

Electronics and Physical Computing

1. Introduction to Physical Computing

1.1 Definition

Physical computing refers to the use of tangible, embedded microcontroller-based interactive systems that can sense and respond to the physical world. These systems create a bridge between the digital world of computers and the physical world we live in.



1.2 Real-World Applications

- Temperature-controlled fans responding to room temperature
- Motion-sensing lights activating when someone enters a room
- Smart home automation systems
- Interactive art installations
- Robotics and automation systems



2. Physical Computing Platforms

2.1 Available Platforms

- Arduino (Most popular for beginners)
- Raspberry Pi
- BBC Micro:Bit
- ESP8266/ESP32
- Particle
- Tessel 2
- BeagleBone
- Makey Makey
- Teensy



3. Understanding Microcontrollers

3.1 What is a Microcontroller?

A microcontroller is a compact integrated circuit (IC) designed to perform specific tasks in embedded systems. It contains:

- Processor (CPU)
- Memory (RAM, ROM)
- Input/output (I/O) peripherals

All these components are integrated into a single chip.



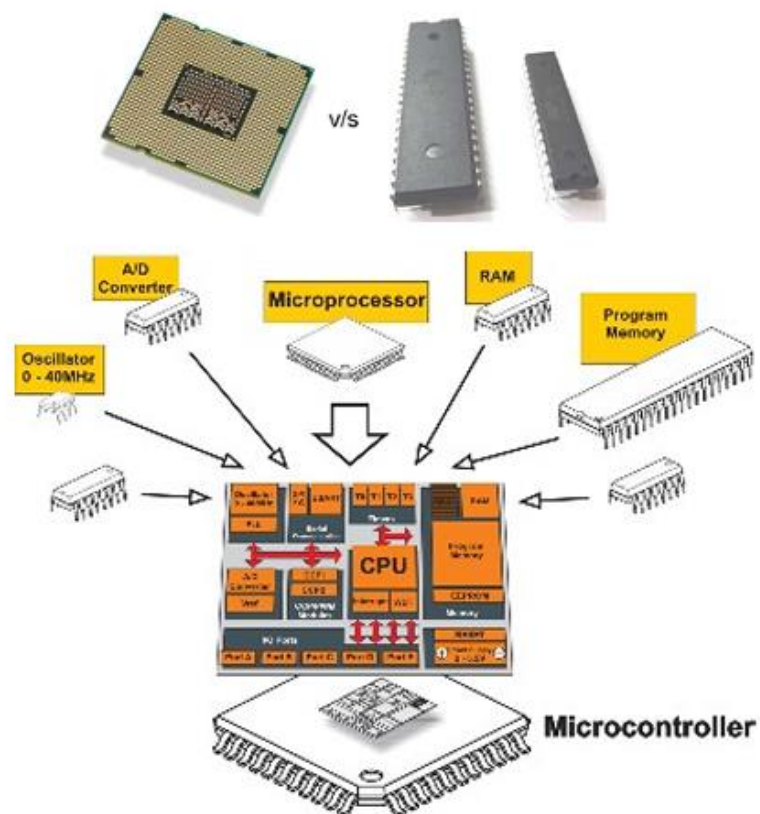
3.2 Microcontroller vs. Microprocessor

Microcontroller:

- Designed for specific tasks
- Built-in memory and I/O peripherals on the same chip
- Used in embedded systems like washing machines, robots, and IoT devices
- Optimized for specific applications

Microprocessor:

- General-purpose CPU
- Requires external components (memory, I/O devices)
- Used in computers, smartphones, and servers
- Optimized for versatility



3.3 Microcontroller Programmer

A hardware device or software tool used to load ("flash") programs or firmware into the microcontroller's memory. It acts as a bridge between the development environment and the microcontroller.



Figure: PICkit 3 Programmer

4. Arduino Platform

4.1 What is Arduino?

Arduino is an open-source electronics platform consisting of:

- Hardware (microcontroller boards)
- Software (Arduino IDE)
- Extensive community and resources

4.2 Key Features

1. Open-Source:

- Hardware designs and software are freely available
- Community can create and share modifications
- Large library of resources and tutorials

2. Plug-and-Play Hardware:

- Easy connection of components
- Digital and analog I/O pins
- Built-in USB communication

3. Cross-Platform:

- Works on Windows, macOS, and Linux
- Consistent development environment

4.3 Arduino Boards

Arduino Uno (Most Popular)

- Based on ATmega328P microcontroller
- 14 digital I/O pins
- 6 analog input pins
- 16 MHz clock speed
- USB connection
- Power jack
- ICSP header
- Reset button



Figure: Arduino Uno R3 Board

Other Arduino Boards

- Arduino Nano: Compact version for small projects
- Arduino Mega: More I/O pins and memory
- Arduino Leonardo: Built-in USB communication
- Each designed for specific use cases

4.4 Arduino Uno Technical Specifications

- Microcontroller: ATmega328P
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Digital I/O Pins: 14 (6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- Flash Memory: 32 KB (0.5 KB used for bootloader)
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz

5. Arduino Hardware Components

5.1 Digital Pins (0-13)

- Can be used as INPUT or OUTPUT
- Each can source/sink up to 40mA
- Pins 0(RX) and 1(TX) used for serial communication
- Pin 13 connected to built-in LED

5.2 Analog Pins (A0-A5)

- Read analog values (0-1023)
- Can also be used as digital I/O
- 10-bit resolution
- Default reference voltage: 5V

5.3 Power Pins

- Vin: External power input (7-12V)
- 5V: Regulated 5V output
- 3.3V: Regulated 3.3V output
- GND: Ground pins
- RESET: Board reset when connected to ground

5.4 Special Function Pins

- PWM (~): Pins 3, 5, 6, 9, 10, 11
- External Interrupts: Pins 2 and 3
- SPI: Pins 10(SS), 11(MOSI), 12(MISO), 13(SCK)
- I²C: A4(SDA) and A5(SCL)

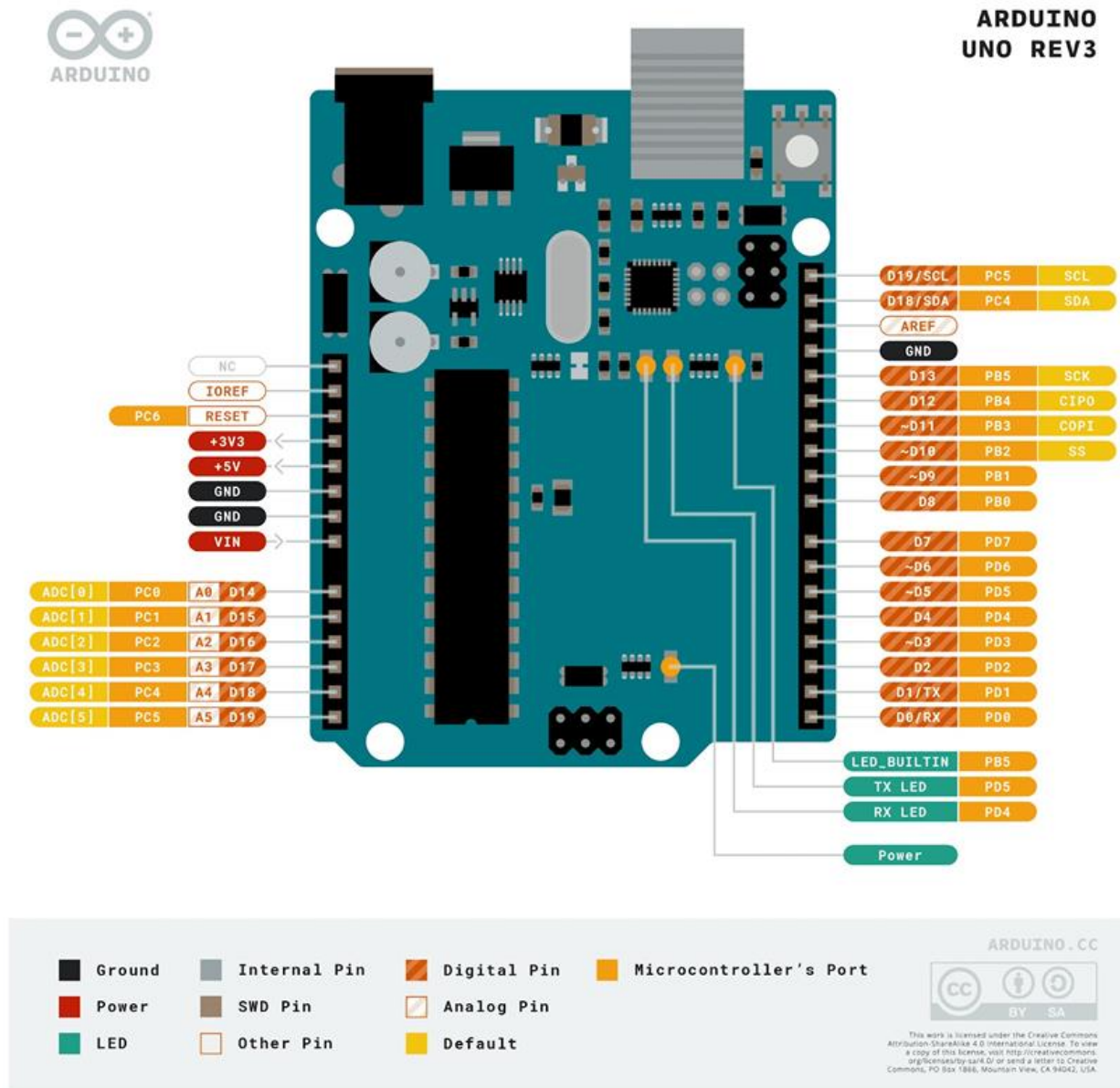


Figure: Pinout Diagram of the Arduino Uno Rev3 Board

6. Programming Arduino

6.1 Arduino IDE

- Simple, user-friendly interface
- Code editor with syntax highlighting
- One-click compile and upload
- Serial monitor for debugging
- Library manager

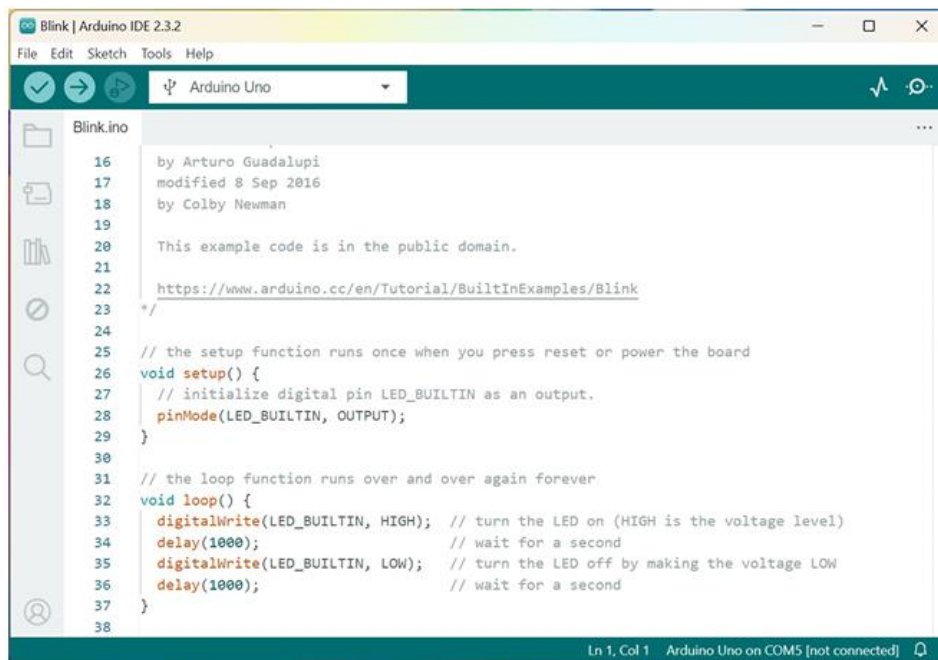


Figure: Arduino IDE 2.3.2

6.2 Basic Program Structure

```
void setup() {  
    // Runs once at startup  
    // Initialize pins and settings  
}
```

```
void loop() {  
    // Runs continuously  
    // Main program logic  
}
```


