NDSU | NORTH DAKOTA STATE UNIVERSITY

Tutorial

Arduino Uno Simulator using Wokwi

Contents

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:

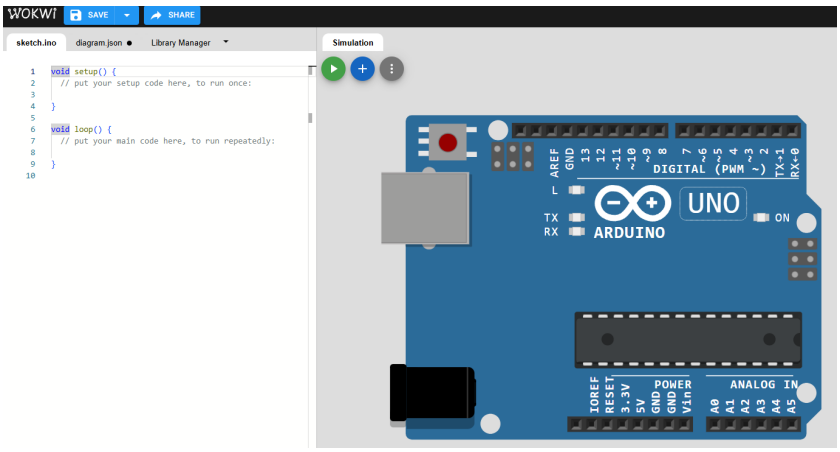
- **Work Safely** → No risk of burning components or damaging hardware.
- **Save Time and Cost** → No need to purchase parts; everything is virtual.
- **Debug Easily** → Visual tools help track signals, values, and errors in real time.
- **Experiment Freely** → Try ideas without worrying about breaking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

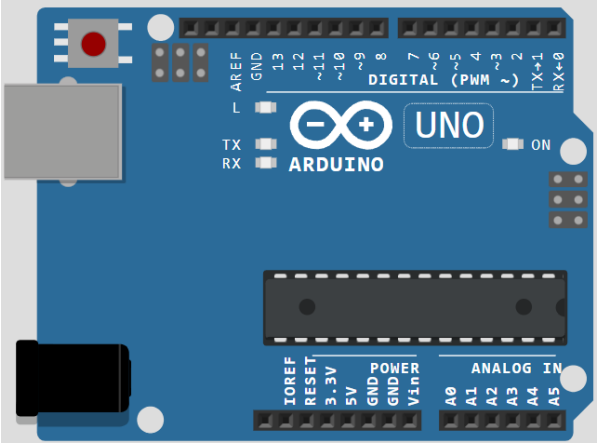
2 Wokwi Setup

Access the Wokwi website by searching "Wokwi" on Google, or by navigating directly to the following link (Wokwi). 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

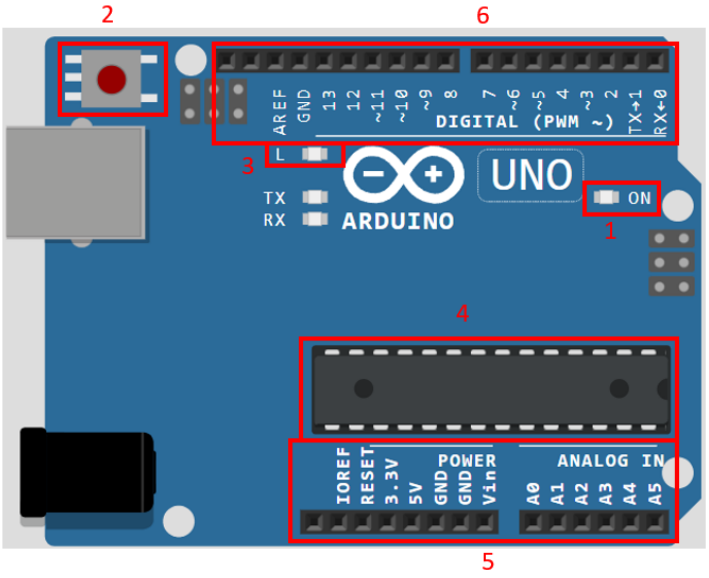
In Wokwi simulation, the board is automatically powered ON as soon as you click the green “Start Simulation” button. You don’t need to connect power manually. The power indicator is shown in red box 1 in the figure below. If the LED is glowing, it means the board is receiving power and your code is running.

