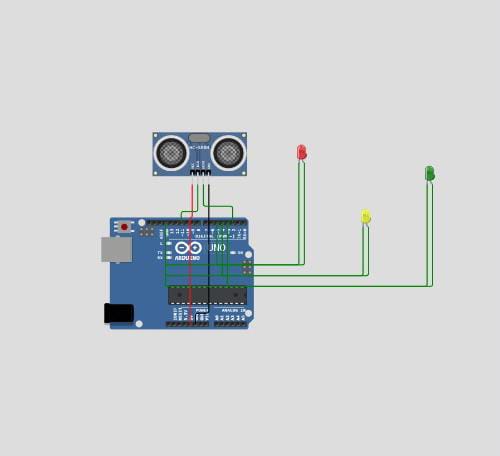
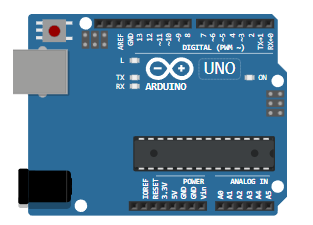
# **PHASE 5** – **TRAFFIC MANAGEMENT SYSTEM**



# Arduino-uno Reference

Arduino Uno is the most popular board in the Arduino family. It is powered by the ATmega328p chip, which has 32K bytes of Flash program memory, 2k bytes of SRAM and 1K bytes of EEPROM.

ONL TX RX  DIGITAL (PWM ~)AREF GND 13 12 ~11 ~10 ~9 8 7 ~6 ~5 4 ~3 2 TX→1 RX←0  POWER ANALOG INIOREF RESET 3.3V 5V GND GND Vin A0 A1 A2 A3 A4 A5  ARDUINOUNO



# **Pin names**[​](https://docs.wokwi.com/parts/wokwi-arduino-uno#pin-names)

Pins 0 to 13 are digital GPIO pins. Pins A0 to A5 double as analog input pins, in addition to being digital GPIO pins.

There are three ground pins: GND.1, which is on top of the board, next to pin 13, and GND.2/GND.3, which are on the bottom.

Pins VIN / 5V are connected to the positive power supply.

Pins 3.3V / IOREF / AREF / RESET are not available in the simulation.

Digital pins 3, 5, 6, 9, 10, and 11 have hardware PWM support.

Some of the digital pins also have additional functions:

|  |  |  |
| --- | --- | --- |
| **Pin** | **Function** | **Signal** |
| 0 | Serial (USART) | RX |
| 1 | Serial (USART) | TX |
| 2 | External interrupt | INT0 |
| 3 | External interrupt | INT1 |
| 10 | SPI | SS (Chip select) |
| 11 | SPI | MOSI |
| 12 | SPI | MISO |
| 13 | SPI | SCLK (Clock) |
| A4 | I2C | SDA (Data) |
| A5 | I2C | SCL (Clock) |
|  |  |  |
|  |  |  |

### On board LEDs[​](https://docs.wokwi.com/parts/wokwi-arduino-uno#on-board-leds)

The board includes four LEDs:

|  |  |
| --- | --- |
| **LED** | **Function** |
| L | Connected to digital pin 13 |
| RX | Serial RX Activity |
| TX | Serial TX Activity |
| ON | Power LED. Always on while the simulation is running |

In general, only the "L" LED can be controlled by the user's code. You can use the LED\_BUILTIN constant to reference it from your code:

**pinMode(LED\_BUILTIN, OUTPUT);  
digitalWrite(LED\_BUILTIN, HIGH);**

See [Blink](https://wokwi.com/projects/344891652101374548) for a complete code example.

# **Attributes**[**​**](https://docs.wokwi.com/parts/wokwi-arduino-uno#attributes)

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Default value** |
| frequency | MCU clock frequency, in hertz. Common values: "8m", "16m", and "20m" \* | "16m" |

\* Many Arduino libraries assume 16 MHz clock frequency. Changing the clock frequency will void your warranty!