6.3 Common Functions

```
// Digital I/O

pinMode(pin, MODE);    // Set pin mode (INPUT/OUTPUT)

digitalWrite(pin, value); // Write HIGH or LOW

digitalRead(pin);      // Read digital pin


// Analog I/O

analogRead(pin);       // Read analog value

analogWrite(pin, value); // Write PWM value

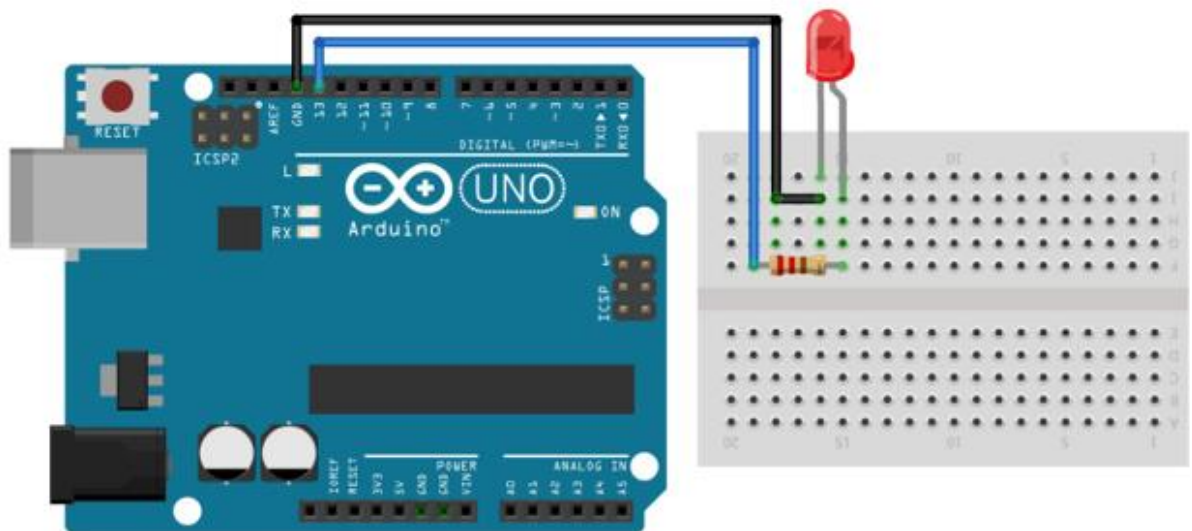

// Time Functions

delay(ms);             // Pause program

millis();              // Get runtime
```

7. Basic Projects and Examples

7.1 LED Control



fritzing

Figure: Wiring Diagram of the Circuit

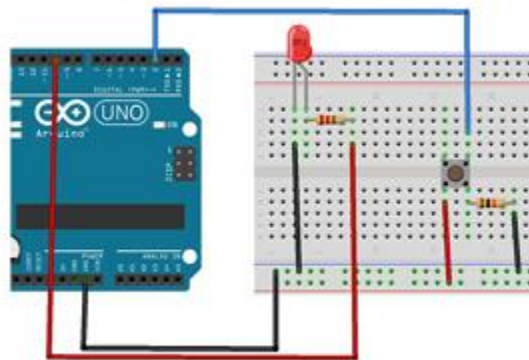
Basic LED Blink Code:

```
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

7.2 Push Button Input



Button Read Code:

```
const int buttonPin = 2;
const int ledPin = 13;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}
```

```

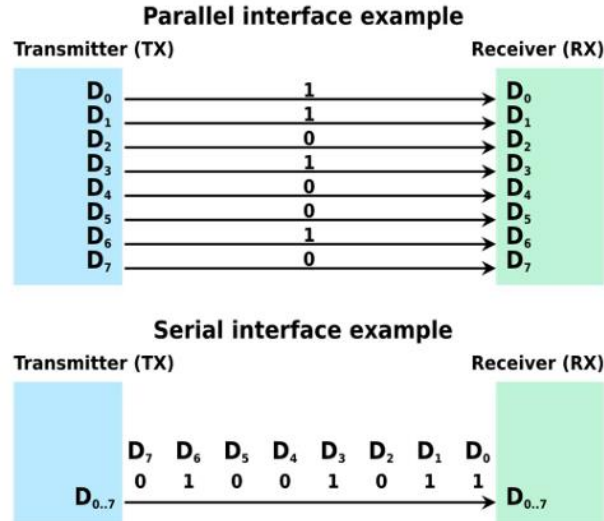
void loop() {
    if (digitalRead(buttonPin) == HIGH) {
        digitalWrite(ledPin, HIGH);
    } else {
        digitalWrite(ledPin, LOW);
    }
}

```

8. Serial Communication

8.1 Basics of Serial Communication

- Asynchronous vs Synchronous
- Baud rate (commonly 9600)
- Data packets and framing



8.2 Communication Types

1. UART (Universal Asynchronous Receiver/Transmitter)
2. SPI (Serial Peripheral Interface)
3. I²C (Inter-Integrated Circuit)
4. RS-232 (Legacy standard)

8.3 Serial Communication Code Example

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    if (Serial.available() > 0) {  
        char incomingByte = Serial.read();  
        Serial.print("Received: ");  
        Serial.println(incomingByte);  
    }  
}
```

9. Safety and Best Practices

9.1 Electrical Safety

- Never exceed maximum pin current (40mA)
- Always use current-limiting resistors with LEDs
- Avoid short circuits
- Use proper power supply voltage

9.2 Programming Best Practices

- Comment your code
- Use meaningful variable names
- Break complex tasks into functions
- Include error checking
- Use serial monitor for debugging

9.3 Hardware Best Practices

- Double-check connections before power-up
- Use breadboard for prototyping

- Keep wires organized and color-coded
- Use proper tools for assembly
- Protect board from static electricity

10. Troubleshooting Guide

10.1 Common Issues

- Upload errors
- Power problems
- Connection issues
- Code compilation errors
- Serial communication problems

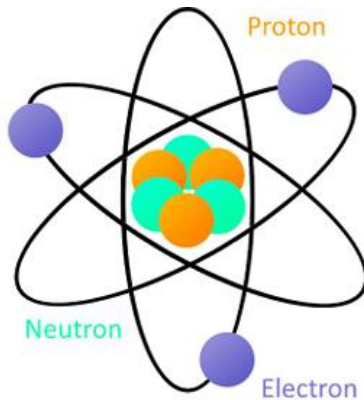
10.2 Debugging Steps

1. Check physical connections
2. Verify power supply
3. Confirm correct board and port selection
4. Use `Serial.println()` for debugging
5. Test components individually
6. Check code syntax and logic

Part 2

1. Understanding Electricity

1.1 The Nature of Electricity



Electricity fundamentally involves the behavior of electrons within atoms. Every atom consists of:

- Protons (positive charge) in the nucleus
- Neutrons (no charge) in the nucleus
- Electrons (negative charge) orbiting the nucleus

The movement of electrons between atoms creates electrical current. This forms the basis of all electrical and electronic systems.

1.2 Historical Development

Key discoveries that shaped our understanding of electricity:

1. Benjamin Franklin (1772)

- Demonstrated electricity in clouds
- Established positive/negative charge concept
- Invented the lightning rod

2. Luigi Galvani (1780)

- Discovered "animal electricity"
- Pioneer in bioelectricity
- Famous frog leg experiments

3. Alessandro Volta (1800)

- Invented first chemical battery

- Created continuous electrical current
- Unit 'Volt' named after him

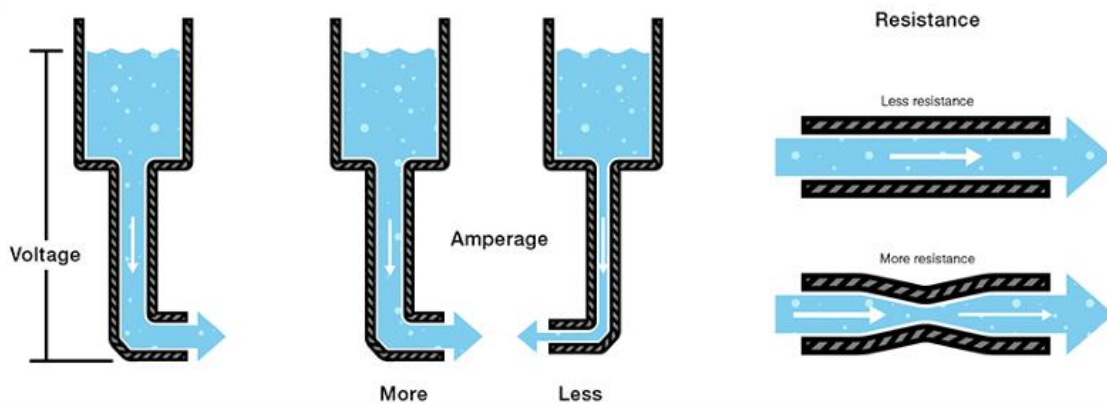
4. André-Marie Ampère (1820)

- Connected electricity and magnetism
- Developed electromagnetic theory
- Unit 'Ampere' honors his work

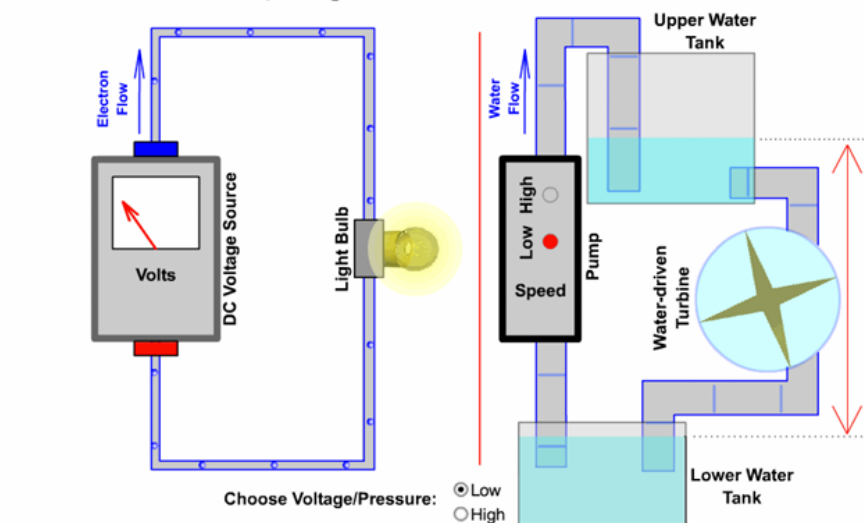
5. Georg Ohm (1827)

- Established $V = I \times R$ relationship
- Created foundation for circuit analysis
- Unit 'Ohm' commemorates his discovery

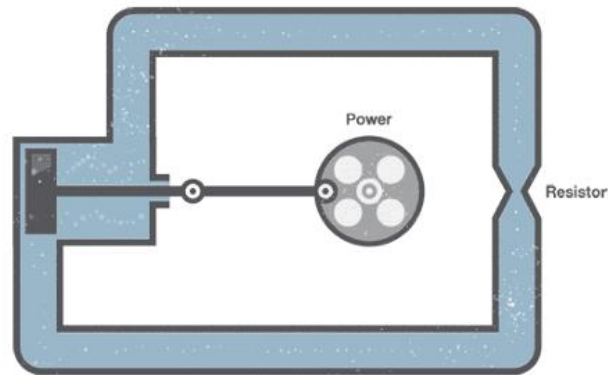
1.3 Fundamental Electrical Quantities



Comparing a DC Circuit to the Flow of Water



Alternating Current: The Water Analogy



1.3.1 Voltage (V)

- Definition: Electrical potential difference
- Unit: Volts (V)
- Characteristics:
 - * Measure of electrical "pressure"
 - * Drives current through circuits
 - * Like water pressure in pipes
- Common values:
 - * 1.5V (AA battery)
 - * 5V (USB)
 - * 230V (Mains electricity)

