Arduino Uno Pinout Diagram

![[arduino-uno-pinout-1.webp]]

Power Pins

3.3V, 5V,2 ground, Vin(7-12V)

Digital I/O pins

14 digital pins: 0,1: TXD,RXD 2,3: Interrupt 1,2 10-13: SPI pins:(SS, MOSI,MISO,SCK) 13: Includes a built-in LED for quick testing

ANALOG PINS

6 pins: A4,A5 : I2C pins: SDA,SCL

Communication Ports

USB-B PORT: Upload the code and power the board via your computer TX,RX pins for serial communication.

Other Components

- Reset button: Restart your code execution.
- ATmega16U2 Microcontroller: Handles USB-to-serial conversion.
- 16MHz Crystal Oscillator: Ensures timing precision.
- Voltage Regulator: Provides stable power for the entire board.

ICSP Headers?

These headers, designed for in-circuit programming, are available for two chips:

- ATmega328: VCC, MOSI, GND, RESET, SCK, and MISO.
- USB Interface: Used for advanced USB-specific applications.

Arduino Architecture

Arduino’s processor basically uses the Harvard architecture where the program code and program data have separate memory. It consists of two memories- Program memory and the data memory. The code is stored in the flash program memory, whereas the data is stored in the data memory. The Atmega328 has 32 KB of flash memory for storing code (of which 0.5 KB is used for the bootloader), 2 KB of SRAM and 1 KB of EEPROM and operat-es with a clock speed of 16MHz. ![[2.jpg]]

Current Rating

I/O Pins (Digital and Analog)

- **Max current per I/O pin**: **40 mA (absolute max)**
  ⚠ Exceeding this can damage the microcontroller.
  □ Recommended safe value: **< 20 mA**
- **Total current for all I/O pins combined**: **~200 mA max**
  (Practically safer to stay below **150 mA** total across all I/O)

###□ **5V Pin**

- **Current depends on power source**:
  - **USB powered**: Max **~500 mA** total (shared by everything on board)
  - **Barrel jack / VIN powered**:
    - Depends on regulator; **~800 mA to 1 A max**, but regulator heats up fast
  - This current powers
    - The ATmega328P chip
    - Any peripherals connected to I/O pins
    - Anything connected to **5V pin**
- ⚠ If you **draw too much from the 5V pin**, voltage may drop or the onboard regulator may overheat/shutdown.

⚡ 3.3V Pin

- Generated by onboard regulator (for things like SD cards, NRF24L01, etc.)
- **Max current**: **~50 mA**
- ⚠ Very limited; don’t power motors or high-draw sensors from here.

□ VIN Pin

- Input pin when supplying power externally (7–12V recommended)
- Goes through onboard **voltage regulator** to drop to 5V
- Max input voltage: **6–20V** (but 7–12V is ideal)
- Over 12V = more heat; under 7V = unstable 5V supply

□ ATmega328P Total Current

- **Max continuous current through Vcc and GND pins**: **~200 mA total** ![[Pasted image 20250429181654.png]]

Pulse Width Modulation

- On **Arduino Uno**, PWM is done using `Timer0`, `Timer1`, and `Timer2`.
- PWM frequency (by default):
  - Pins 5 and 6: ~980 Hz
  - Pins 3, 9, 10, 11: ~490 Hz You can **change PWM frequency** using hardware timers (e.g., with `TimerOne` library) if needed for motors or servos.

⚠ Tip:

PWM is **not true analog** — it's a digital signal with varying pulse width. It works well for LEDs, motors, and simulating analog output, but **not for analog sensors**.

Simple Code to change brightness of LED

```
void setup() {
// Set pin 9 as an output
pinMode(9, OUTPUT);
}
void loop() {
// Gradually increase the brightness of the LED
for (int i = 0; i < 255; i++) {
analogWrite(9, i);
delay(10);
}
// Gradually decrease the brightness of the LED
for (int i = 255; i > 0; i--) {
analogWrite(9, i);
delay(10);
}
}
```

Serial Communication

Baud rate is how fast data is sent over serial — measured in **bits per second** in Arduino ![[Pasted image 20250429184659.png]]

analogReference() Function

Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are -

- **DEFAULT** − The default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
- **INTERNAL** − An built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (not available on the Arduino Mega)
- **INTERNAL1V1** − A built-in 1.1V reference (Arduino Mega only)
- **INTERNAL2V56** − A built-in 2.56V reference (Arduino Mega only)
- **EXTERNAL** − The voltage applied to the AREF pin (0 to 5V only) is used as the reference

How `analogRead()` Works in Arduino

`analogRead(pin)` is used to **read analog voltage** (0–5V on most Arduinos) and convert it to a **digital number** using an **ADC (Analog-to-Digital Converter)**. Returns an int value of analog voltage value.