# **Simulation features**[​](https://docs.wokwi.com/parts/wokwi-arduino-uno#simulation-features)

The Arduino Uno is simulated using the [AVR8js Library](https://github.com/wokwi/avr8js). The table below summarizes the status of features:

|  |  |  |
| --- | --- | --- |
| **Peripheral** | **Status** | **Notes** |
| Processor | ✔️ |  |
| GPIO | ✔️ | Including External/Pin Change Interrupts |
| 8-bit timers | ✔️ | Timer0, Timer2 |
| 16-bit timer | ✔️ | Timer1 |
| Watchdog Timer | ✔️ | [Usage example](https://wokwi.com/projects/309372800631571009) |
| USART | ✔️ |  |
| SPI | 🟡 | Master mode only |
| I2C | 🟡 | Master mode only |
| EEPROM | ✔️ |  |
| Clock Prescale | ✔️ |  |
| ADC | ✔️ | Used by analogRead() |
| Analog Comparator | ❌ |  |
| GDB Debugging | ✔️ | See the [GDB Debugging Guide](https://docs.wokwi.com/gdb-debugging) |

Legend:  
✔️ Simulated  
🟡 Simulated, but see notes  
❌ Not implemented

If you need any of the missing features, please [open an issue on the AVR8js repo](https://github.com/wokwi/avr8js/issues/new) or [reach out on Discord](https://wokwi.com/discord).

### **Serial Monitor**[**​**](https://docs.wokwi.com/parts/wokwi-arduino-uno#serial-monitor)

You can use the Serial Monitor to receive information from your Arduino code, such as debug print. You can also use it to send information to your code, such as textual commands.

For more information and code samples, check out [the Serial Monitor guide](https://docs.wokwi.com/guides/serial-monitor). It also explains how to configure the Serial monitor, e.g. set the line ending characters.

### Libraries[​](https://docs.wokwi.com/parts/wokwi-arduino-uno#libraries)

The simulator supports many popular Arduino libraries. For a complete list, see the [Libraries guides](https://docs.wokwi.com/guides/libraries).

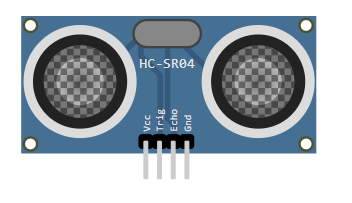
# **Simulator examples**[​](https://docs.wokwi.com/parts/wokwi-arduino-uno#simulator-examples)

* [Arduino Blink](https://wokwi.com/projects/344891652101374548)

# **hc-sr04 Reference**

HC-SR04 Ultrasonic Distance Sensor

HC-SR04



# **Pin names**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#pin-names)

|  |  |
| --- | --- |
| **Name** | **Description** |
| VCC | Voltage supply (5V) |
| TRIG | Pulse to start the measurement |
| ECHO | Measure the high pulse length to get the distance |
| GND | Ground |
|  |  |

# **Attributes**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#attributes)

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Default value** |
| distance | Initial distance value, in centimeters | "400" |

# **Operation**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#operation)

To start a new distance measurement set the TRIG pin to high for 10uS or more. Then wait until the ECHO pin goes high, and count the time it stays high (pulse length). The length of the ECHO high pulse is proportional to the distance. Use the following table to convert the ECHO pulse length in microseconds into centimeters / inches

|  |  |
| --- | --- |
| **Unit** | **Distance** |
| Centimeters | PulseMicros / 58 |
| Inches | PulseMicros / 148 |

# **Setting the distance**[**​**](https://docs.wokwi.com/parts/wokwi-hc-sr04#setting-the-distance)

To change the distance while the simulation is running, click on the HC-SR04 drawing in the diagram and use the slider to set the distance value. You can choose any value between 2cm and 400cm.

