Digital Logic Design
Chapter NO 1 Compiled Notes

Instructor: Wali Ullah Shinwari
Assistant Professor at Department of IT, Faculty of Computer Science Kardan University

Mob#: 0093782358093

Email: w.shinwari@kardan.edu.af

DLD Compiled Notes

Lecture 1: introduction to DLD

DLD Compiled Notes

1.1 Introduction to Digital Logic Design (DLD)

What is Digital Logic Design (DLD)?

Digital Logic Design (DLD) is a field of study that focuses on how electronic circuits process information using only two states: **ON (1)** and **OFF (0)**. These two states form the foundation of all modern computers, smartphones, and other digital devices.

Imagine a simple light switch: it's either ON or OFF. Digital circuits work similarly but with billions of tiny switches (transistors) that help process and store data.

Why is Digital Logic Important?

Everything a computer does—calculating numbers, running apps, displaying images—happens because of digital logic. Without it, we wouldn't have devices like:

- Computers and laptops
- Smartphones and tablets
- Digital cameras
- Smart home devices (like Alexa, smart TVs, and security systems)
- Cars with automated systems (self-driving technology, sensors, etc.)

How Does Digital Logic Work?

At the core of digital logic are small building blocks called **logic gates** (AND, OR, NOT, etc.). These gates take input signals (0s and 1s) and produce outputs based on predefined rules. For example:

- An **AND gate** gives 1 only if both inputs are 1.
- An **OR gate** gives 1 if at least one input is 1.
- A **NOT gate** flips 0 to 1 and 1 to 0.

These gates work together to perform tasks like adding numbers, storing data, and running software.

1.2 Number Systems, Digits, and Numbers

What is a Number System?

A number system is a method of representing numbers using symbols (digits). Different number systems exist, and each has its own rules for counting and performing calculations.

What is a Digit?

A **digit** is a single symbol used to represent a value in a number system.

- In the decimal system, digits range from **0 to 9**.
- In the binary system, only **0 and 1** are used.
- In the hexadecimal system, digits include **0-9 and A-F**.

What is a Number?

A **number** is a combination of digits used to represent a specific value.
For example:

- The number **256** in decimal consists of the digits **2, 5, and 6**.
- The binary number **1011** consists of the digits **1, 0, 1, and 1**.

Each number system follows a different set of rules based on its base (explained in the next section).

1.3 Four Main Types of Number Systems

1. Binary Number System (Base-2)

The **binary number system** is the simplest system used in computers. It has only two digits: **0 and 1**.

How Binary Works

Each digit in a binary number represents a power of **2**, starting from the right.
For example, the binary number **1011** is calculated as:
$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 8 + 0 + 2 + 1 = 11$ (decimal)

Where is Binary Used?

- In all computers, as they store and process data in binary.
- In digital circuits, where **0 means OFF** and **1 means ON**.
- In networking and IP addressing.

---

2. Decimal Number System (Base-10)

The **decimal system** is the one we use in daily life. It has ten digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**.

How Decimal Works

Each digit in a decimal number represents a power of **10**.
For example, the number **238** is calculated as:
$(2 \times 10^2) + (3 \times 10^1) + (8 \times 10^0) = 200 + 30 + 8 = 238$

Where is Decimal Used?

- In money, measurements, and everyday math.
- In calculations and general computation.

---

3. Octal Number System (Base-8)

The **octal system** uses **eight digits: 0 to 7**.

How Octal Works

Each digit in an octal number represents a power of **8**.
For example, the octal number **725_8** is calculated as:
$(7 \times 8^2) + (2 \times 8^1) + (5 \times 8^0) = 448 + 16 + 5 = 469$ (decimal)

Where is Octal Used?

- In computing, to simplify binary representation.
- In old computer systems and microcontrollers.

---

4. Hexadecimal Number System (Base-16)

The **hexadecimal system** (or simply "hex") uses **16 symbols**:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (where A=10, B=11, ..., F=15).

How Hexadecimal Works

Each digit in a hexadecimal number represents a power of **16**.
For example, the hex number **$1A3_{16}$** is calculated as:
$(1 \times 16^2) + (A \times 16^1) + (3 \times 16^0)$
$= (1 \times 256) + (10 \times 16) + (3 \times 1)$
$= 256 + 160 + 3 = 419$ **(decimal)**

Where is Hex Used?

