

**NDSU**

NORTH DAKOTA STATE UNIVERSITY

TUTORIAL

---

# Arduino Uno Simulator using Wokwi

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	What is Arduino Uno? . . . . .	2
1.2	Why Simulate? . . . . .	2
1.3	Overview of Wokwi . . . . .	2
<b>2</b>	<b>Wokwi Setup</b>	<b>3</b>
<b>3</b>	<b>Arduino Uno Board</b>	<b>4</b>
3.1	Power . . . . .	4
3.2	Reset . . . . .	4
3.3	On-board LED . . . . .	4
3.4	Microcontroller . . . . .	5
3.5	Analog and Power Pins . . . . .	5
3.6	Digital Pins . . . . .	6
<b>4</b>	<b>Language and Sketches</b>	<b>6</b>
4.1	The Language . . . . .	6
4.2	The Sketch . . . . .	6
<b>5</b>	<b>Using the Serial Monitor</b>	<b>7</b>
5.1	Printing with Serial.print() and Serial.println() . . . . .	7
5.2	Printing Variables . . . . .	7
5.3	Reading Input using Serial.read() . . . . .	8
<b>6</b>	<b>Using LEDs</b>	<b>8</b>
6.1	Blink on-board LED . . . . .	8
6.2	Blink external LED . . . . .	9
<b>7</b>	<b>IF-ELSE &amp; Switch Constructs</b>	<b>9</b>
7.1	Using Push Button and External LED . . . . .	9
7.2	Toggling LEDs based on User Choice . . . . .	10
7.3	Temperature Status with LEDs . . . . .	10
7.4	LED Toogling using Switch . . . . .	11

# 1 Introduction

## 1.1 What is Arduino Uno?

The Arduino Uno is one of the most widely used microcontroller development boards. It is built around the ATmega328P microcontroller, a low-power 8-bit chip that makes it easy for beginners and professionals to create electronic projects. Key features of the Arduino Uno include:

- **14 Digital Input/Output Pins** → Can be used to read digital signals (e.g., button press) or control devices (e.g., turning an LED ON/OFF).
- **6 Analog Input Pins** → Allow the board to read continuous signals, such as voltage from sensors (e.g., temperature or light intensity).
- **PWM (Pulse Width Modulation)** → Some digital pins can generate analog-like signals, useful for dimming LEDs or controlling motor speed.
- **USB Connectivity** → Used for programming the board and providing power.

The Arduino Uno is popular because it is open-source, affordable, and beginner-friendly, making it an excellent choice for learning embedded systems and hardware programming.

## 1.2 Why Simulate?

In traditional labs, you would need physical boards, sensors, wires, and other hardware. While this is important for real-world applications, it can be challenging for beginners due to wiring mistakes, hardware costs, or limited availability. By using a simulator, we can:

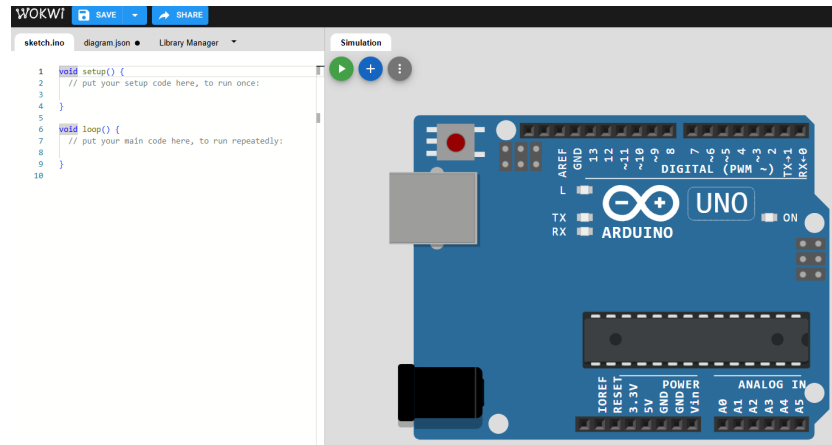
- **Work Safely** → No risk of burning components or damaging hardware.
- **Save Time and Cost** → No need to purchase parts; everything is virtual.
- **Debug Easily** → Visual tools help track signals, values, and errors in real time.
- **Experiment Freely** → Try ideas without worrying about breaking the board.