1.3.2 Current (I)

- Definition: Rate of electron flow
- Unit: Amperes (A)
- Characteristics:
 - * Measure of electron movement
 - * Flows from higher to lower potential
 - * Like water flow rate
- Typical values:
 - * 1-500mA (Small electronics)

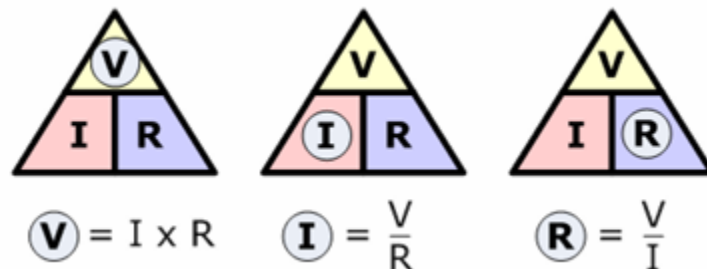
- * 1-15A (Household appliances)
- * 100A+ (Industrial equipment)

1.3.3 Resistance (R)

- Definition: Opposition to current flow
- Unit: Ohms (Ω)
- Characteristics:
 - * Controls current flow
 - * Converts electrical energy to heat
 - * Like friction in pipes
- Common values:
 - * 1Ω - $1M\Omega$ (Most circuits)
 - * $10k\Omega$ (Common in electronics)

2. Basic Electronic Concepts

2.1 Ohm's Law



The fundamental relationship: $V = I \times R$

Example calculations:

1. Finding current:

- Given: $V = 12V$, $R = 100\Omega$
- $I = V/R = 12/100 = 0.12A = 120mA$

2. Finding voltage:

- Given: $I = 0.5A$, $R = 50\Omega$

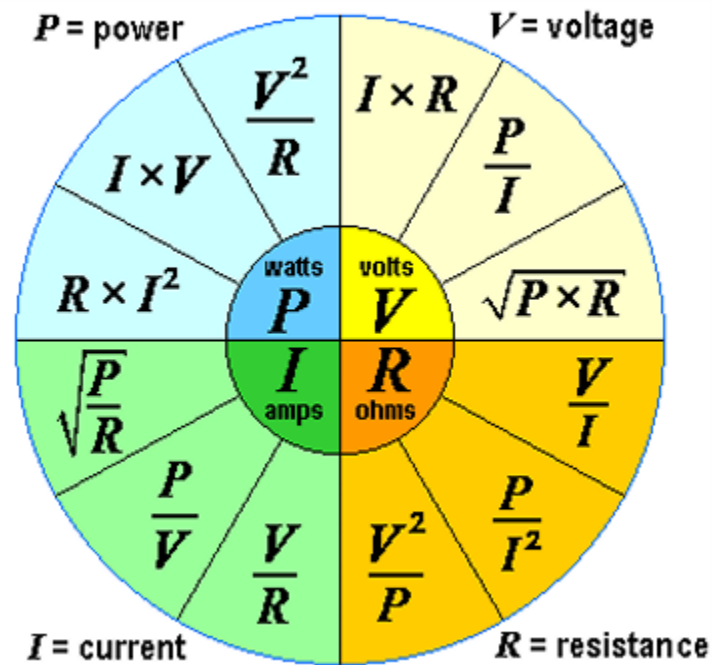
- $V = I \times R = 0.5 \times 50 = 25V$

3. Finding resistance:

- Given: $V = 5V, I = 0.1A$

- $R = V/I = 5/0.1 = 50\Omega$

2.2 Power Calculations



Multiple formulas for power:

- $P = V \times I$

- $P = I^2 R$

- $P = \frac{V^2}{R}$

Example calculations:

1. LED power consumption:

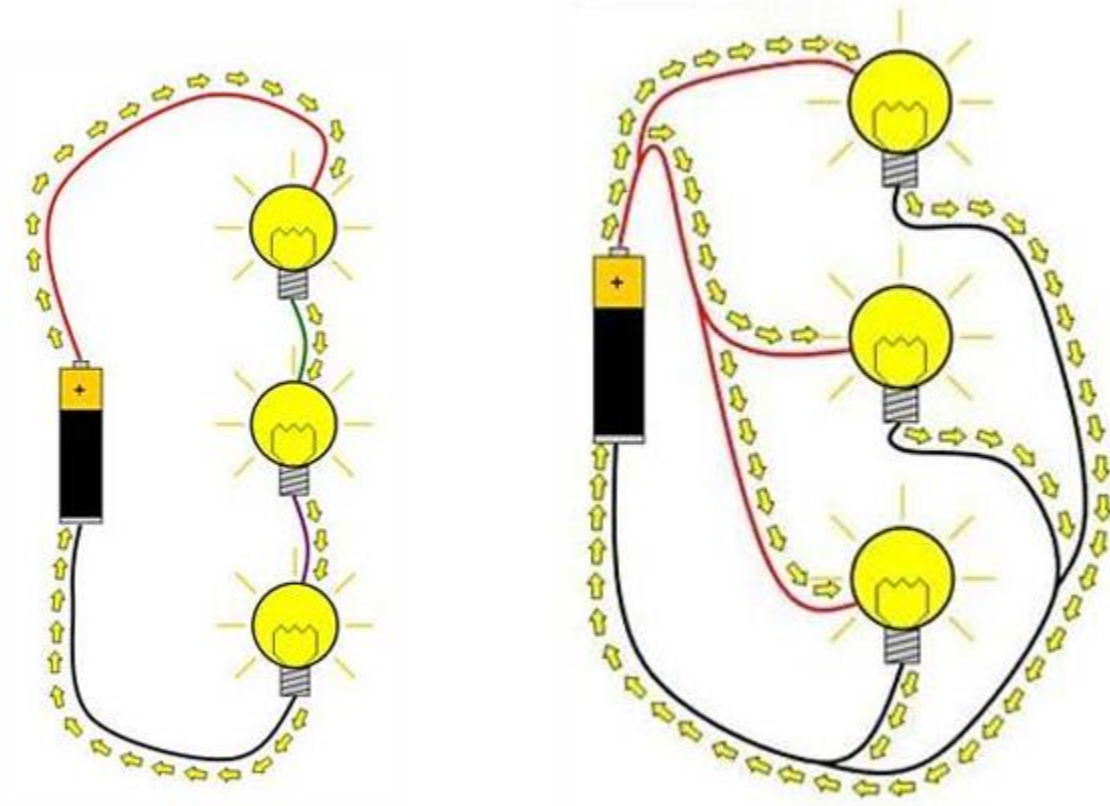
- $V = 3.3V, I = 20mA$

- $P = 3.3 \times 0.02 = 0.066W = 66mW$

2. Resistor power rating:

- $R = 100\Omega$, $I = 0.1A$
- $P = (0.1)^2 \times 100 = 1W$

3. Circuit Types and Analysis



3.1 Series Circuits

Characteristics:

- Single path for current
- Same current through all components
- Voltages add up to source voltage
- Total resistance = sum of individual resistances

Example: Given three resistors in series:

$$R_1 = 100\Omega, R_2 = 220\Omega, R_3 = 330\Omega$$

$$R_{\text{total}} = R_1 + R_2 + R_3 = 650\Omega$$

3.2 Parallel Circuits

Characteristics:

- Multiple current paths
- Same voltage across components
- Currents add up to total current
- Total resistance = $1/(1/R1 + 1/R2 + 1/R3)$

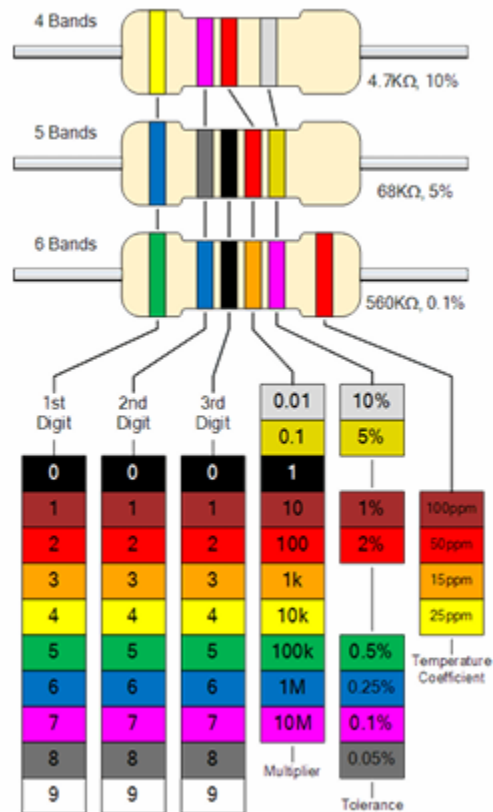
Example: Given two resistors in parallel:

$R1 = 100\Omega$, $R2 = 100\Omega$

$R_{total} = (R1 \times R2)/(R1 + R2) = 50\Omega$

4. Electronic Components

4.1 Resistors



4.1.1 Fixed Resistors

- Color code reading:

- * 1st band: First digit
- * 2nd band: Second digit
- * 3rd band: Multiplier
- * 4th band: Tolerance

Example:

Red-Violet-Orange-Gold

Red (2), Violet (7), Orange ($\times 1,000$), Gold ($\pm 5\%$)

$= 27,000\Omega \pm 5\% = 27k\Omega \pm 5\%$

4.1.2 Variable Resistors

Types:

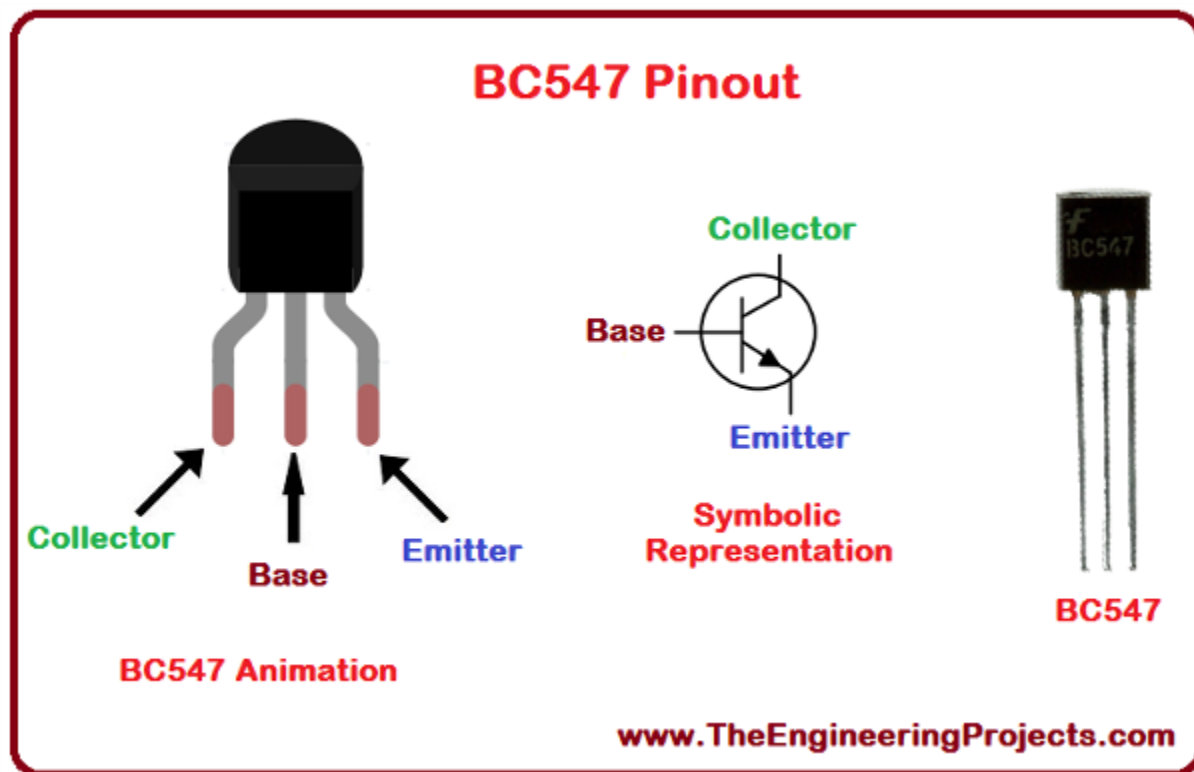
1. Potentiometers (three terminals)
2. Rheostats (two terminals)
3. Trimmer potentiometers
4. Digital potentiometers