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

```
void setup()
{
  Serial.begin(9600); // Start serial communication at 9600 bits/sec
  Serial.print("Hello"); //Print text (stays on the same line)
  Serial.print(" World!"); // Continue on the same line
  Serial.println(" <- This ends with a new line"); // Print and move to
   the next line
  Serial.print("Exiting..."); // This message is printed on a new line
}

void loop()
{
  // Nothing here, code runs only once in setup()
}
```

5.2 Printing Variables

Program 2

```
int counter = 0; // Declare a counter variable

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

void loop()
{
  Serial.print("Counter value: ");
  Serial.println(counter); // Print value and move to next line
  counter++; // Increase counter by 1
  delay(1000); // Wait for 1 second before repeating
}
```

5.3 Reading Input using Serial.read()

Program 3

```
char command; // Variable to store the received character

void setup()
{
  Serial.begin(9600);
  Serial.println("Enter a character: ");
}

void loop()
{
  // Check if data is available in Serial Monitor
  if (Serial.available() > 0)
  {
    command = Serial.read(); // Read one character

    // Ignore non-printable characters like newline and carriage return
    if (command != '\n' && command != '\r')
    {
      Serial.print("You entered: ");
      Serial.println(command);
      Serial.println("Enter a character: ");
    }
  }
}
```

6 Using LEDs

6.1 Blink on-board LED

Program 1

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT); // Set on-board LED as output
}

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

6.2 Blink external LED

Program 2

```
void setup()
{
  pinMode(5, OUTPUT); // Configure digital pin 5 as output
  // Connected to external LED
}

void loop()
{
  digitalWrite(5, HIGH); // LED on
  delay(1000);
  digitalWrite(5, LOW); // LED off
  delay(1000);
}
```

7 IF-ELSE & Switch Constructs

7.1 Using Push Button (Press and Hold) and External LED

Program 1

```
int buttonPin = 2;    // Push button connected to pin 2
int ledPin = 5;       // LED connected to pin 5
int buttonState;      // Variable to store button status

void setup()
{
  pinMode(buttonPin, INPUT_PULLUP); // Use internal pull-up (button active
    LOW)
  pinMode(ledPin, OUTPUT);  // LED as output
}

void loop()
{
  buttonState = digitalRead(buttonPin); // Read button

  if (buttonState == LOW) // Button pressed
  {
    digitalWrite(ledPin, HIGH); // LED ON
  }
  else // Button not pressed
  {
    digitalWrite(ledPin, LOW); // LED OFF
  }
}
```

7.2 Toggling LEDs based on User Choice

Program 2

```
char choice;

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

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

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

7.3 Temperature Status with LEDs

Program 3

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

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

```
12 void loop()
{
14   int potValue = analogRead(potPin);  // Read potentiometer (0-1023)
   temperature = map(potValue, 0, 1023, 0, 50); // Map to 0C - 50C
16
   // Turn off all LEDs first
18   digitalWrite(5, LOW);
   digitalWrite(6, LOW);
20   digitalWrite(7, LOW);

22   // Light LED based on temperature
   if (temperature < 20.0)
24     digitalWrite(6, HIGH);  // Blue - Cold
   else if (temperature <= 30.0)
26     digitalWrite(7, HIGH);  // Green - Comfortable
   else
28     digitalWrite(5, HIGH);  // Red - Hot

30   // Print temperature on Serial Monitor
   Serial.print("Temperature: ");
32   Serial.print(temperature);
   Serial.println(" C");
34
   delay(200); // Small delay for stability
36 }
```

7.4 LED Toogling using Switch

Program 4

```
1 char command;

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

```
      ;// do nothing, skip this loop iteration
    else
    {
      switch(command)
      {
        case '1':
          digitalWrite(5, !digitalRead(5)); // Toggle RED
          Serial.println("LED1 toggled");
          break;
        case '2':
          digitalWrite(6, !digitalRead(6)); // Toggle Blue
          Serial.println("LED2 toggled");
          break;
        case '3':
          digitalWrite(7, !digitalRead(7)); // Toggle Green
          Serial.println("LED3 toggled");
          break;
        default:
          Serial.println("Invalid command");
      }
    }
  }
}
```