Simulation allows students to focus on learning logic, coding, and system design before handling real-world electronics.

## 1.3 Overview of Wokwi

Wokwi is an online Arduino simulator that works directly in your web browser. No installation or setup is required. Key advantages of Wokwi:

- **Free and Online** → Runs on any computer with internet access.
- **Arduino Support** → Fully supports Arduino Uno, Nano, Mega, and other boards.



- **Wide Range of Components** → LEDs, buttons, sensors, motors, displays, and more.
- **Real-Time Simulation** → Code runs instantly, and you can interact with components (e.g., pressing buttons, adjusting potentiometers).
- **Serial Monitor Support** → Just like the real Arduino IDE, you can see program output in real time.

With Wokwi, you can practice coding in C for Arduino, build circuits, and simulate projects without any physical hardware, making it ideal for classroom teaching and learning.

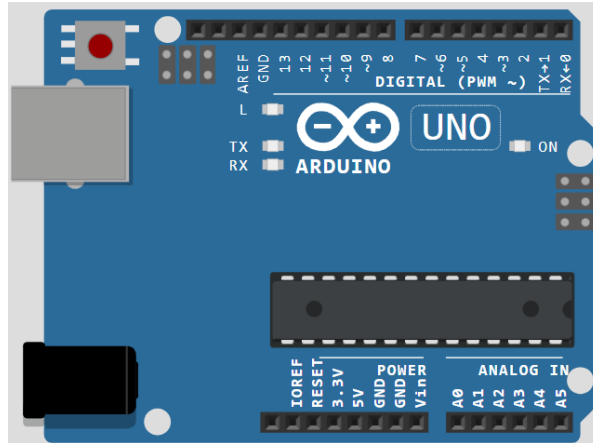
## 2 Wokwi Setup

Access the Wokwi website by searching "Wokwi" on Google, or by navigating directly to the following link ([Wokwi](https://wokwi.com)). You should create an account to save your project. Wokwi allows you to create an account easily using your Google email or an equivalent service.

- Once on the main page and logging in, select Arduino (Uno, Mega, Nano).
- From Start From Scratch section on the main page, select Arduino Uno.
- The following environment will open up. The left half is the part where you will enter your code and on the right hand side is the Arduino board in the circuit drawing area.
- The green play button is the "Start the Simulation" button. Clicking it will start the simulation.
- The blue plus button allows you to select and add components to the circuit area.
- The grey 3 dotted button allows you to change the appearance of the circuit drawing area.

### 3 Arduino Uno Board

In the Wokwi simulator, the Arduino Uno board looks exactly like the physical version, and all its components work the same way during simulation. Features available on the board are detailed below.



#### 3.1 Power

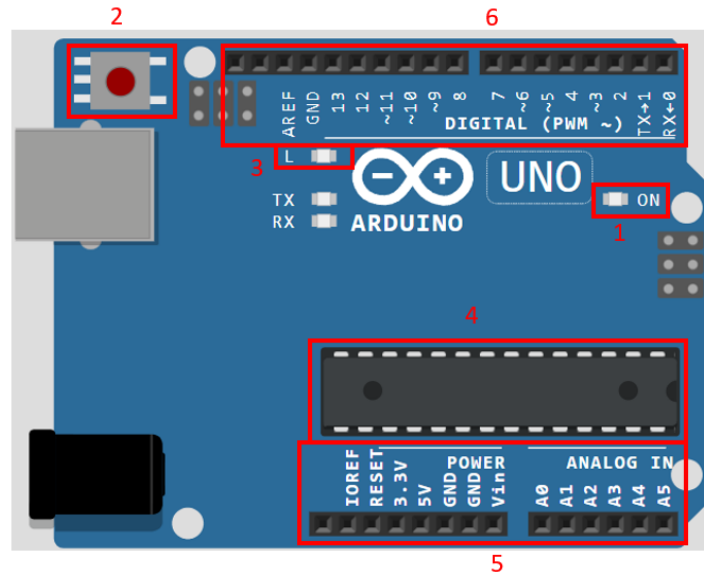
In Wokwi simulation, the board is automatically powered ON as soon as you click the green “Start Simulation” button. You don’t need to connect power manually. The power indicator is shown in red box 1 in the figure below. If the LED is glowing, it means the board is receiving power and your code is running.