□ How Arduino Handles Interrupts

1. **Interrupt is Triggered**

When a certain condition is met (e.g. a pin goes HIGH or LOW), the interrupt is **triggered**.

2. **Arduino Pauses Current Code**

The microcontroller **pauses the main `loop()` or any current function**.

3. **ISR is Executed**

The **Interrupt Service Routine (ISR)** — a special function you wrote — is called **immediately**. Example:

```
void myISR() {
  // This runs when the interrupt occurs
  digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
}
```

4. **Return to Normal**

Once the ISR finishes, Arduino **returns** to what it was doing before.

Rules for ISR Functions

- **Keep it short** — ISRs block other interrupts.
- **No `delay()`, `Serial.print()`, or long calculations**.
- Use **volatile** variables if you communicate between ISR and `loop()`

□ Interrupt Priority in ATmega328P (used by Arduino Uno):

Interrupt Type	Priority (higher up = higher priority)
Reset	Highest (not user-controllable)
External INT0 (pin 2)	High
External INT1 (pin 3)	Medium
Timer/Counter, ADC	Lower priorities
Serial (USART Rx)	Lower

```
const int buttonPin = 2;
const int ledPin = 13;
void setup() {
pinMode(buttonPin, INPUT_PULLUP);
pinMode(ledPin, OUTPUT);
// Attach the ISR to the interrupt
attachInterrupt(digitalPinToInterrupt(buttonPin), handleButton,
FALLING);
}
void loop() {
// Do something else here
}
// ISR function
void handleButton() {
digitalWrite(ledPin, !digitalRead(ledPin));
}
```

□ Overview of Sensors

Sensors convert physical quantities (like temperature, light, motion) into electrical signals.

- **Analog sensors**: Give a continuous voltage (e.g., 0–5V), mapped via `analogRead()`.
- **Digital sensors**: Output HIGH (1) or LOW (0), read using `digitalRead()`

□ Analog vs Digital Sensors

Feature	Analog Sensor	Digital Sensor
Output	Continuous voltage (0–5V)	Binary (HIGH/LOW)
Read Function	`analogRead(pin)`	`digitalRead(pin)`
Resolution	Depends on ADC (e.g. 10-bit = 1024 values)	Just 0 or 1
Example	LM35, LDR	PIR, magnetic switch

□ Temperature Sensor (e.g., LM35)

- Output: Analog voltage proportional to temperature.
- Equation: ![[Pasted image 20250429214434.png]] (e.g., 250mV → 25°C)
- Code: `analogRead()` + mapping function.

□ Humidity Sensor (e.g., DHT11/DHT22)

- Output: Digital signal using a custom 1-wire protocol.
- Returns: Humidity and Temperature.
- Equation: Built into library (e.g., `dht.readHumidity()`).
- Use the `DHT` library in Arduino IDE.

□ Motion Sensor (e.g., PIR Sensor)

- Detects **IR radiation** from moving objects (humans).
- Output: Digital HIGH when motion is detected.
- No complex equation — just `digitalRead()`.

□ Light Sensor (e.g., LDR - Light Dependent Resistor)

- Resistance decreases with light.
- Usually forms a voltage divider.
- Equation: ![[Pasted image 20250429214515.png]] where R1R_1R1 is the LDR or a fixed resistor.
- Use `analogRead()` to get brightness level.

□ Magnetic Field Sensor (e.g., Hall Effect Sensor)

- Detects magnetic field presence.
- Output: Digital HIGH/LOW or analog voltage (in linear sensors).
- Equation:
  ![[Pasted image 20250429214538.png]] where BBB = magnetic field strength, kkk = sensitivity.
- Use `analogRead()` or `digitalRead()`.

□ pH Sensor

- Outputs analog voltage corresponding to pH.
- Typically: ![[Pasted image 20250429214608.png]] (calibrated experimentally).
- Requires analog input + smoothing/filtering.

□ Gas Sensor (e.g., MQ Series)

- Resistance changes based on gas concentration.
- Output: Analog voltage.
- Equation (simplified): ![[Pasted image 20250429214635.png]] where RsR_sRs = sensor resistance in gas, R0R_0R0 = resistance in clean air.
- Requires calibration curves from the datasheet.

⚙ Interfacing of Actuators with Arduino

1. □ Relay

- Electromechanical switch.
- Use `digitalWrite()` to control.
- Can switch high-voltage AC loads.

2. ⚡ DC Motor

- Controlled via **transistor/MOSFET + external power**.
- For speed control: use **PWM** (`analogWrite()`).
- Direction control: H-bridge (e.g., L298N).