- **In computer memory addresses** (e.g., file storage locations).
- **In RGB color codes** for web design (e.g., **#FF5733** represents an orange color).
- **In programming and debugging**, where hexadecimal makes it easier to read binary data.

Comparing the Four Number Systems

Number System	Base	Digits Used	Example	Where It's Used
Binary	2	0, 1	$1011_2 = 11_{10}$	Computers, digital circuits
Decimal	10	0-9	$347_{10} = 347_{10}$	Everyday math, finance
Octal	8	0-7	$725_8 = 469_{10}$	Legacy computing
Hexadecimal	16	0-9, A-F	$1A3_{16} = 419_{10}$	Memory addressing, color codes

---

Weighted Form Representation of Number Systems

In any number system, numbers are represented using **digits** that are assigned a specific value based on their **position**. This concept is called **weighted notation**, where each digit is multiplied by a weight (a power of the base).

The **value of a number** is determined by summing up the products of each digit and its respective place value. This applies to **binary (base-2), decimal (base-10), octal (base-8), and hexadecimal (base-16)** systems.

Weighted Form Representation of Number Systems

In any number system, numbers are represented using **digits** that have specific values based on their **position**. This concept is called **weighted notation**, where each digit is multiplied by a weight (a power of the base).

The **value of a number** is determined by summing up the products of each digit and its respective place value. This applies to **binary (base-2), decimal (base-10), octal (base-8), and hexadecimal (base-16)** systems.

Examples of Weighted Form Representation

1. Decimal Number System (Base-10)

Example: 357_{10}

The **decimal system** uses **base-10**, meaning each digit is multiplied by a power of **10** based on its position.

357_{10} = (3 × 10^2) + (5 × 10^1) + (7 × 10^0)
= (3 × 100) + (5 × 10) + (7 × 1)
= 300 + 50 + 7
= 357_{10}

---

2. Binary Number System (Base-2)

Example: 1011_2

The **binary system** uses **base-2**, meaning each digit is multiplied by a power of **2**.

1011_2 = (1 × 2^3) + (0 × 2^2) + (1 × 2^1) + (1 × 2^0)
= (1 × 8) + (0 × 4) + (1 × 2) + (1 × 1)
= 8 + 0 + 2 + 1
= 11_{10}

---

3. Octal Number System (Base-8)

Example: 725_8

The **octal system** uses **base-8**, meaning each digit is multiplied by a power of **8**.

725_8 = (7 × 8^2) + (2 × 8^1) + (5 × 8^0)

= (7 × 64) + (2 × 8) + (5 × 1)
= 448 + 16 + 5
= 469_{10}

---

4. Hexadecimal Number System (Base-16)

Example: $2F3_{16}$

The **hexadecimal system** (base-16) uses digits **0-9 and A-F** (where **A = 10, B = 11, ..., F = 15**).

$2F3_{16}$ = (2 × 16^2) + (F × 16^1) + (3 × 16^0)
= (2 × 256) + (15 × 16) + (3 × 1)
= 512 + 240 + 3
= 755_{10}

DLD Compiled Notes

A short note

✓ **Each digit in a number has a weight based on its position.**
✓ **The weight is determined by the base raised to the power of the digit's position.**
✓ **The sum of all weighted values gives the decimal equivalent of the number.**

Understanding **weighted notation** is crucial for **converting numbers between different systems** and for **computer operations that process binary data**.

Number System Conversions – 12 Examples

1. Binary to Decimal

Example: Convert **10110_2** to decimal.

10110_2 = (1 × 2^4) + (0 × 2^3) + (1 × 2^2) + (1 × 2^1) + (0 × 2^0)

= 16 + 0 + 4 + 2 + 0

= 22_{10}

2. Decimal to Binary

Example: Convert **29_{10}** to binary.