#### 3.2 Reset

The Reset button on the Arduino Uno is located near the USB connector at the top left of the board as shown in red box 2 in the below figure. Its purposes are 1) Restarts the microcontroller (ATmega328P), 2) Stops the running program and begins executing it again from the start, and 3) Useful for debugging, testing, or restarting your circuit without stopping the whole simulation. You can click the Reset button directly with your mouse. The program restarts instantly, just as it would on the real board. The ON power LED stays lit (showing the board still has power), but your code runs from the beginning.

#### 3.3 On-board LED

The Arduino Uno board has a built-in LED as shown in red box 3 in the below figure. It’s a single LED that is labeled with the letter "L" on the board. This LED is connected to digital pin 13 in most Arduino boards.



### 3.4 Microcontroller

The microcontroller on the Arduino Uno is the ATmega328P, which serves as the "brain" or "heart" of the board. Its primary purpose is to execute the program that you upload to it. In the Wokwi simulator, it functions exactly as it would on the physical board. Its purpose is to serve as the central processing unit, executing the code you write to control the board's behavior. The Wokwi simulator accurately models this behavior, allowing you to see how your program affects the microcontroller and any connected components. The microcontroller is shown in red box 4.

### 3.5 Analog and Power Pins

The Analog and Power pins are two crucial groups of pins on the Arduino Uno that manage how the board interacts with real-world signals and how it receives or provides electrical energy. Both are modeled accurately in the Wokwi simulator and are shown in red box 5.

The Arduino Uno board has six Analog Input pins, labeled A0 through A5. These pins are found near the Power header on the bottom right of the board. Their primary role is to measure continuous signals from analog sensors, such as reading the varying voltage output from a temperature sensor or a potentiometer. They are connected to an internal Analog-to-Digital Converter (ADC) inside the ATmega328P microcontroller. The ADC takes the analog voltage (which can be any value between 0V and 5V) and converts it into a digital number ranging from 0 to 1023 (a 10-bit resolution). Although labeled "Analog In," these pins can also be configured and used as General Purpose Digital Input/Output (GPIO) pins if you need extra digital pins in your project.

## 3.6 Digital Pins

The Digital Input/Output (I/O) Pins on the Arduino Uno board (Pins 0 to 13) are the primary way the microcontroller interacts with external components using a binary, or two-state, logic system. The Arduino Uno has 14 digital I/O pins in total, numbered D0 to D13. These pins function as General-Purpose Input/Output (GPIO), meaning they can be configured to:

- **Input Mode:** Used to read digital signals (a value of either HIGH or LOW) from a device, such as reading whether a button has been pressed or a sensor has crossed a threshold. HIGH represents the board's operating voltage, which is 5V on the Uno. LOW represents 0V (Ground).
- **Output Mode:** Used to control devices by setting the pin voltage to either HIGH (5V) or LOW (0V), such as turning an LED ON/OFF or activating a relay.

## 4 Language and Sketches

### 4.1 The Language

The programming language used for Arduino, and thus within the Wokwi simulator, is based on C++ but is simplified by the Wiring framework. This language is specifically designed for ease of use in electronics and embedded programming. It abstracts complex hardware register manipulation into simple, clear functions like `pinMode()`, `digitalWrite()`, and `analogRead()`.