Applications:

- Volume control
- Light dimming
- Sensor calibration
- Speed control

4.2 Transistors



BC547 NPN Transistor:

- Pins: Collector, Base, Emitter
- Maximum ratings:
 - * V_{ce0} : 45V
 - * I_c : 100mA
 - * Power: 500mW

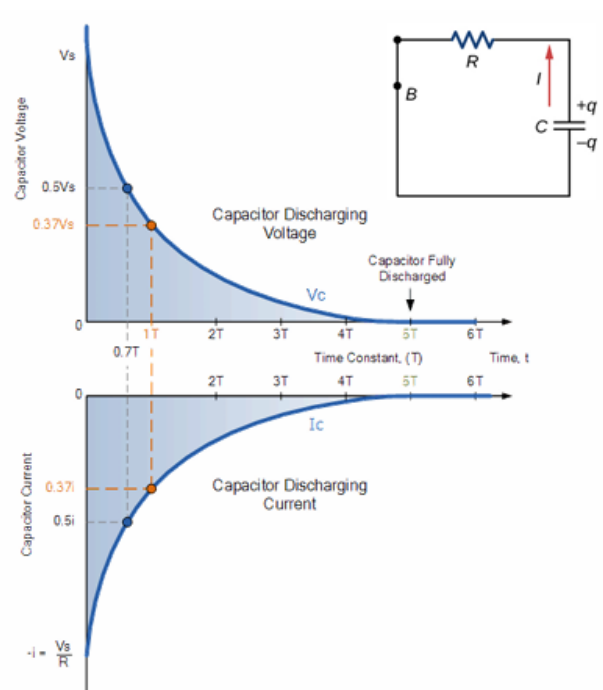
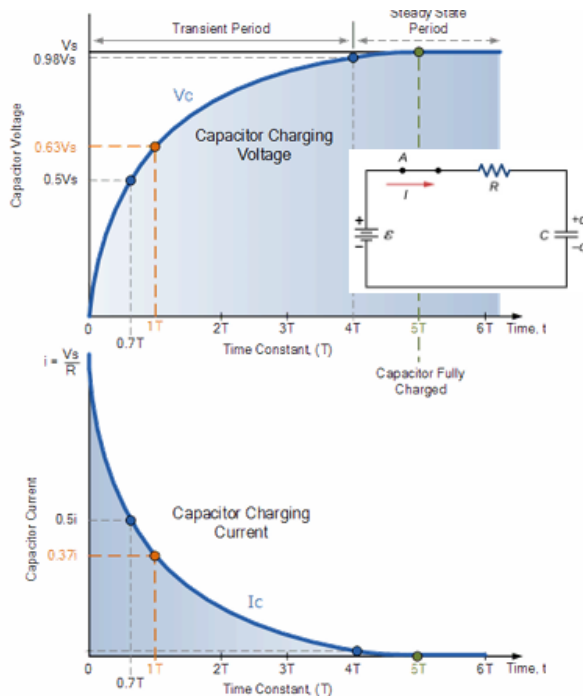
Capacitors in Digital Circuits

Basic Principles

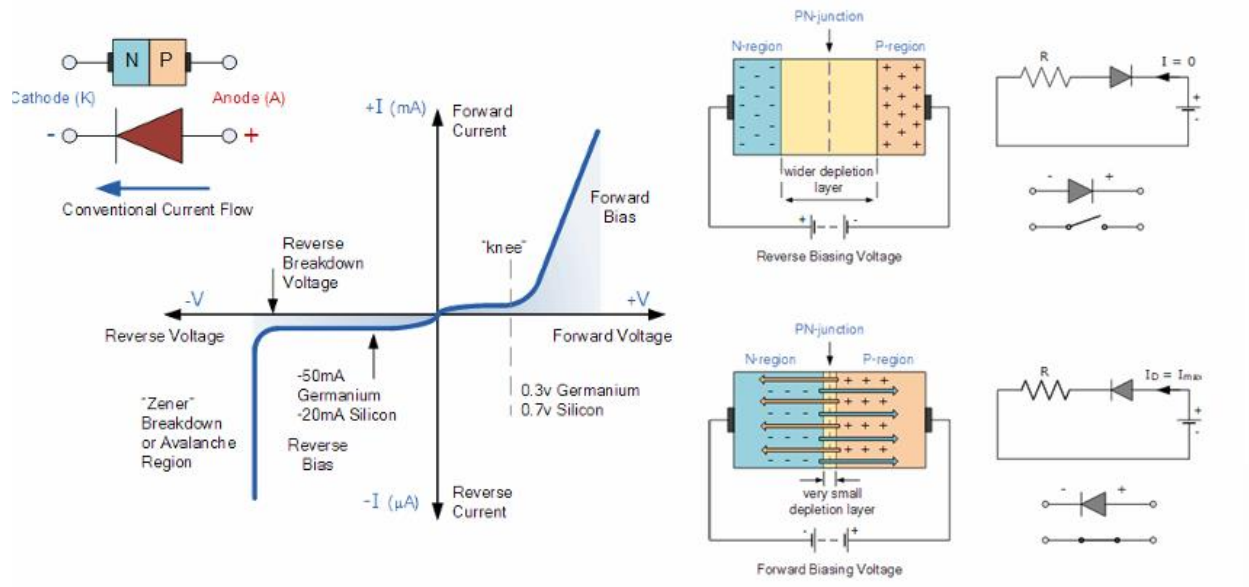
- Store electrical charge
- Used for:
 - Power supply filtering
 - Timing circuits
 - Coupling/decoupling
 - Signal smoothing

Time Constant (τ)

- $\tau = RC$ (where R is resistance in ohms, C is capacitance in farads)
- 63.2% charge after one time constant
- 95% charge after 3 time constants
- 99% charge after 5 time constants



Diodes

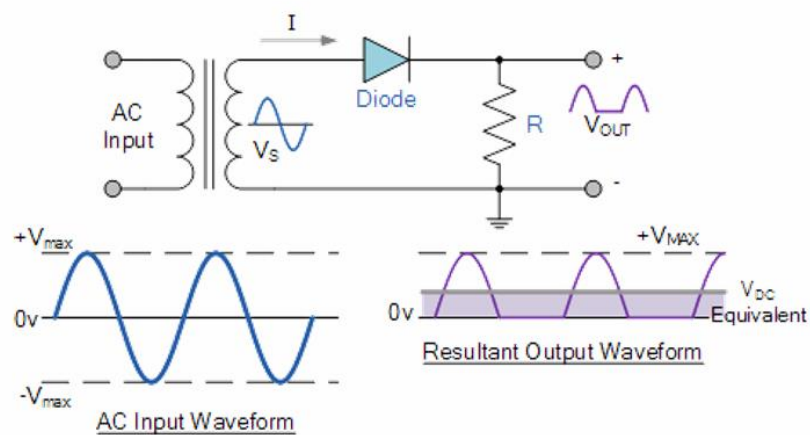


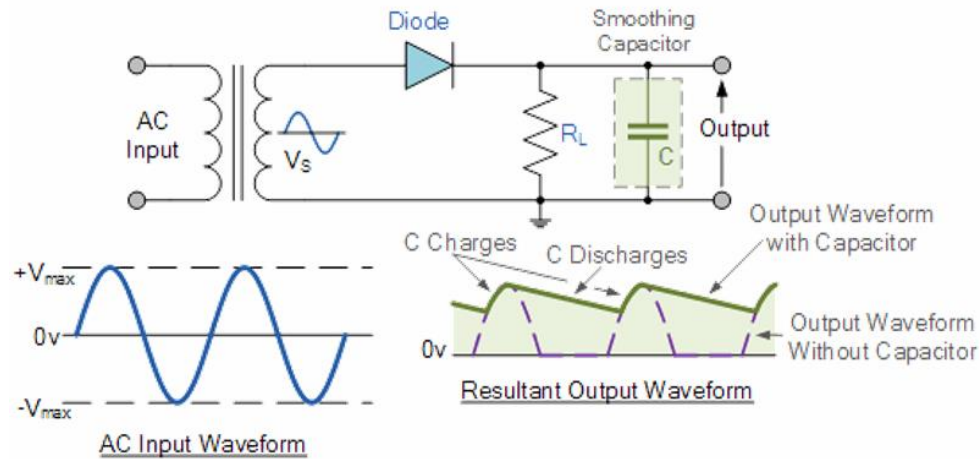
Basic Diode Operation

- Unidirectional current flow
- Forward bias: Conducts current
- Reverse bias: Blocks current
- Voltage drop: $\sim 0.7V$ (silicon)

Rectifier Circuits

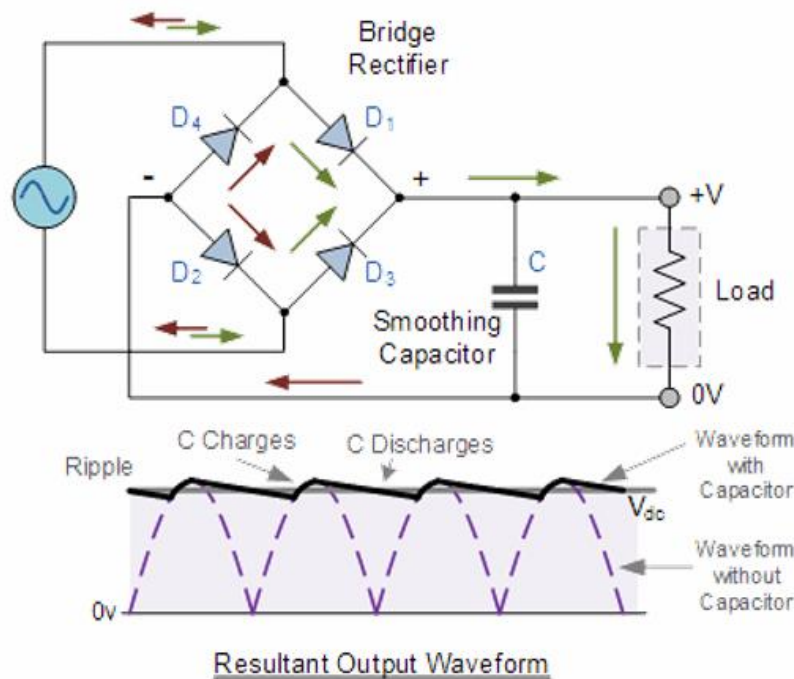
1. Half-Wave Rectification





- Single diode
- Converts AC to pulsating DC
- Output uses only half of input cycle

2. Full-Wave Rectification



- Bridge configuration (4 diodes)
- More efficient power conversion
- Smoother output

Basic Applications:

1. Switch circuit:

```
// Arduino code for transistor switch

const int basePin = 2;

void setup() {
    pinMode(basePin, OUTPUT);
}

void loop() {
    digitalWrite(basePin, HIGH); // Turn on
    delay(1000);
    digitalWrite(basePin, LOW); // Turn off
    delay(1000);
}
```

2. Light sensor circuit:

```
// Arduino code for light sensor

const int basePin = 2;
const int sensorPin = A0;

void setup() {
    pinMode(basePin, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int lightLevel = analogRead(sensorPin);
    Serial.println(lightLevel);
    digitalWrite(basePin, lightLevel < 500 ? HIGH : LOW);
    delay(100);
}
```