29 ÷ 2 = 14 remainder 1

14 ÷ 2 = 7 remainder 0

7 ÷ 2 = 3 remainder 1

3 ÷ 2 = 1 remainder 1

1 ÷ 2 = 0 remainder 1

Reading remainders from bottom to top: **11101_2**

3. Binary to Octal

Example: Convert **110101_2** to octal.

Group into sets of three from right: 110 101

Convert each group: $110_2 = 6_8$, $101_2 = 5_8$

Final answer: **65_8**

4. Octal to Binary

Example: Convert **34_8** to binary.

$3_8 = 011_2$

$4_8 = 100_2$

Final answer: **011100_2**

5. Binary to Hexadecimal

Example: Convert **10111110_2** to hexadecimal.

Group into sets of four from right: 1011 1110

Convert each group: $1011_2 = B_{16}$, $1110_2 = E_{16}$

Final answer: **BE_{16}**

6. Hexadecimal to Binary

Example: Convert **$3F_{16}$** to binary.

$3_{16} = 0011_2$

$F_{16} = 1111_2$

Final answer: **00111111_2**

7. Decimal to Octal

Example: Convert **83_{10}** to octal.

83 ÷ 8 = 10 remainder 3

10 ÷ 8 = 1 remainder 2

1 ÷ 8 = 0 remainder 1

Reading remainders from bottom to top: **123_8**

8. Octal to Decimal

Example: Convert **245_8** to decimal.

245_8 = (2 × 8^2) + (4 × 8^1) + (5 × 8^0)

= (2 × 64) + (4 × 8) + (5 × 1)

= 128 + 32 + 5

= **165_{10}**

9. Decimal to Hexadecimal

Example: Convert **255_{10}** to hexadecimal.

255 ÷ 16 = 15 remainder F

15 ÷ 16 = 0 remainder F

Reading remainders from bottom to top: **FF_{16}**

10. Hexadecimal to Decimal

Example: Convert **$2A_{16}$** to decimal.

$2A_{16}$ = (2 × 16^1) + (A × 16^0)

= (2 × 16) + (10 × 1)

= 32 + 10

= **42_{10}**

11. Octal to Hexadecimal

Example: Convert 72_8 to hexadecimal.

1. Convert 72_8 to binary:

$7_8 = 111_2$

$2_8 = 010_2$

So, $72_8 = 111010_2$

So, $72_8 = 111010_2$

2. **Convert binary 111010_2 to hexadecimal:**

Group into four: 11 1010

Convert: $11_2 = 3_{16}$, $1010_2 = A_{16}$

Final answer: **$3A_{16}$**

12. Hexadecimal to Octal

Example: Convert $4F_{16}$ to octal.

1. Convert $4F_{16}$ to binary:

$4_{16} = 0100_2$
$F_{16} = 1111_2$
So, $4F_{16} = 01001111_2$
   2. Convert binary $\mathbf{01001111_2}$ to octal:
Group into three: 010 011 111
Convert: $010_2 = 2_8$, $011_2 = 3_8$, $111_2 = 7_8$
Final answer: **237_8**

Binary Arithmetic Examples

1. Binary Addition (+)

Example 1:

Add **$1011_2 + 1101_2$**

```
  1011
+ 1101
-------
```

11000 (Carry over the extra bits) Explanation:

- 1 + 1 = 10 (Write down 0, carry over 1)
- 1 + 0 + 1 (carry) = 10 (Write down 0, carry over 1)
- 0 + 1 + 1 (carry) = 10 (Write down 0, carry over 1)
- 1 + 1 + 1 (carry) = 11 (Write down 1, carry over 1)

Example 2:

Add $\mathbf{11001_2 + 1010_2}$

```
  11001
+  1010
-------
  100011  (Carry over the extra bits)
```

Explanation:

- 1 + 0 = 1
- 0 + 1 = 1
- 0 + 0 = 0
- 1 + 1 = 10 (Write down 0, carry over 1)
- 1 + 1 + 1 (carry) = 11 (Write down 1, carry over 1)

Final result: $\mathbf{100011_2}$

2. Binary Subtraction (-)

Example 1:

Subtract $\mathbf{1011_2 - 0110_2}$

```
   1011
-  0110
-------
   0101
```

Explanation:

- 1 - 0 = 1
- 1 - 1 = 0
- 0 - 1 = 1 (borrow from the next digit)
- 1 - 0 = 1

Final result: **0101_2**

Example 2:

Subtract **11010_2 - 1001_2**

```
   11010
-  1001
-------
   1001
```

Explanation:

- 0 - 1 = 1 (borrow from the next digit)
- 1 - 0 = 1
- 0 - 0 = 0
- 1 - 1 = 0
- 1 - 0 = 1

Final result: **1001_2**

3. Binary Multiplication (*)

Example 1:

Multiply **101_2 * 11_2**

```
     101
   x  11
  -------
     101   (101 * 1)
+   1010   (101 * 1, shifted one position to the left)
  -------
    1111
```

Explanation:

- 101 * 1 = 101
- 101 * 1 (shifted) = 1010
- Adding the results: **1111_2**

Final result: **1111_2**

Example 2:

Multiply $\mathbf{110_2 * 101_2}$

```
     110
   x 101
  -------
     110  (110 * 1)
+  0000   (110 * 0, shifted one position to the left)
+ 11000   (110 * 1, shifted two positions to the left)
  -------
   11110
```

Explanation:

- 110 * 1 = 110
- 110 * 0 (shifted) = 0000
- 110 * 1 (shifted) = 11000
- Adding the results: $\mathbf{11110_2}$

Final result: $\mathbf{11110_2}$

---

4. Binary Division (/)

Example 1:

Divide $\mathbf{10110_2 \div 10_2}$

```
     1011  (Quotient)
    --------
10 | 10110
     10
    ----
      001
      10
    ----
       10
       10
    ----
        0
```

Explanation:

- 10 goes into 101, the result is 10 (quotient is 10, remainder is 0)
- Bring down the next 1 and repeat the process.

- Final quotient: **1011_2**, remainder: **0**

Final result: **1011_2**

Example 2:

Divide **$110101_2 \div 11_2$**

```
      1011   (Quotient)
    ---------
11 | 110101
     11
    ----
     010
     11
    ----
      101
      11
    ----
       00
```

Explanation:

- 11 goes into 110, the result is 10 (quotient is 10, remainder is 0)
- Repeat the process with the next digits.
- Final quotient: **1011_2**, remainder: **0**

Final result: **1011_2**

3. Binary Multiplication (*)

Example 1:

Multiply **$101_2 * 11_2$**

```
sql
CopyEdit
     101
   x  11
  -------
     101   (101 * 1)
+  1010    (101 * 1, shifted one position to the left)
  -------
    1111
```

Explanation:

- 101 * 1 = 101
- 101 * 1 (shifted) = 1010
- Adding the results: **1111_2**

Final result: **1111_2**

Example 2:

Multiply **110_2 * 101_2**

```
css
CopyEdit
     110
   x 101
  -------
     110  (110 * 1)
+  0000   (110 * 0, shifted one position to the left)
+ 11000   (110 * 1, shifted two positions to the left)
  -------
   11110
```

Explanation:

- 110 * 1 = 110
- 110 * 0 (shifted) = 0000
- 110 * 1 (shifted) = 11000
- Adding the results: **11110_2**

Final result: **11110_2**

---

4. Binary Division (/)

Example 1:

Divide **10110_2 ÷ 10_2**

```
lua
CopyEdit
     1011  (Quotient)
    --------
10 | 10110
     10
    ----
      001
```

```
  10
----
   10
   10
----
    0
```

Explanation:

- 10 goes into 101, the result is 10 (quotient is 10, remainder is 0)
- Bring down the next 1 and repeat the process.
- Final quotient: $\mathbf{1011_2}$, remainder: **0**

Final result: $\mathbf{1011_2}$

Example 2:

Divide $\mathbf{110101_2 \div 11_2}$

```
lua
CopyEdit
      1011   (Quotient)
    ---------
11 | 110101
     11
    ----
     010
     11
    ----
      101
      11
    ----
       00
```

Explanation:

- 11 goes into 110, the result is 10 (quotient is 10, remainder is 0)
- Repeat the process with the next digits.
- Final quotient: $\mathbf{1011_2}$, remainder: **0**

Final result: $\mathbf{1011_2}$