### 4.2 The Sketch

An Arduino program is referred to as a sketch. Every sketch is structured around two mandatory functions:

- **`void setup()`:** This function is the initialization block. it runs only once immediately after the program starts or the board is reset, and its primary purpose is to set up the necessary initial conditions for the Arduino. Within this function, you configure hardware settings such as defining which digital pins will act as inputs or outputs using the `pinMode()` function, starting any necessary serial communication with `Serial.begin()`, and initializing external libraries or variables.
- **`void loop()`:** This function is executed directly after the completion of the `setup()`. This is the main execution block of the sketch. This function runs the program's primary logic continuously and repeatedly until simulation is stopped, allowing the Arduino to perform its ongoing tasks, such as constantly checking the state of sensors with `digitalRead()`, controlling actuators with `digitalWrite()` or `analogWrite()`, and responding to changing conditions in the environment.

## 5 Using the Serial Monitor

The Serial Monitor allows the Arduino Uno to send and receive text data from your computer. This is the first step to debugging and interacting with your programs. Type the following code in the editor and click the Start Simulation button to start the simulation.

### 5.1 Printing with Serial.print() and Serial.println()

#### Program 1

```
1 void setup()
2 {
3   Serial.begin(9600); // Start serial communication at 9600 bits/sec
4   Serial.print("Hello"); // Print text (stays on the same line)
5   Serial.print(" World!"); // Continue on the same line
6   Serial.println("<- This ends with a new line"); // Print and move to
   the next line
7   Serial.print("Exiting..."); // This message is printed on a new line
8 }
9
10 void loop()
11 {
12   // Nothing here, code runs only once in setup()
13 }
```

### 5.2 Printing Variables

#### Program 2

```
1 int counter = 0; // Declare a counter variable
2
3 void setup()
4 {
5   Serial.begin(9600);
6 }
7
8 void loop()
9 {
10  Serial.print("Counter value: ");
11  Serial.println(counter); // Print value and move to next line
12  counter++; // Increase counter by 1
13  delay(1000); // Wait for 1 second before repeating
14 }
```



## 5.3 Reading Input using Serial.read()

### Program 3

```
char command; // Variable to store the received character
2
void setup()
4 {
    Serial.begin(9600);
    Serial.println("Enter a character: ");
6 }
8
void loop()
10 {
    // Check if data is available in Serial Monitor
    if (Serial.available() > 0)
12 {
        command = Serial.read(); // Read one character
14
        // Ignore non-printable characters like newline and carriage return
        if (command != '\n' && command != '\r')
16 {
            Serial.print("You entered: ");
            Serial.println(command);
            Serial.println("Enter a character: ");
20
        }
22 }
24 }
```

## 6 Using LEDs

### 6.1 Blink on-board LED

#### Program 1

```
void setup()
2 {
    pinMode(LED_BUILTIN, OUTPUT); // Set on-board LED as output
4 }
6
void loop()
8 {
    digitalWrite(LED_BUILTIN, HIGH); // LED on
    delay(1000);
10 digitalWrite(LED_BUILTIN, LOW); // LED off
    delay(1000);
12 }
```

## 6.2 Blink external LED

### Program 2

```
1 void setup()
2 {
3   pinMode(5, OUTPUT); // Configure digital pin 5 as output
4   // Connected to external LED
5 }
6
7 void loop()
8 {
9   digitalWrite(5, HIGH); // LED on
10  delay(1000);
11  digitalWrite(5, LOW); // LED off
12  delay(1000);
13 }
```

## 7 IF-ELSE & Switch Constructs

### 7.1 Using Push Button and External LED

#### Program 1

```
1 int buttonPin = 2; // Push button connected to pin 2
2 int ledPin = 5; // LED connected to pin 5
3 int buttonState; // Variable to store button status
4
5 void setup()
6 {
7   pinMode(buttonPin, INPUT_PULLUP); // Use internal pull-up (button active
8   // LOW)
9   pinMode(ledPin, OUTPUT); // LED as output
10 }
11
12 void loop()
13 {
14   buttonState = digitalRead(buttonPin); // Read button
15
16   if (buttonState == LOW) // Button pressed
17   {
18     digitalWrite(ledPin, HIGH); // LED ON
19   }
20   else // Button not pressed
21   {
22     digitalWrite(ledPin, LOW); // LED OFF
23   }
24 }
```