3. RC Time Constant Measurement

```
// Arduino RC Time Constant Measurement

const int chargePin = 8;

const int measurePin = A0;


void setup() {

  pinMode(chargePin, OUTPUT);

  Serial.begin(9600);

}


void measureRCTime() {

  // Discharge capacitor

  digitalWrite(chargePin, LOW);

  delay(1000);


  // Start charging and measuring

  unsigned long startTime = micros();

  digitalWrite(chargePin, HIGH);


  int targetVoltage = (int)(0.632 * 1023); // 63.2% of max

  while(analogRead(measurePin) < targetVoltage);


  unsigned long endTime = micros();

  unsigned long timeConstant = endTime - startTime;


  Serial.print("Time Constant: ");

  Serial.print(timeConstant);

  Serial.println(" microseconds");

}
```

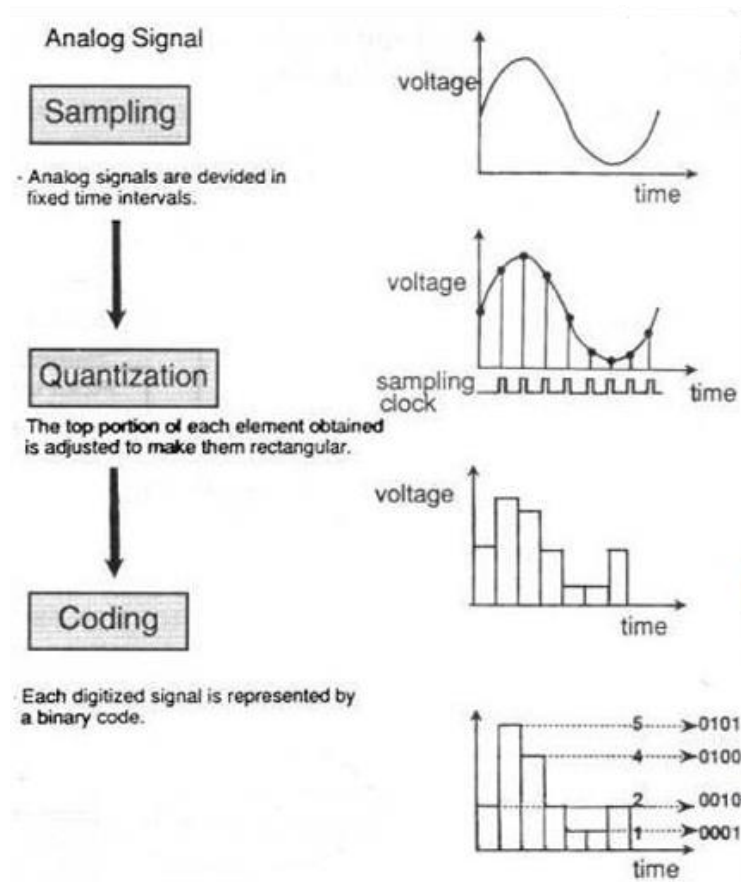
```

void loop() {
    measureRCTime();
    delay(5000);
}

```

5. Analog-to-Digital Conversion

5.1 ADC Basics



Characteristics:

- Converts analog to digital values
- Resolution determines accuracy
- Arduino uses 10-bit ADC (0-1023)

Formula:

Digital Value = (Input Voltage × 1024) / Reference Voltage

5.2 Arduino ADC Implementation

```
// Basic ADC reading

int sensorValue = analogRead(A0); // 0-1023


// Converting to voltage

float voltage = sensorValue * (5.0 / 1023.0);


// Reading with averaging for noise reduction

int getAverageReading(int pin, int samples = 10) {

    long sum = 0;

    for(int i = 0; i < samples; i++) {

        sum += analogRead(pin);

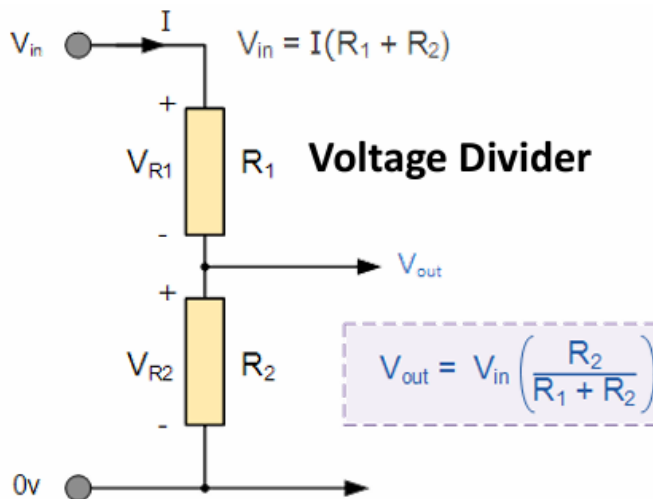
        delay(1);

    }

    return sum / samples;

}
```

5.3 Voltage Divider Applications



Formula:

$$V_{out} = V_{in} \times (R_2 / (R_1 + R_2))$$

Example for measuring 0-25V with Arduino:

```
// Voltage measurement with voltage divider

const float R1 = 40000.0; // 40kΩ
const float R2 = 10000.0; // 10kΩ

void setup() {
    Serial.begin(9600);
}

void loop() {
    int rawValue = analogRead(A0);
    float voltage = rawValue * (5.0 / 1023.0);
    float inputVoltage = voltage * ((R1 + R2) / R2);

    Serial.print("Input Voltage: ");
    Serial.println(inputVoltage);
    delay(1000);
}
```

6. Practical Projects

6.1 LED Control Project

```
// PWM LED control with potentiometer

const int ledPin = 9; // PWM pin
const int potPin = A0; // Analog input pin

void setup() {
    pinMode(ledPin, OUTPUT);
}

void loop() {
```

```

    int potValue = analogRead(potPin);

    int brightness = map(potValue, 0, 1023, 0, 255);

    analogWrite(ledPin, brightness);

    delay(10);
}

```

6.2 Temperature Monitor

```

// Temperature monitoring with LCD display
#include <LiquidCrystal.h>

const int tempPin = A0;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
    lcd.begin(16, 2);
}

void loop() {
    int sensorValue = analogRead(tempPin);

    float voltage = sensorValue * (5.0 / 1023.0);

    float tempC = (voltage - 0.5) * 100;

    lcd.setCursor(0, 0);
    lcd.print("Temp: ");
    lcd.print(tempC);
    lcd.print(" C");
    delay(1000);
}

```

Part 3

Introduction to Digital Electronics

Digital electronics forms the foundation of modern computing and electronic systems. Unlike analog electronics which deals with continuous signals, digital electronics works with discrete voltage levels, typically represented as:

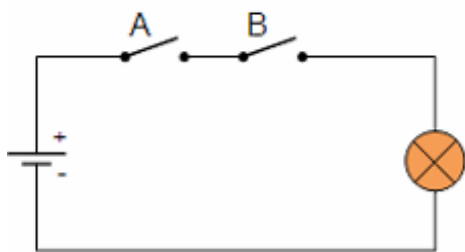
- Logic 0 (LOW): 0V
- Logic 1 (HIGH): 5V or 3.3V (depending on the system)

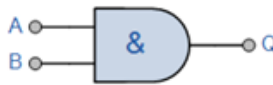
Advantages of Digital Systems

- Higher noise immunity
- Easy to store information
- More accurate signal processing
- Greater circuit stability
- Easier to design complex systems

Basic Logic Gates

AND Gate



Symbol	Truth Table		
 2-input AND Gate	B	A	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = A.B$	Read as A AND B gives Q		

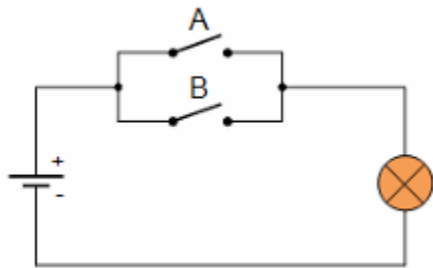
Characteristics:

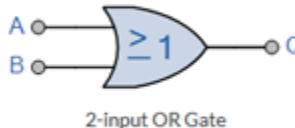
- Output is HIGH (1) only when all inputs are HIGH
- Used for condition verification
- Boolean Expression: $Y = A \cdot B$

- Truth Table:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate



Symbol	Truth Table		
 2-input OR Gate	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1
Boolean Expression $Q = A + B$		Read as A OR B gives Q	

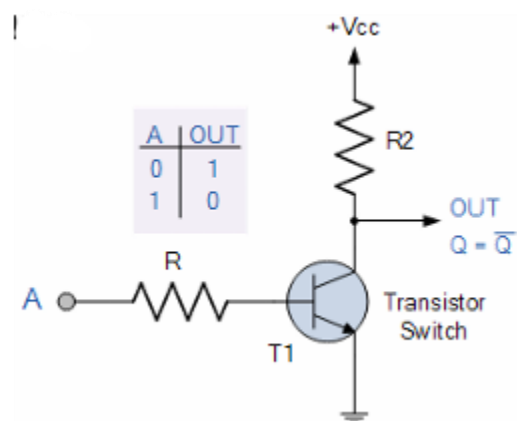
Characteristics:

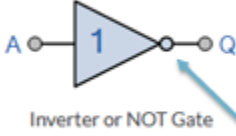
- Output is HIGH (1) when any input is HIGH
- Used for condition combining
- Boolean Expression: $Y = A + B$

- Truth Table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate (Inverter)



Symbol	Truth Table	
 Inverter or NOT Gate	A	Q
	0	1
	1	0
Boolean Expression $Q = \text{not } A$ or \bar{A}		Read as inverse of A gives Q

Characteristics:

- Single input gate
- Inverts the input signal
- Used for signal inversion
- Boolean Expression: $Y = \bar{A}$

- Truth Table:

A	Y
0	1
1	0

Digital IC Families

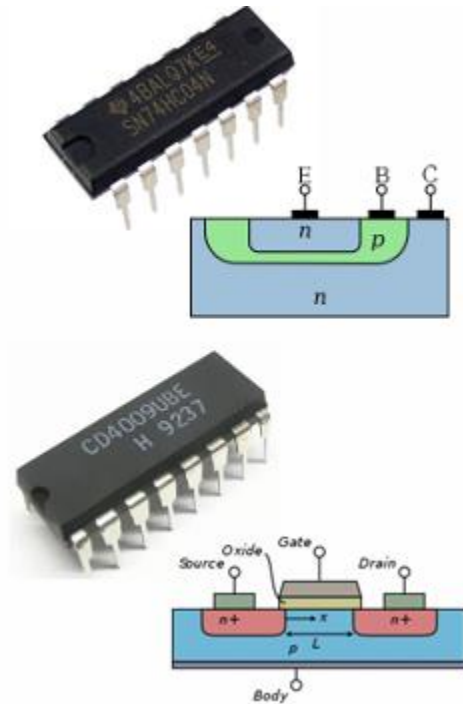
TTL (Transistor-Transistor Logic)

- Series 74xx
- Operating voltage: 5V
- Typical current: 10-20mA
- Speed: Medium to fast
- Noise immunity: Good

CMOS (Complementary Metal-Oxide-Semiconductor)

- Series 40xx
- Operating voltage: 3-15V

- Very low power consumption
- Excellent noise immunity
- High input impedance



IC Classification by Scale of Integration

1. SSI (Small Scale Integration): <10 gates
2. MSI (Medium Scale Integration): 10-100 gates
3. LSI (Large Scale Integration): 100-1000 gates
4. VLSI (Very Large Scale Integration): >1000 gates
5. ULSI (Ultra Large Scale Integration): >1 million gates

Practical Experiments

1. Logic Gate Tester using Arduino

// Arduino Logic Gate Tester

const int inputA = 2; // First input

const int inputB = 3; // Second input

const int output = 4; // Output pin

```
void setup() {  
    pinMode(inputA, OUTPUT);  
    pinMode(inputB, OUTPUT);  
    pinMode(output, INPUT);  
    Serial.begin(9600);  
}  
  
void testGate() {  
    // Test all combinations  
    for(int a = 0; a <= 1; a++) {  
        for(int b = 0; b <= 1; b++) {  
            digitalWrite(inputA, a);  
            digitalWrite(inputB, b);  
            delay(100);  
  
            int result = digitalRead(output);  
            Serial.print("A=");  
            Serial.print(a);  
            Serial.print(" B=");  
            Serial.print(b);  
            Serial.print(" Output=");  
            Serial.println(result);  
        }  
    }  
}
```

```
void loop() {  
    testGate();  
    delay(2000);  
}
```

```
}
```