Equation for speed: ![[Pasted image 20250429214711.png]]

3. □ Servo Motor

- Controlled via PWM signal (but not `analogWrite()` — use `Servo` library).
- Pulse width determines angle:
    - ~1ms = 0°, ~1.5ms = 90°, ~2ms = 180° **Equation** (approximate): ![[Pasted image 20250429214747.png]]

□ Arduino Programming Basics

Every Arduino sketch consists of two primary functions:

□ Interfacing an LED

Objective: Blink an LED connected to pin 13. **Wiring**:

- Connect the **anode** (long leg) of the LED to **digital pin 13** through a **220Ω resistor**.
- Connect the **cathode** (short leg) to **GND**.

```
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH); // Turn LED on
  delay(500);             // Wait 500ms
  digitalWrite(13, LOW);  // Turn LED off
  delay(500);             // Wait 500ms
}
```

□ Interfacing a Push Button

Objective: Turn on an LED when the button is pressed. **Wiring**:

- Connect one side of the **push button** to **digital pin 2**.
- Connect the other side to **GND**.
- Enable the internal pull-up resistor in the code. **Code**:

```
#define BUTTON_PIN 2
#define LED_PIN 13
void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP); // Enable internal pull-up resistor
  pinMode(LED_PIN, OUTPUT);
}
void loop() {
  if (digitalRead(BUTTON_PIN) == LOW) { // Button pressed
    digitalWrite(LED_PIN, HIGH);
  } else {
    digitalWrite(LED_PIN, LOW);
  }
}
```

Interfacing a Buzzer

- **Working**: Converts electrical signal into sound using piezoelectric material.
- **Types**:
    - **Active** buzzer: sounds with just `HIGH` signal.
    - **Passive** buzzer: requires PWM or tone signal.
- **Control**: On/off or tones via `tone(pin, frequency)`. **Code**:

```
#define BUZZER_PIN 9
#define BUTTON_PIN 2
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}
void loop() {
  if (digitalRead(BUTTON_PIN) == LOW) { // Button pressed
    digitalWrite(BUZZER_PIN, HIGH);
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }
}
```

□ Interfacing a 16x2 LCD (I2C)

Objective: Display "Hello, Arduino!" on the LCD. **Wiring**:

- Connect the **SDA** and **SCL** pins of the LCD to the corresponding pins on the Arduino (A4 and A5 on Uno).
- Connect **VCC** to **5V** and **GND** to **GND**.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); // Adjust the address if necessary

void setup() {
  lcd.begin();
  lcd.backlight();
  lcd.print("Hello, Arduino!");
}

void loop() {
  // Nothing to do here
}
```

Interfacing a Seven-Segment Display (Common Cathode)

Objective: Display digits 0 to 9 sequentially. **Wiring**:

- Connect segments **a** to **g** to Arduino digital pins 2 to 8 through **220Ω resistors**.
- Connect the **common cathode** to **GND**.

Code:

```
int segmentPins[] = {2, 3, 4, 5, 6, 7, 8};
byte digits[10][7] = {
  {1,1,1,1,1,1,0}, // 0
  {0,1,1,0,0,0,0}, // 1
  {1,1,0,1,1,0,1}, // 2
  {1,1,1,1,0,0,1}, // 3
  {0,1,1,0,0,1,1}, // 4
  {1,0,1,1,0,1,1}, // 5
  {1,0,1,1,1,1,1}, // 6
  {1,1,1,0,0,0,0}, // 7
  {1,1,1,1,1,1,1}, // 8
  {1,1,1,1,0,1,1}  // 9
};

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

void loop() {
  for (int num = 0; num < 10; num++) {
    for (int seg = 0; seg < 7; seg++) {
      digitalWrite(segmentPins[seg], digits[num][seg]);
    }
    delay(1000); // Wait 1 second
  }
}
```

Interfacing a Rotary Encoder

- **Working**: Detects the rotation direction and amount. Outputs two square wave signals (A & B). Sequence of pulses determines rotation direction.
- **Button press**: Many encoders include a push button (SW).
- **Used for**: Volume controls, menu navigation, motor speed input. Objective: Read the rotation direction and count steps. Wiring: Connect CLK to digital pin 2. Connect DT to digital pin 3. Connect SW (button) to digital pin 4 (optional). Connect VCC to 5V and GND to GND.

Code:

```
#define CLK 2
#define DT 3

int counter = 0;
int currentStateCLK;
int lastStateCLK;

void setup() {
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  Serial.begin(9600);
  lastStateCLK = digitalRead(CLK);
}

void loop() {
  currentStateCLK = digitalRead(CLK);
  if (currentStateCLK != lastStateCLK) {
    if (digitalRead(DT) != currentStateCLK) {
      counter++;
    } else {
      counter--;
    }
    Serial.print("Counter: ");
    Serial.println(counter);
  }
  lastStateCLK = currentStateCLK;
}
```