world, we think in **binary**.
- However, shorter patterns of hex characters are easier to recognise than longer patterns of 1's and 0's. → **Number-base conversions** (进制转换)

Numbers with 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

Figure: Screenshot of Table 1.2 in [1].

Outline

Number-base Conversions (进制转换)

- **Base-r system**:
  - In general, we can express any number in any **base** or **radix** (基数) "r".
  - Any number in a base-r system, having n digits to the left and m digits to the right of the **decimal point** (小数点), can be expressed as

$$a_n \cdot r^{n-1} + a_{n-1} \cdot r^{n-2} + a_{n-2} \cdot r^{n-3} + \cdots + a_2 \cdot r^1 + a_1 \cdot r^0 + b_1 r^{-1} + b_2 \cdot r^{-2} + \cdots + b_m \cdot r^{-m}$$

    for $(a_n a_{n-1} \ldots a_2 a_1 . b_1 b_2 \ldots b_m)_r$.
  - For example,

$$(4021.2)_5 = 4 \cdot 5^3 + 0 \cdot 5^2 + 2 \cdot 5^1 + 1 \cdot 5^0 + 2 \cdot 5^{-1} = (511.4)_{10}$$

Conversion between Number Systems

- Decimal numbers are the key:
  base-p to base-q $\rightarrow$ base-p to base-10 to base-q.

- Steps:
  1. Base p to base-10:

$$
\begin{aligned}
&(a_n a_{n-1} \ldots a_2 a_1 . b_1 b_2 \ldots b_m)_p \\
&= (a_n \cdot p^{n-1} + a_{n-1} \cdot p^{n-2} + \cdots + a_2 \cdot p^1 + a_1 \cdot p^0 + b_1 \cdot p^{-1} + b_2 \cdot p^{-2} + \cdots + b_m \cdot p^{-m})_{10}
\end{aligned}
$$

  2. Base-10 to base-q ?
     - We use multiplication for base-p to decimal.
     - The inverse of multiplication is division.

Conversion between Number Systems

Example

- Convert $(26)_{10}$ into a binary number (thus, base-10 to base-2).

Index	Division	Quotient	Remainder	
1	26/2	13	0	// $26 = 13 * 2^1 + 0 * 2^0$
2	13/2	6	1	// $26 = 6 * 2^2 + 1 * 2^1 + 0 * 2^0$
3	6/2	3	0	// $26 = 3 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$
4	3/2	1	1	// $26 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$
5	1/2	0	1	// $26 = 0 * 2^5 + 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$

The converted binary number is $(11010)_2$.

Conversion between Number Systems: Fraction

- For fraction, the computation is reversed again. → **Multiplication** !
- Example: convert $(25.625)_{10}$ into a binary number.

  1. First, $(25)_{10} = (11001)_2$.
  2. Then, $(0.625)_{10} = (0.101)_2$.

  Therefore, $(25.625)_{10} = (11001.101)_2$.

Division	Quotient	Remainder
$25/2$	12	1
$12/2$	6	0
$6/2$	3	0
$3/2$	1	1
$1/2$	0	1

Multiplication	Product
$0.625 \cdot 2$	1.250
$0.250 \cdot 2$	0.500
$0.500 \cdot 2$	1.000

Outline

Conversion between Binary and Octal

- The maximum digit in an octal number system is 7
  → Represented as $(111)_2$ in a binary system.
- Starting from the least significant bit (LSB, (最低有效位)), we group three digits at a time and replace them by the octal equivalent of those groups.
- **Example**: convert $(101101010)_2$ into an octal number.

Starting with LSB and grouping 3 bits	101	101	010
Octal equivalent	5	5	2

The octal number is $(552)_8$.

Conversion between Binary and Hexadecimal

Homework

Read page 9 of [1] to learn about **conversion between binary and hexadecimal** by yourself !

[1] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

Outline of This Lecture

Binary Addition, Subtraction and Multiplication

Examples

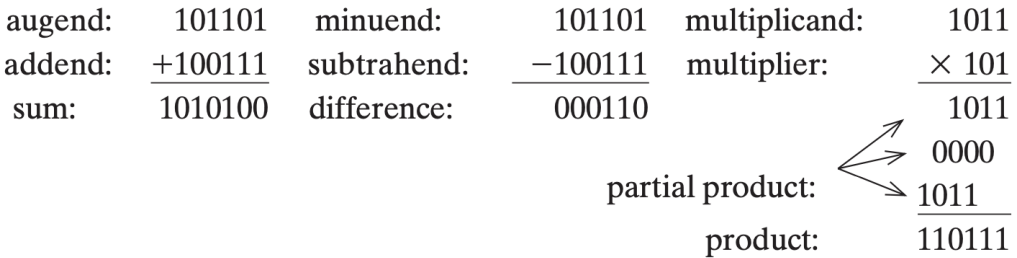


Figure: Screenshot of an example in [1].