2. Digital Pulse Generator

```
// Arduino Digital Pulse Generator

const int outputPin = 9;

unsigned long pulseWidth = 1000; // microseconds


void setup() {
    pinMode(outputPin, OUTPUT);
    Serial.begin(9600);
}


void generatePulse() {
    digitalWrite(outputPin, HIGH);
    delayMicroseconds(pulseWidth);
    digitalWrite(outputPin, LOW);
    delayMicroseconds(pulseWidth);
}


void loop() {
    generatePulse();

    // Check for serial input to adjust pulse width
    if(Serial.available()) {
        pulseWidth = Serial.parseInt();
    }
}
```

Part 4

1. Introduction to Digital Logic

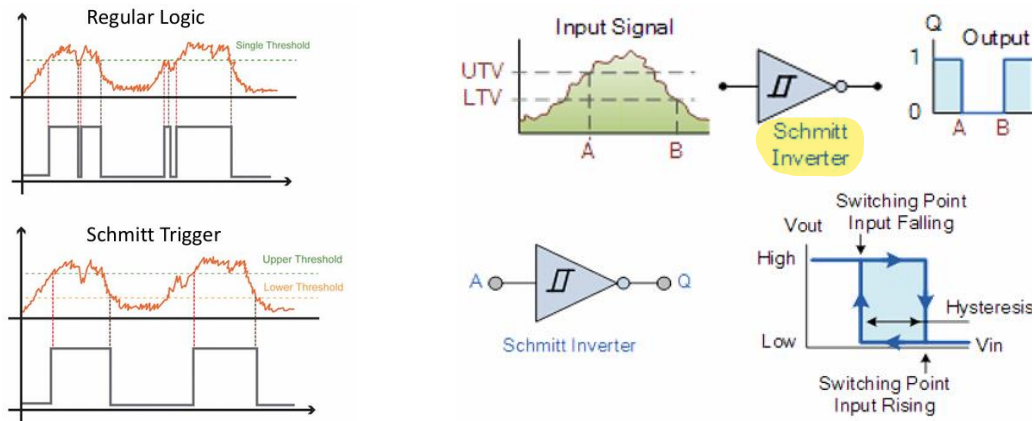
Digital electronics forms the foundation of modern computing and electronic devices. This guide covers essential concepts and practical applications.

Why Study Digital Electronics?

- Foundation for understanding computer architecture
- Essential for embedded systems development
- Basis for IoT device design
- Fundamental to electronic circuit design
- Critical for troubleshooting digital systems

2. Basic Logic Functions

2.1 Schmitt Trigger



[image from page 4 showing Schmitt Trigger symbol and waveforms]

A Schmitt Trigger is a comparator circuit that implements hysteresis by applying positive feedback to the noninverting input of a comparator or differential amplifier.

Key Features:

- Converts noisy input into clean digital output
- Uses two threshold levels (UTV and LTV)

- Provides immunity to noise
- Prevents false triggering

Practical Applications:

1. Signal conditioning
2. Noise reduction in digital circuits
3. Pulse generation
4. Level detection
5. Oscillator circuits

Experiment 1: Building a Schmitt Trigger Noise Filter

```
// Arduino pins
const int INPUT_PIN = A0;
const int OUTPUT_PIN = 13;

// Threshold values
const int UPPER_THRESHOLD = 700;
const int LOWER_THRESHOLD = 300;

bool currentState = false;

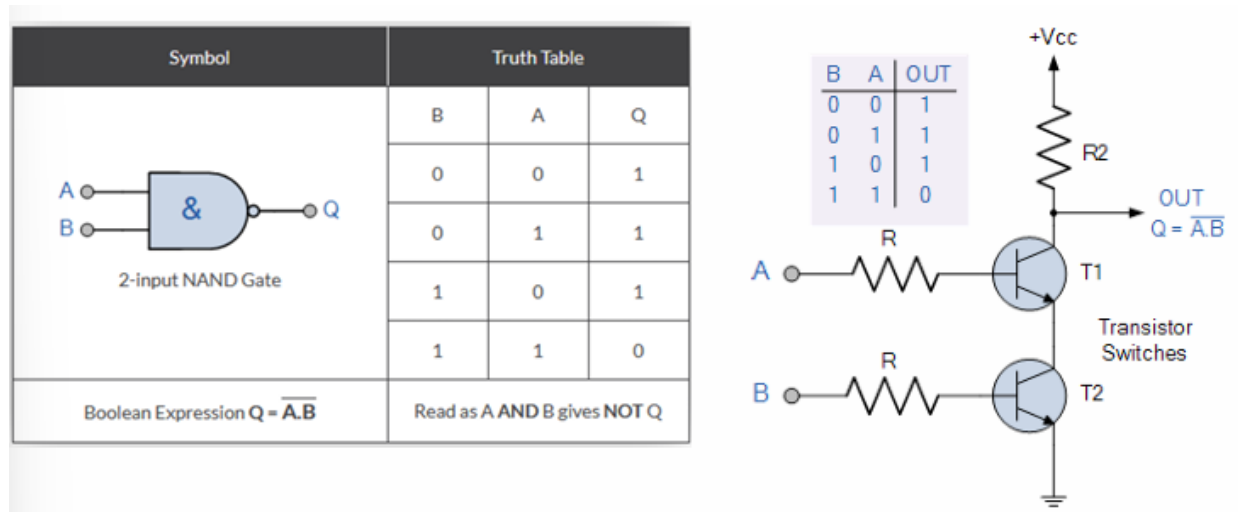
void setup() {
  pinMode(INPUT_PIN, INPUT);
  pinMode(OUTPUT_PIN, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(INPUT_PIN);

  // Implement hysteresis
  if (sensorValue > UPPER_THRESHOLD) {
    currentState = true;
  } else if (sensorValue < LOWER_THRESHOLD) {
    currentState = false;
  }

  digitalWrite(OUTPUT_PIN, currentState);
  Serial.println(sensorValue);
  delay(10);
}
```

2.2 NAND Gate (Sheffer Stroke Function)



[image from page 8 showing NAND gate symbol and truth table]

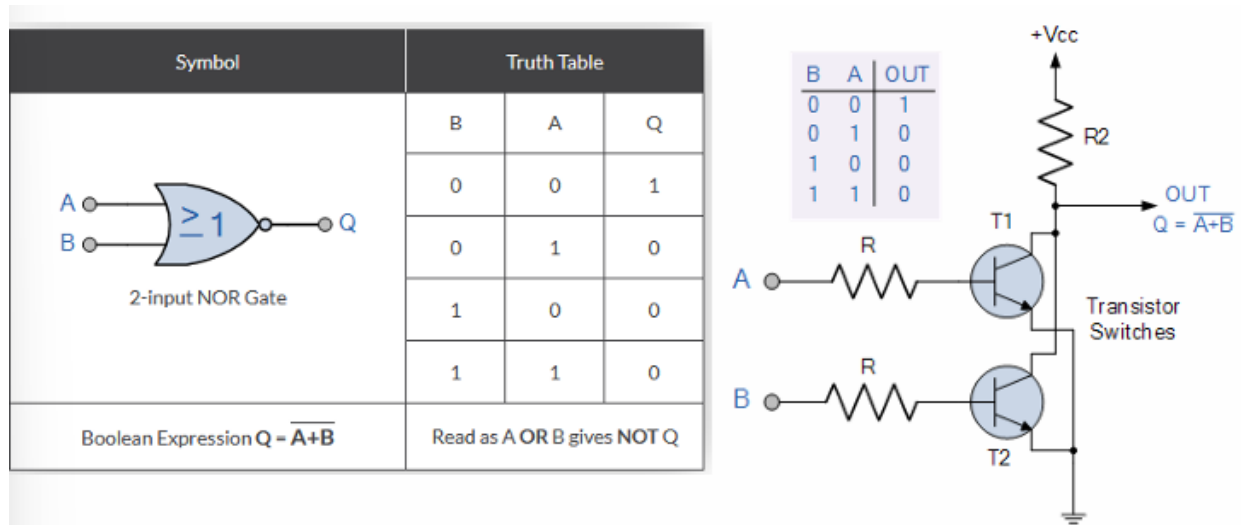
Key Characteristics:

- Universal gate - can implement any other logic function
- Output is LOW only when all inputs are HIGH
- Common in IC form (74LS00, CD4011)

Truth Table:

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

2.3 NOR Gate (Pierce Function)



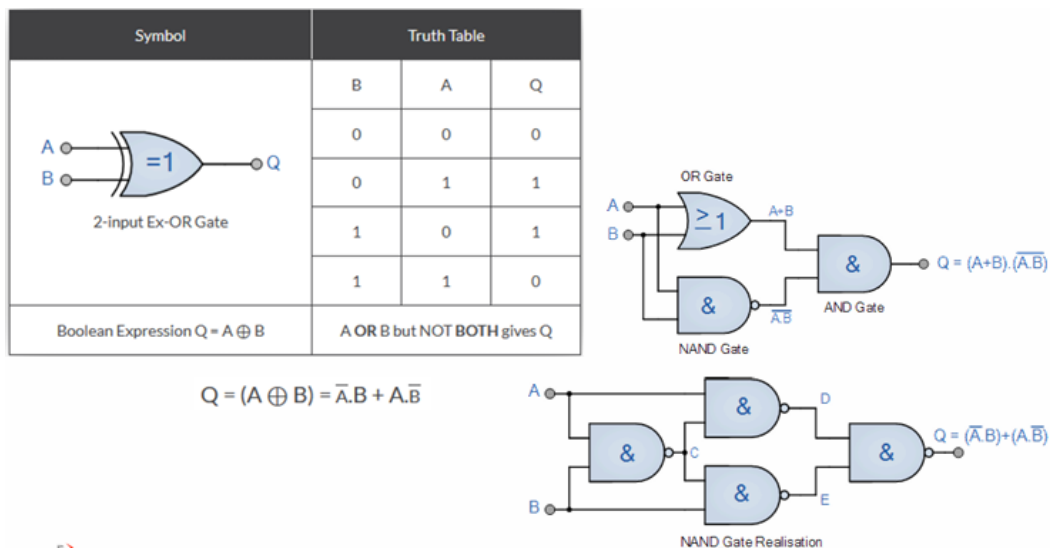
[Insert image from page 12 showing NOR gate symbol and truth table]

Key Characteristics:

- Also a universal gate
- Output is HIGH only when all inputs are LOW
- Available as IC (74LS02, CD4001)

2.4 XOR

“The Logic XOR Function output is only true when its two input terminals are at different logic levels with respect to each other.”