# **Arduino code example**[​](https://docs.wokwi.com/parts/wokwi-hc-sr04#arduino-code-example)

**#*define* PIN\_TRIG 3  
#*define* PIN\_ECHO 2  
  
*void* setup() {  
Serial.begin(115200);  
pinMode(PIN\_TRIG, OUTPUT);  
pinMode(PIN\_ECHO, INPUT);  
}  
  
*void* loop() {  
*// Start a new measurement:*  
digitalWrite(PIN\_TRIG, HIGH);  
delayMicroseconds(10);  
digitalWrite(PIN\_TRIG, LOW);  
  
*// Read the result:*  
*int* duration = pulseIn(PIN\_ECHO, HIGH);  
Serial.print("Distance in CM: ");  
Serial.println(duration / 58);  
Serial.print("Distance in inches: ");  
Serial.println(duration / 148);  
  
 delay(1000);  
}**

Try [this example on Wokwi](https://wokwi.com/projects/304444938977804866)

## Simulator examples[​](https://docs.wokwi.com/parts/wokwi-hc-sr04#simulator-examples)

* [Distance sensor and LED](https://wokwi.com/projects/290056311044833800)
* [Display distance on LCD screen](https://wokwi.com/projects/290043622233997832)
* [Seven segment distance display](https://wokwi.com/projects/295030553275532810)
* [Franzininho ultrasonic sensor](https://wokwi.com/projects/302020345098928648)
* [Parking sensor](https://wokwi.com/projects/290964046833779209)

# led Reference

Standard 5mm LED.

# **Pin names**[**​**](https://docs.wokwi.com/parts/wokwi-led#pin-names)

|  |  |
| --- | --- |
| **Name** | **Description** |
| A | Anode (positive pin) |
| C | Cathode (negative pin) |

# **Attributes**[**​**](https://docs.wokwi.com/parts/wokwi-led#attributes)

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Default value** |
| color | The color of the LED body | "red" |
| lightColor | The color of the light | depends on the color |
| label | Text that appears below the led |  |
| gamma | Gamma correction factor | "2.8" |
| flip | Flips the led horizontally | "" |

Note: To rotate LEDs, click on them and press "R", or set the ["rotate" property](https://docs.wokwi.com/diagram-format#parts).

# **Gamma correction**[**​**](https://docs.wokwi.com/parts/wokwi-led#gamma-correction)

The LED automatically applies gamma correction. This means that even a very short burst of current will result in some visible light, similar to how physical LEDs work, so you get more accurate simulation in the following cases:

1. Using analogWrite() with very small values (short duty cycle),
2. LED scanning techniques such as [Charlieplexing](https://goodarduinocode.com/guides/charlieplexing).

You can disable the gamma correction by setting the "gamma" attribute to "1.0". You can also choose a different gamma factor by setting this attribute to the desired value. The default gamma correction factor is 2.8.

The [Gamma Correction Demo project](https://wokwi.com/projects/304762988710068800) shows the behavior of different gamma values: the LED on the left has the default gamma factor of 2.8, while the LED on the right has a gamma factor of 1.0. You can see how lower values of analogWrite() look much brighter on the left LED.

For more information about gamma correction, including some code examples, check out this [great guide from Adafruit](https://learn.adafruit.com/led-tricks-gamma-correction).

# **Simulator examples**[**​**](https://docs.wokwi.com/parts/wokwi-led#simulator-examples)

* [Blink](https://wokwi.com/projects/344891652101374548) - Arduino's standard Blink sketch
* [Fade](https://wokwi.com/projects/313268562698437186) - Using analogWrite() + gamma correction.

# **Schematics** **diagram**

# WhatsApp Image 2023-10-31 at 10.22.31 AM.jpeg

# **Circuit diagram**

# WhatsApp Image 2023-10-31 at 10.42.25 AM.jpeg

# **Objectives:**

The primary objective of the Smart Traffic Management System is to optimize traffic flow, enhance road safety, and reduce congestion in urban areas. The project aims to achieve the following objectives:

1. **congestion levels**.
2. **Smart Traffic Lights:** Implement adaptive traffic signal control to reduce waiting times and improve traffic flow.
3. **Incident Detection:** Detect and respond to accidents, road closures, and other incidents promptly.
4. **Traffic Data Analysis**: Analyze historical traffic data to identify patterns and make informed decisions for infrastructure improvements.
5. **Mobile App Integration**: Develop a user-friendly mobile app for drivers and pedestrians to access real-time traffic information and receive alerts and guidance.
6. **Environmental Impact:** Reduce greenhouse gas emissions by optimizing traffic flow and minimizing vehicle idling.