Complements for Substraction

- When human do subtraction, we use "borrow" concept to borrow a 1 from a higher significant position.
- It is hard for circuits to design "borrow". So **complement**s are used to implement subtraction.
  - Simplify the subtraction operation.
  - Simpler, less expensive circuits.
- Two types of complements for each base-r system:
  - **Radix complement** (补码): r's complement;
  - **Diminished radix complement** (反码): $r-1$'s complement.
  - Examples:
    - For a binary system: 2's complement and 1's complement.
    - For a decimal system: 10's complement and 9's complement.

Outline

Diminished Radix Complement: $r-1$'s Complement

- **Solution**: Use $r-1$ minus each digit.
- Examples:
  - The 9's complement of 546700 is $999999 - 546700 = 453299$.
  - The 9's complement of 12398 is $999999 - 012398 = 987601$.

Radix Complements: r's Complement

- **Solution 1**: Calculate the diminished radix complement, then plus one. E.g.,
  - The 10's complement of 546700 is $999999 - 546700 + 1 = 453299 + 1 = 453300$.
  - The 10's complement of 12398 is $999999 - 012398 + 1 = 987601 + 1 = 987602$.

- **Solution 2**: Use r minus the least significant non-zero digit, and $r-1$ minus digits on the left. E.g.,
  1. The least significant non-zero digit of 546700 is 7, therefore r minus the least significant non-zero digit is:
  $$10 - 7 = 3$$
  2. The digits on the left of 7 are 546, therefore $r-1$ minus the digits on the left is:
  $$999 - 546 = 453$$
  3. The 10's complement of 546700 is 453300.

Outline

Binary Subtraction

- Three ways:
  - The direct “borrow” method (借位法).
  - The r’s complement method.
  - The $r-1$’s complement method.
- We discuss the r’s complement method in this course.

Binary Subtraction using r's Complement Method (1/2)

Subtraction $M-N$, if $M \geq N$

- Steps:
  1. Add M to r's complement of N,
  2. then discard the end carry (进位).

- Example: $72532 - 3250$

$$
\begin{aligned}
M &= 72532 \\
N &= 3250 \\
r &= 10 \\
10\text{'s complement of } N &= 99999 - 3250 = 96750 \\
\text{Sum of } M \text{ and } r\text{'s complement of } N &= 72532 + 96750 = 169282 \\
\text{Discard the end carry} &= -100000 \\
&= 69282
\end{aligned}
$$

Binary Subtraction using r's Complement Method (2/2)

Subtraction $M - N$, if $M < N$

- Steps:
  1. Add M to r's complement of N,
  2. then take its r's complement,
  3. finally add a negative sign.

- Example: $3250 - 72532$

$$
\begin{aligned}
M &= 3250 \\
N &= 72532 \\
r &= 10 \\
10\text{'s complement of } N &= 99999 - 72532 = 27468 \\
\text{Sum of } M \text{ and } r\text{'s complement of } N &= 3250 + 27468 = 30718 \\
10\text{'s complement} &= 99999 - 30718 = 69282 \\
\text{Add a negative sign} &= -69282
\end{aligned}
$$

Outline of This Lecture

Signed Binary Numbers

- In real life one may have to face a situation where both positive and negative numbers may arise.
  - We have $+$ and $-$.
  - Digital systems represent everything with binary digits, including the "sign".
- Three types of representations of signed binary numbers:
  - Sign-magnitude representation (符号量级表示法)
  - Signed-complement system
    - Signed-1's complement representation
    - Signed-2's complement representation

Sign-magnitude Representation

- An additional bit is used as the **sign bit**, usually placed as the most significant bit (MSB), which is the bit in a multiple-bit binary number with the largest value.
- Generally,
  - a 0 is reserved for a positive number and
  - a 1 is reserved for a negative number.
- Examples:
  - An 8-bit signed binary number 01101001 represents a **positive** number whose magnitude is $(1101001)_2 = (105)_{10}$.
  - An 8-bit signed binary number 11101001 represents a **negative** number whose magnitude is $(1101001)_2 = (105)_{10}$, i.e., -105.