Odd function (A or B but not both)

More than 2 inputs? Mod-2-sum

Ex: $1,0,1 = 2 \bmod 2 = 0$

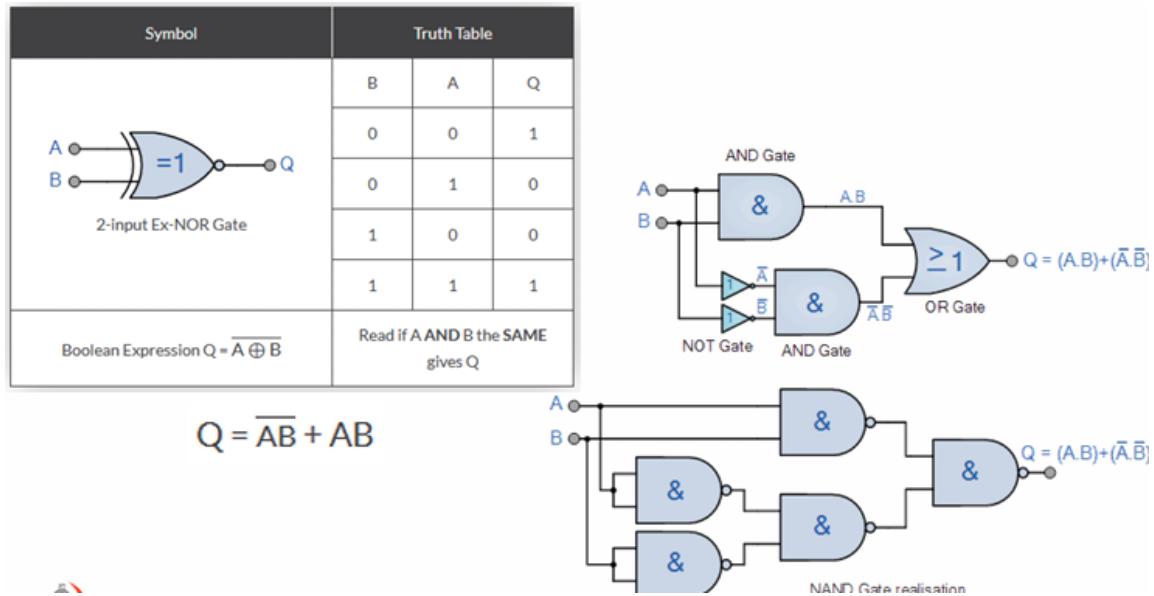
$1,1,1 = 3 \bmod 2 = 1$

$1,0,0 = 1 \bmod 2 = 1$

2.5 XNOR

“The Logic XNOR Function output is only true when both of its inputs are at the same/equal logic level”

“Compliment/Inverse of XOR”



Even function

More than 2 inputs? Inverse of Mod-2-sum

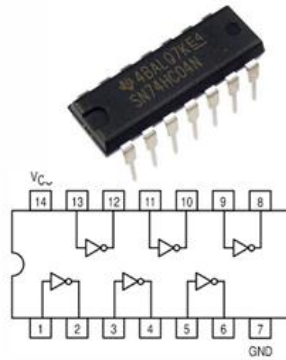
Ex: $1,0,1 = 2 \bmod 2 = 0 \rightarrow 1$

$1,1,1 = 3 \bmod 2 = 1 \rightarrow 0$

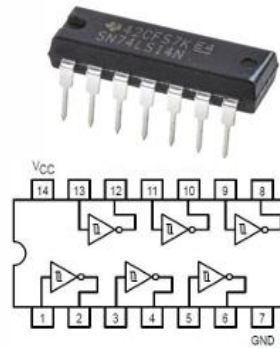
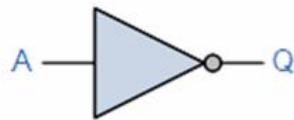
$1,0,0 = 1 \bmod 2 = 1 \rightarrow 0$

$0,0,0 = 0 \bmod 2 = 0 \rightarrow 1$

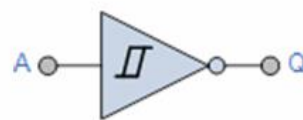
2.6 Schmidt Inverter



Hex Inverter



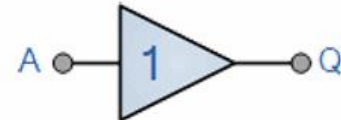
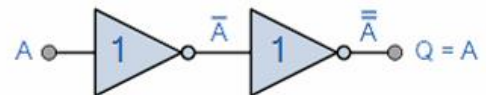
Hex Schmitt Inverter



2.7 Digital Buffer

“Q is true, only when A is true”

It can provide current amplification in a digital circuit to drive output loads, such as relays, solenoids and lamps (called “fan-out”)



2.7.1 Tri-State Buffer

Active-high Tri-state Buffer

Active-high inverting tri-state buffer

Active-low Tri-state Buffer

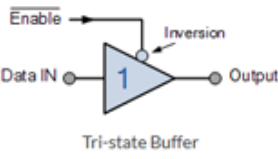
Active-low inverting tri-state buffer

Symbol	Truth Table		
<p>Tri-state Buffer</p>	Enable	IN	OUT
	0	0	Hi-Z
	0	1	Hi-Z
	1	0	0
	1	1	1
Read as Output = Input if Enable is equal to “1”			

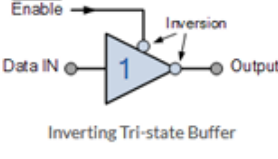
Active-high Tri-state Buffer

Symbol	Truth Table		
<p>Inverting Tri-state Buffer</p>	Enable	IN	OUT
	0	0	Hi-Z
	0	1	Hi-Z
	1	0	1
	1	1	0
Read as Output = Inverted Input if Enable equals “1”			

Active-high inverting tri-state buffer

Symbol	Truth Table		
	Enable	IN	OUT
	0	0	0
	0	1	1
	1	0	Hi-Z
	1	1	Hi-Z
Read as Output = Input if Enable is NOT equal to "1"			

Active-low Tri-state Buffer

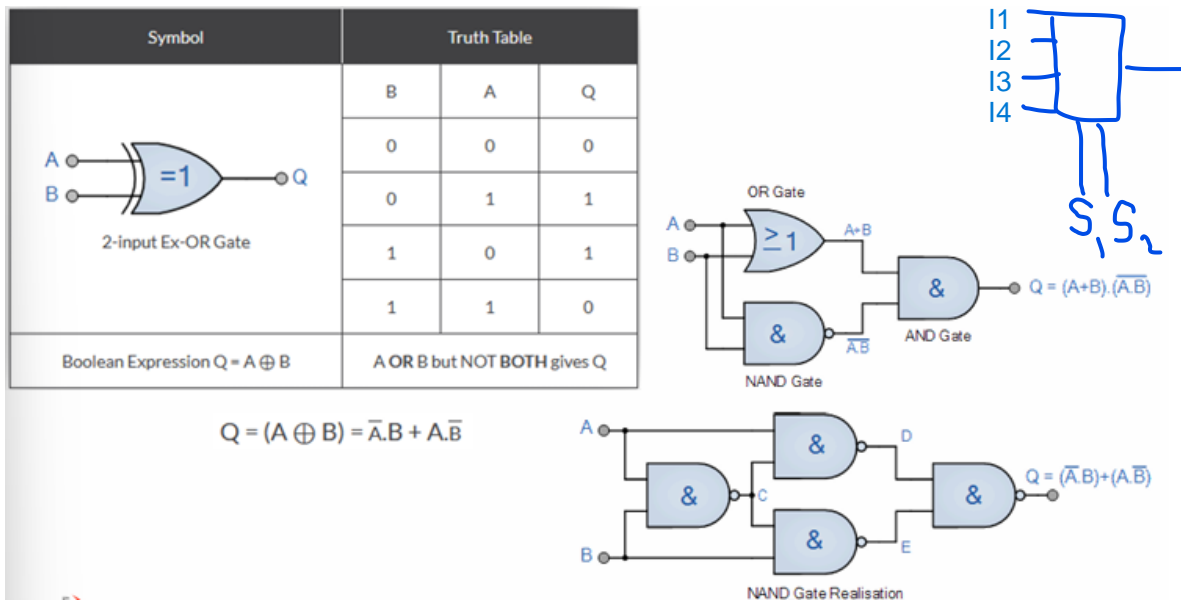
Symbol	Truth Table		
	Enable	IN	OUT
	0	0	1
	0	1	0
	1	0	Hi-Z
	1	1	Hi-Z
Read as Output = Inverted Input if Enable is NOT equal to "1"			

Active-low inverting tri-state buffer

3. Combinational Logic Circuits

3.1 Multiplexer (MUX)

“is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal”



[Insert image from page 16 showing multiplexer diagram]

IF there is two selectors you can add 2^2 input, like wise if $s = 4$ then input is 2^4

Function: Selects one of several input lines and forwards it to a single output line.

Applications:

- Data selection
- Parallel-to-serial conversion
- Data routing
- Memory addressing

Experiment 2: 4-to-1 Multiplexer

```
// Define pins
const int INPUT_PINS[] = {2, 3, 4, 5}; // Data inputs
const int SELECT_PINS[] = {6, 7};      // Selection pins
const int OUTPUT_PIN = 8;              // Output

void setup() {
  // Configure pins
}
```

```

for(int i = 0; i < 4; i++) {
    pinMode(INPUT_PINS[i], INPUT);
}
for(int i = 0; i < 2; i++) {
    pinMode(SELECT_PINS[i], INPUT);
}
pinMode(OUTPUT_PIN, OUTPUT);
}

void loop() {
    // Read selection pins
    int select = digitalRead(SELECT_PINS[0]) |
                (digitalRead(SELECT_PINS[1]) << 1);

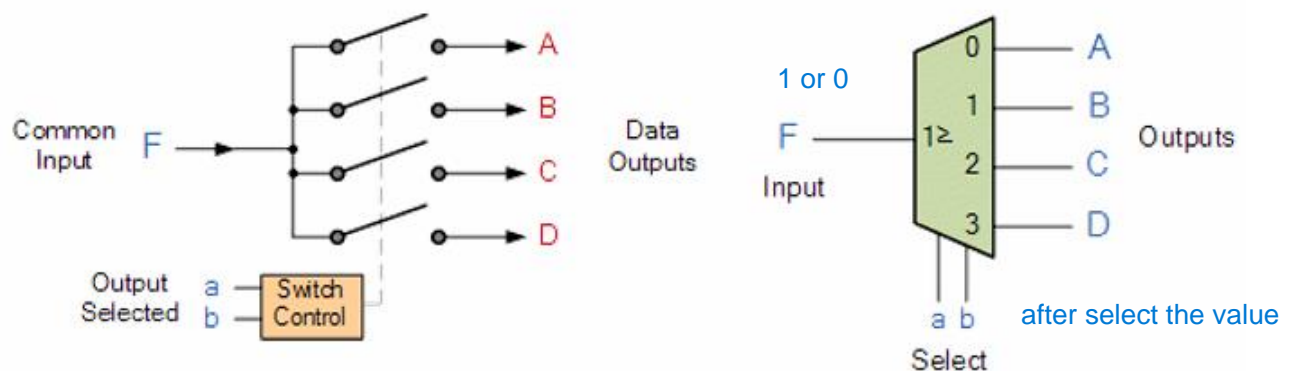
    // Output selected input
    digitalWrite(OUTPUT_PIN, digitalRead(INPUT_PINS[select]));
}

```

3.2 Demultiplexer (Demux)

Exact opposite of the multiplexer

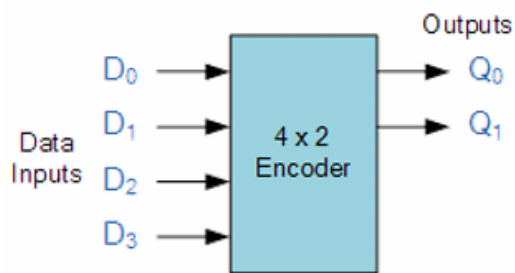
“takes one single input data line and then switches it to any one of a number of individual output lines one at a time”



3.3 Binary Encoder

“takes all its data inputs one at a time and then converts them into a single encoded output”

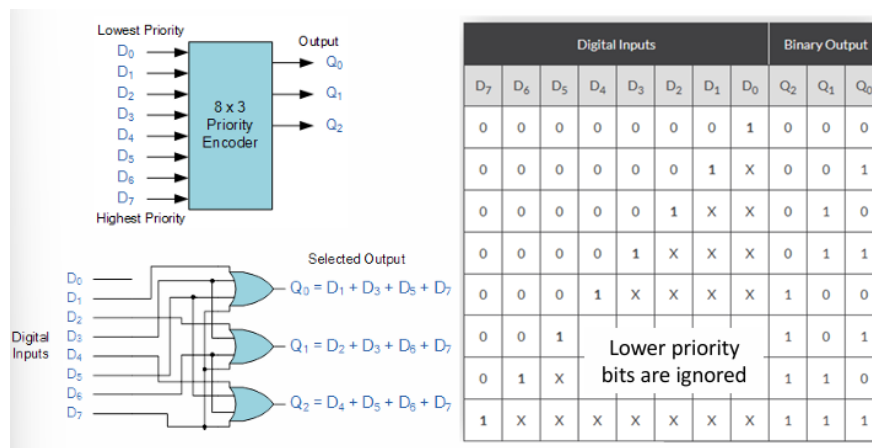
“usually an “n-bit” binary encoder has 2ⁿ input lines and n bit output lines”



Inputs				Outputs	
D ₃	D ₂	D ₁	D ₀	Q ₁	Q ₀
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

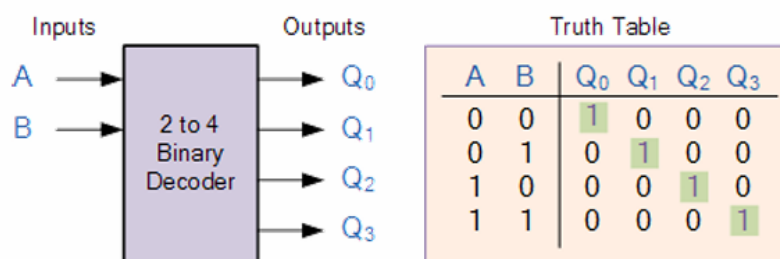
3.3.1 Priority Encoder

If there is more than one input at logic level “1” at the same time, the actual output code would only correspond to the input with the highest designated priority



Application: Keyboard encoder

3.4 Binary Decoder



[Insert image from page 27 showing decoder diagram]

Function: Converts encoded inputs into a set of decoded outputs.