7.5 Push Button (Press & Release) and External LED

Program 5

```
int buttonPin = 2;    // Push button connected to pin 2
int ledPin = 5;       // LED connected to pin 5
int buttonState;      // Variable to store button status

void setup()
{
  pinMode(buttonPin, INPUT_PULLUP); // Use internal pull-up (active LOW)
  pinMode(ledPin, OUTPUT);  // LED as output
}

void loop()
{
  buttonState = digitalRead(buttonPin); // Read button
  if (buttonState == LOW) // Check if button is pressed
  {
    digitalWrite(ledPin, HIGH);
    delay(5000);
    digitalWrite(ledPin, LOW);
  }
}
```

8 Loops

8.1 Blink an LED 5 Times

Program 1

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT); // Built-in LED
}

void loop()
{
  for (int i = 1; i <= 5; i++) // Run 5 times
  {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on LED
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW); // Turn off LED
    delay(1000);
  }
  // LED is currently OFF
  while(1); // stay OFF indefinitely
}
```

8.2 LED Brightness Control with PWM

Program 2

```
int ledPin = 9; // PWM-capable pin

void setup()
{
  pinMode(ledPin, OUTPUT); // Built-in LED
}

void loop()
{
  // Fade in (0 -> 255)
  for (int brightness = 0; brightness <= 255; brightness++)
  {
    analogWrite(ledPin, brightness); // Set LED brightness
    delay(10);
  }
  // Fade out (255 -> 0)
  for (int brightness = 255; brightness >= 0; brightness--)
  {
    analogWrite(ledPin, brightness);
    delay(10);
  }
}
```

8.3 Blink LED Until Button Pressed

Program 3

```
void setup()
{
  pinMode(5, OUTPUT); // LED on Pin 5
  pinMode(2, INPUT_PULLUP); // Press Button active LOW
}

void loop()
{
  // Keep blinking LED until button is pressed
  while (digitalRead(buttonPin) == HIGH)
  {
    digitalWrite(5, HIGH);
    delay(300);
    digitalWrite(5, LOW);
    delay(300);
  }
  digitalWrite(5, LOW); // Once button is pressed, Turn off LED
}
```

8.4 Serial Input Validation

Program 4

```
//Keep waiting for a valid input using while
char command;

void setup()
{
  Serial.begin(9600);
  Serial.println("Enter 'Y' to turn ON LED or 'N' to turn it OFF.");
  pinMode(5, OUTPUT);
}

void loop()
{
  // Wait until something is typed
  while (Serial.available() == 0)
  {
    ;// Do nothing, just wait
  }

```

```
  command = Serial.read();

  if (command == 'Y' || command == 'y')
  {
    digitalWrite(5, HIGH);
    Serial.println("LED turned ON");
  }
  else if (command == 'N' || command == 'n')
  {
    digitalWrite(5, LOW);
    Serial.println("LED turned OFF");
  }
  else
    Serial.println("Invalid input. Please enter Y or N.");
}
```

8.5 Five Beeps break

Program 5

```
#define BUZZER 5
// This is a macro. Replaces all occurences of BUZZER with 5

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

void loop()
{
  for (int i = 1; i <= 10; i++)
  {
    tone(BUZZER, 1000, 300); // Beep at 1kHz for 300ms
    delay(500); // silence for 200 ms
    if (i == 5)
      break; // No beeps after 5th beep
  }
  delay(2000); // repeat after 2 seconds
}
```

8.6 LED Sequence with continue

Program 6

```
#define BUZZER 5
// This is a macro. Replaces all occurences of BUZZER with 5

```

```
void setup()
{
  pinMode(BUZZER, OUTPUT);
}

void loop()
{
  for (int i = 1; i <= 10; i++)
  {
    tone(BUZZER, 1000, 300); // Beep at 1kHz for 300ms
    delay(500); // silence for 200 ms
    if (i == 5)
      break; // No beeps after 5th beep
  }
  delay(2000); // repeat after 2 seconds
}
```