Signed-complement System: 1's and 2's Complement Representation

- In **signed-1's complement representation**, both numbers are a complement of each other.
  - MSB 0 for positive numbers and 1 for negative numbers.
  - Example: $(0111)_2$ represents $(+7)_{10}$ and $(1000)_2$ represents $(-7)_{10}$.

- In **signed-2's complement representation** (most common), 1 is added to 1's complement representation.
  - MSB 0 for positive numbers and 1 for negative numbers.
  - Example: $(0110)_2$ represents $(+6)_{10}$ and $(1010)_2$ represents $(-6)_{10}$.

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Figure: Four-bit Signed Binary Numbers (Table 1.3 of [1]).

Comparison

- **Sign-magnitude representation** ← Separate handling of the sign and the magnitude makes it awkward when employed in computer arithmetic.
- **Signed-complement system**
  - Signed-1's complement representation ← Useful as a logical operation.
  - Signed-2's complement representation ← Widely used for signed binary arithmetic.

Question

- **Why prefer signed-2's complement representation over the other two ?**

Answer: It allows to perform both addition and subtraction operations using a single circuit because:
"*Compared to other systems for representing signed numbers (e.g., ones' complement), two's complement has the advantage that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers (as long as the inputs are represented in the same number of bits as the output, and any overflow beyond those bits is discarded from the result). This property makes the system simpler to implement, especially for higher-precision arithmetic. Unlike ones' complement systems, two's complement has no representation for negative zero, and thus does not suffer from its associated difficulties.*"
(`https://en.wikipedia.org/wiki/Two's_complement`)

Outline of This Lecture

Binary Codes

- Computers and other digital circuits process data in binary format.
- The interpretation of the data is only possible if the **code** in which the data is being represented is known.
- Example: 1000010 represents
  - 66 (decimal) in straight binary,
  - 42 (decimal) in Binary-Coded-Decimal (BCD), and
  - letter B in American Standard Code for Information Interchange (ASCII) code.

Commonly Seen Codes

- Binary-Coded-Decimal (BCD, 8421).
- Gray code.
- American Standard Code for Information Interchange (ASCII).
- Error-detection code.

Binary-Coded-Decimal (BCD, 8421)

- Four bits are required to code each decimal number.
  - Also known as **8-4-2-1 code**, as 8, 4, 2, and 1 are the weights of the four bits of BCD.
  - Only use 0000 – 1001 for representing 0 – 9 (decimal), the other 6 are not used.
- It is convenient to use BCD for input and output in digital systems.

- Example: Give the BCD equivalent for the decimal number 69.27.

Decimal number	6	9	.	2	7
BCD code	0110	1001	.	0010	0111

Therefore, $(69.27)_{10} = (01101001.00100111)_{\text{BCD}}$.

BCD Addition

Rules

- First add the two numbers using normal rules for binary addition.
- If the 4-bit sum is equal to or less than 9, it becomes a valid BCD number.
- If the 4-bit sum is greater than 9, or if a carry (进位) out of the group is generated, it is an invalid result.
  - In such a case, add 0110_2 or 6_{10} to the 4-bit sum in order to skip the six invalid states and return the code to BCD.
  - If a carry results when 6 is added, add the carry to the next 4-bit group.
- Example: $0111_{\text{BCD}} + 1001_{\text{BCD}}$:

$$
\begin{array}{rl}
 & 0111 \\
+ & 1001 \\
\hline
 & 10000 \rightarrow \text{Invalid BCD number} \\
+ & 0110 \rightarrow \text{Add } 6 \\
\hline
0001 & \quad 0110 \rightarrow \text{Valid BCD number}
\end{array}
$$

BCD Subtraction

- Example: $768_{10} - 274_{10}$:
  $768_{10} - 274_{10} = 768_{10} + (-274_{10})$, with 726 the 10's complement of 274

	0111	0110	1000
	+0111	0010	0110
	1110	1000	1110 → Left and right groups are invalid
	+0110	0000	0110 → Add 6
1	0100	1001	0100 → Ignore carry

- The final result is $010010010100_{\text{BCD}}$ or 494_{10}.

Gray Code

- Gray code belongs to a class of code known as **minimum change code**.
- A number changes by only one bit as it proceeds from one number to the next.
- Often used to represent digital data that have been converted from analog data.

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

American Standard Code for Information Interchange (ASCII)

- Many applications require not only handling numbers, but also letters.
- It is necessary to have a binary code for the alphabet and special characters.
- **ASCII**: Seven bits to code 128 characters.