## 7.2 Toggling LEDs based on User Choice

### Program 2

```
1 char choice;
3 void setup()
{
5   pinMode(5, OUTPUT); // Pin 5 output — Red LED
   pinMode(6, OUTPUT); // Pin 6 output — Blue LED
7   pinMode(7, OUTPUT); // Pin 7 output — Green LED
   Serial.begin(9600);
9   Serial.println("Enter your choice: ");
   // Enter 1 to toggle Red, 2 to toggle Blue, and 3 to toggle Green LED
11 }

13 void loop()
{
15   if (Serial.available() > 0)
   {
17     choice = Serial.read();
     if (choice != '\n' && choice != 'r')
19     {
         if (choice == '1') // Toggle RED LED
21         digitalWrite(5, !digitalRead(5)); // Toggle = write opposite of
         current state
         else if (choice == '2') // Toggle Blue LED
23         digitalWrite(6, !digitalRead(6));
         else if (choice == '3') // Toggle Green LED
25         digitalWrite(7, !digitalRead(7));

27         Serial.println("Enter your choice: "); // Prompt again
     }
29 }
}
```

## 7.3 Temperature Status with LEDs

### Program 3

```
int potPin = A0; // Analog pin connected to potentiometer
2 float temperature;

4 void setup()
{
6   pinMode(5, OUTPUT); // RED
   pinMode(6, OUTPUT); // Blue
8   pinMode(7, OUTPUT); // Green
   Serial.begin(9600);
10 }
```

```
12 void loop()
13 {
14     int potValue = analogRead(potPin); // Read potentiometer (0-1023)
15     temperature = map(potValue, 0, 1023, 0, 50); // Map to 0C - 50C
16
17     // Turn off all LEDs first
18     digitalWrite(5, LOW);
19     digitalWrite(6, LOW);
20     digitalWrite(7, LOW);
21
22     // Light LED based on temperature
23     if (temperature < 20.0)
24         digitalWrite(6, HIGH); // Blue - Cold
25     else if (temperature <= 30.0)
26         digitalWrite(7, HIGH); // Green - Comfortable
27     else
28         digitalWrite(5, HIGH); // Red - Hot
29
30     // Print temperature on Serial Monitor
31     Serial.print("Temperature: ");
32     Serial.print(temperature);
33     Serial.println(" C");
34
35     delay(200); // Small delay for stability
36 }
```

## 7.4 LED Toogling using Switch

### Program 4

```
1 char command;
2
3 void setup()
4 {
5     Serial.begin(9600);
6     pinMode(5, OUTPUT); // RED LED
7     pinMode(6, OUTPUT); // Blue LED
8     pinMode(7, OUTPUT); // Green LED
9     Serial.println("Enter command: 1, 2, or 3 to toggle LEDs");
10 }
11
12 void loop()
13 {
14     if (Serial.available() > 0)
15     {
16         command = Serial.read();
17
18         // Ignore newline and carriage return
19         if (command == '\n' || command == '\r')
```

```
21     ;// do nothing, skip this loop iteration
22     else
23     {
24         switch (command)
25         {
26             case '1':
27                 digitalWrite(5, !digitalRead(5)); // Toggle RED
28                 Serial.println("LED1 toggled");
29                 break;
30             case '2':
31                 digitalWrite(6, !digitalRead(6)); // Toggle Blue
32                 Serial.println("LED2 toggled");
33                 break;
34             case '3':
35                 digitalWrite(7, !digitalRead(7)); // Toggle Green
36                 Serial.println("LED3 toggled");
37                 break;
38             default:
39                 Serial.println("Invalid command");
40         }
41     }
42 }
```