Types:

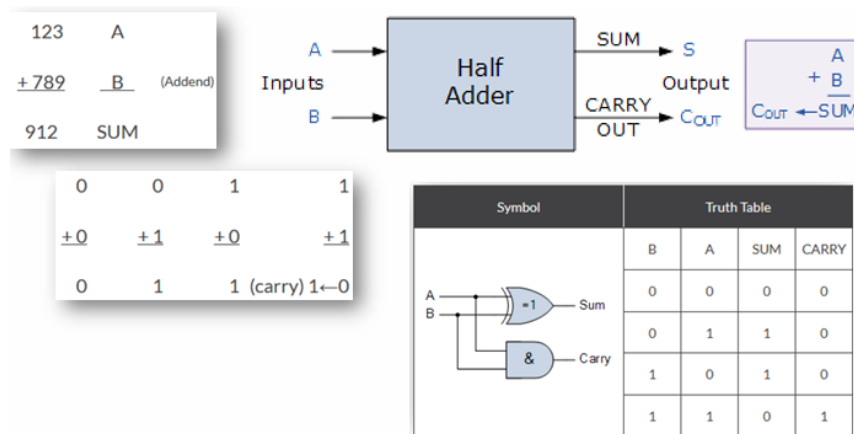
1. Binary-to-Decimal
2. BCD-to-Seven-Segment
3. Address Decoders

Applications:

- Display drivingfbu
- Memory addressing
- Keyboard encoding
- Input/output port selection

3.5 Binary Arithmetic

Half Adder



[image from page 39 showing half adder circuit]

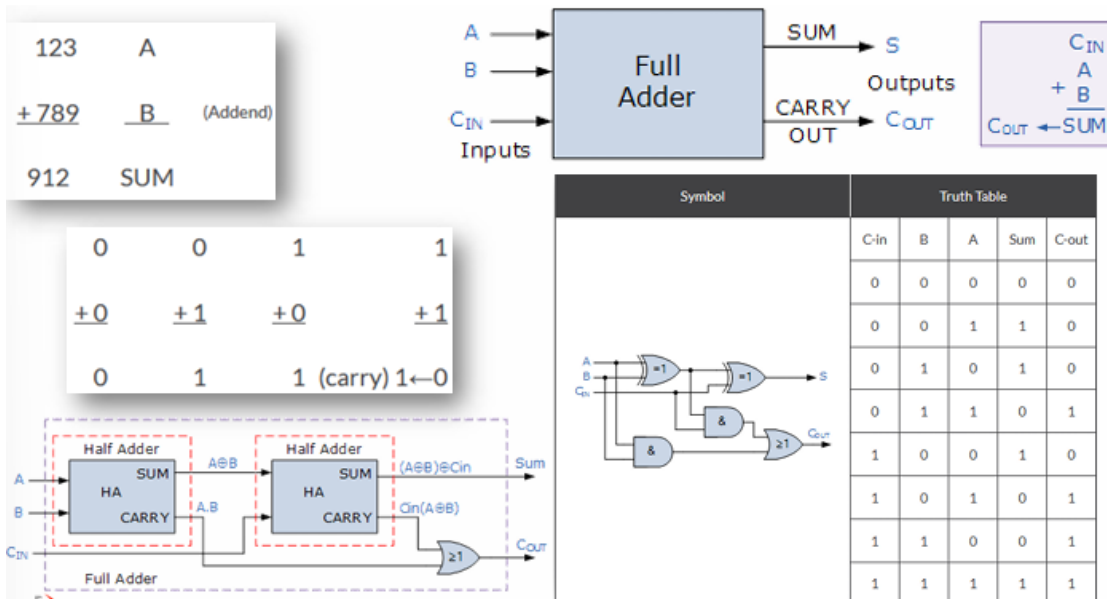
Components:

- XOR gate for sum
- AND gate for carry

Truth Table:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder



[Insert image from page 40 showing full adder circuit]

Additional feature: Handles carry input from previous stage

Truth Table:

Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Experiment 3: 4-bit Binary Adder

```

// Define pins for two 4-bit numbers
const int A_PINS[] = {2, 3, 4, 5}; // First number
const int B_PINS[] = {6, 7, 8, 9}; // Second number
const int SUM_PINS[] = {10, 11, 12, 13}; // Sum output
const int CARRY_PIN = A0; // Carry output

void setup() {

```

```

// Configure inputs
for(int i = 0; i < 4; i++) {
    pinMode(A_PINS[i], INPUT);
    pinMode(B_PINS[i], INPUT);
    pinMode(SUM_PINS[i], OUTPUT);
}
pinMode(CARRY_PIN, OUTPUT);

void loop() {
    int carry = 0;

    // Perform addition bit by bit
    for(int i = 0; i < 4; i++) {
        int a = digitalRead(A_PINS[i]);
        int b = digitalRead(B_PINS[i]);

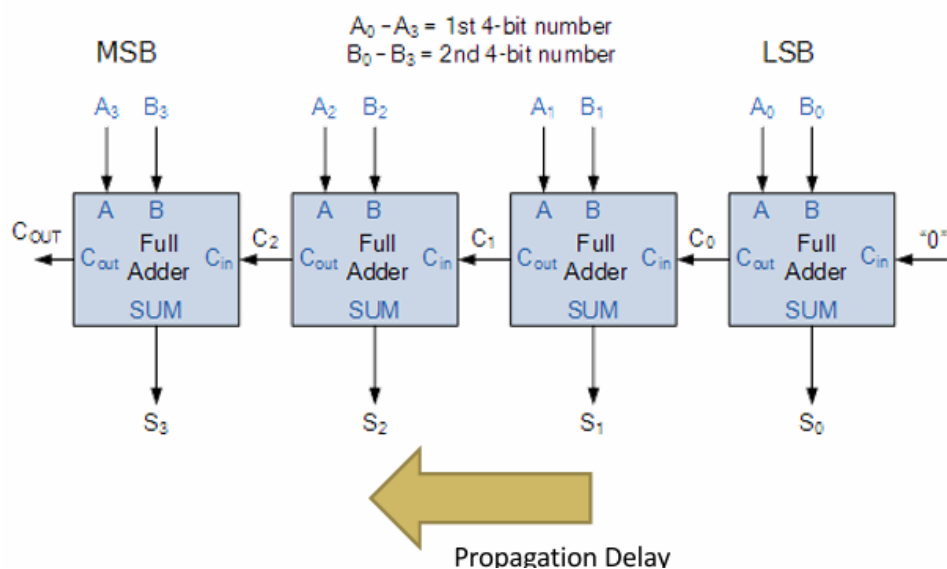
        // Calculate sum and carry
        int sum = a ^ b ^ carry;
        carry = (a & b) | (carry & (a ^ b));

        digitalWrite(SUM_PINS[i], sum);
    }

    digitalWrite(CARRY_PIN, carry);
}

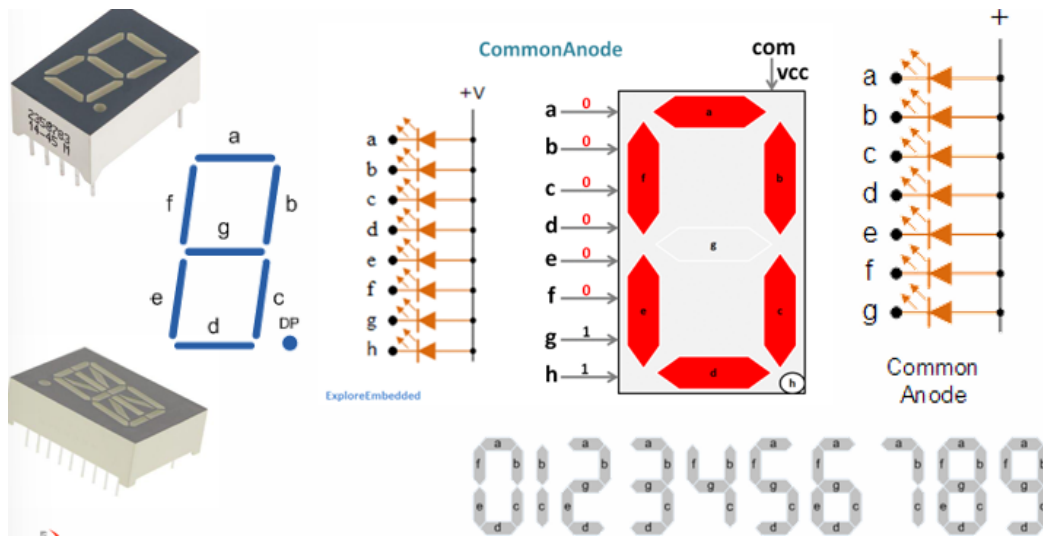
```

Ripple Carry Adder



4. Advanced Topics

4.1 Display Interfacing (SSD)



[Insert image from page 34 showing seven-segment display configuration]

Seven-Segment Display (SSD):

- Common anode vs common cathode
- BCD to seven-segment conversion
- Multiplexed displays
- Current limiting considerations

Experiment 4: Seven-Segment Counter

```
// Seven-segment display pins (A-G)
const int SEGMENT_PINS[] = {2, 3, 4, 5, 6, 7, 8};

//Digit patterns for 0-9
const byte DIGITS[] = {
  0b1111110, // 0
  0b0110000, // 1
  0b1101101, // 2
  0b1111001, // 3
  0b0110011, // 4
  0b1011011, // 5
  0b1011111, // 6
  0b1110000, // 7
  0b1111111, // 8
  0b1111011  // 9
}
```

```

};

void setup() {
  for(int i = 0; i < 7; i++) {
    pinMode(SEGMENT_PINS[i], OUTPUT);
  }
}

void displayDigit(int num) {
  for(int i = 0; i < 7; i++) {
    digitalWrite(SEGMENT_PINS[i],
                 bitRead(DIGITS[num], 6-i));
  }
}

void loop() {
  // Count from 0 to 9
  for(int i = 0; i < 10; i++) {
    displayDigit(i);
    delay(1000);
  }
}

```

5. Troubleshooting and Testing

5.1 Common Issues

1. Noise sensitivity
2. Timing problems
3. Power supply issues
4. Ground bounce
5. Signal integrity

5.2 Testing Methods

1. Logic probes
2. Oscilloscope measurements
3. Pattern generators

4. Logic analyzers
5. IC testers

6. Practice Problems

1. Design a circuit that detects when exactly two inputs out of four are HIGH.
2. Create a frequency divider using flip-flops.
3. Implement a BCD to seven-segment decoder using only NAND gates.
4. Design a 3-bit binary to Gray code converter.