$b_4b_3b_2b_1$	$b_7b_6b_5$ = 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

Control Characters

NUL	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

Figure: ASCII (Table 1.7 of [1]).

Error Detection Codes

- Binary information may be transmitted through some form of communication medium such as wires or radio waves or fiber optic cables, etc.
- Any external noise introduced into a physical communication medium changes bit values from 0 to 1 or vice versa.
- An **error detection code** can be used to detect errors during transmission.
  - The detected error cannot be corrected, but its presence is indicated.

Error Detection Codes

Parity Bit

- A **parity bit** (校验位) is an extra bit included with a message to make the total number of 1's either even or odd.

	With even parity (偶校验位)	With odd parity (奇校验位)
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

Error Detection Codes

Checksum

- Parity bit technique fails for double errors.
- **Checksum** adds all transmitted bytes and transmit the result as an error detection code.
- **Example:** Initially any word A is transmitted; next, another word B is transmitted.
  1. The binary digits in the two words are added and the sum obtained is retained in the transmitter.
  2. Then any other word C is transmitted and added to the previous sum retained in the transmitter and the new sum is now retained.
  3. After transmitting all the words, the final sum, which is called the Checksum, is also transmitted.
  4. The same operation is done at the receiving end.
  5. There is no error if the two sums are equal.

Outline of This Lecture

Binary Logic

- **Binary logic** deals with variables that take on two discrete values and with operations that assume logical meaning.
- Three basic **logical operations**:
  - **AND**: $x \cdot y = z$ or $xy = z$.
  - **OR**: $x + y = z$.
  - **NOT**: $x' = z$ or $\bar{x} = z$.

  where the variables are designated by x, $y \in \{0, 1\}$, and z denotes the binary results.

AND		
x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

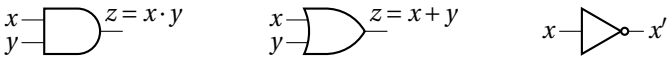
OR		
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
x	x'
0	1
1	0

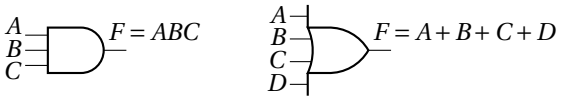
- Binary logic resembles binary arithmetic.
- However, binary logic should not be confused with binary arithmetic.
  - An arithmetic variable designates a number that may consist of many digits.
  - A logic variable is always either 0 or 1.

Logic Gate

- **Logic gates** are electronic circuits that operates on one or more inputs signals to produce an output.



- It is fine to have more than two inputs for AND/OR.



Logic Gate

Example

- Voltage-operated, though on a range, interpreted to be either of the two values.

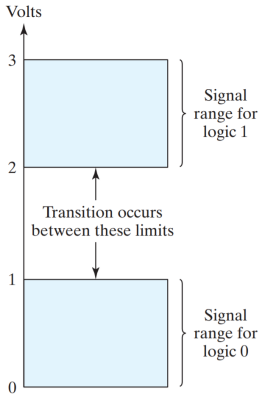


Figure: Signal levels for binary logic values (Figure 1.3 of [1]).

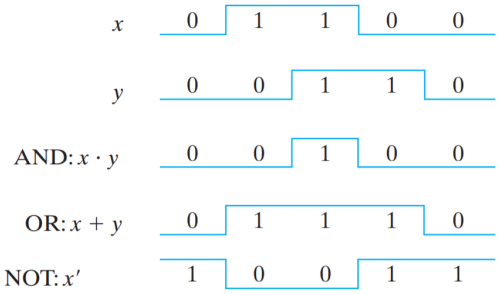


Figure: Input–output signals for gates (Figure 1.5 of [1]).

Outline of This Lecture

Summary

- Digital circuits play an important role in our life.
- Binary, octal and hexadecimal numbers are commonly used as inputs to digital circuits, and their conversions and calculations follow deterministic rules.
- Data can be represented using different code types.

Next week: Boolean Algebra and Logic Gates (布尔代数 & 逻辑门)

Recommended Reading

- Essential reading for this lecture:
  - Pages 1-33 of the textbook [1].
- Essential reading for next lecture (16 September 2022): Boolean Algebra and Logic Gates (布尔代数 & 逻辑门)
  - Pages 38-68 of the textbook [1].

[1] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

PDF available on Sakai !

Today's Lab

- The lab session will start from this week. You won't want to miss the first lab if you want to succeed in this course.
- Main content: USING VIVADO + FPGA DEVELOPMENT BOARD