**IoT Real-time Traffic Monitoring**: Collect real-time data on traffic conditions, including vehicle counts, speeds, and Sensor Setup: The IoT sensor setup is essential for collecting real-time traffic data. Key sensors and devices include:

1. **Traffic Cameras**: Capture images and videos of traffic flow, which can be analyzed to determine vehicle counts, speeds, and congestion.
2. **Vehicle Counting Sensors**: Utilize sensors like inductive loops or infrared sensors to count the number of vehicles passing a given point.
3. **Traffic Flow Sensors**: Collect data on vehicle speeds, direction, and flow patterns.
4. **Environmental Sensors**: Measure air quality and noise levels to assess the environmental impact of traffic.

**Mobile App Development**: The mobile app is a critical component that enables users to access traffic information and contribute to the traffic management system. Key features of the mobile app include:

1. **Real-time Traffic Data**: Display live traffic information, including congestion levels, traffic speeds, and incidents.
2. **Navigation Assistance**: Provide route recommendations to avoid traffic congestion and incidents.
3. **Incident Reporting**: Allow users to report accidents, road closures, and other incidents.
4. **Alerts and Notifications:** Send real-time alerts about traffic incidents, road closures, and alternative routes.
5. **User Feedback**: Collect user feedback on traffic conditions to improve the system's accuracy.
6. **Environmental Impact**: Display information about the environmental impact of traffic and promote eco-friendly transportation options.

**Raspberry Pi Integration**: The Raspberry Pi acts as the central control unit for the traffic management system and performs various tasks, including:

1. **Sensor Data Processing**: Collect and process data from various traffic sensors and cameras.
2. **Adaptive Traffic Signal Control**: Adjust traffic light timings in real-time based on traffic conditions.
3. **Incident Detection**: Implement algorithms to detect and respond to traffic incidents, road closures, and accidents.
4. **Communication Hub**: Facilitate communication between traffic sensors, the mobile app, and the central traffic management server.
5. **Data Storage**: Store historical traffic data for analysis and future planning.

**Code Implementation**: The code implementation includes several components, such as:

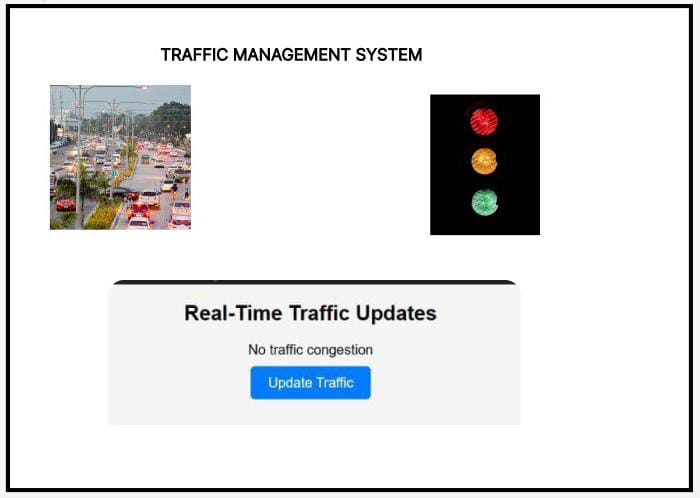
1. **Data Collection**: Code for collecting and processing data from traffic sensors, cameras, and environmental sensors.
2. **Traffic Signal Control**: Implement adaptive traffic signal control algorithms to optimize traffic flow.
3. **Incident Detection**: Code for detecting and responding to traffic incidents in real-time.
4. **Mobile App Development:** Code for creating the mobile app, including the user interface and data communication.
5. **Data Analysis**: Code for analyzing historical traffic data to identify patterns and make informed decisions for traffic management.
6. **User Alerts and Notifications:** Code for sending alerts and notifications to the mobile app users.
7. **Environmental Impact Assessment:** Implement code to calculate and display the environmental impact of traffic.

Successful implementation of these components will result in an efficient Smart Traffic Management System that improves traffic flow, safety, and environmental sustainability while providing real-time information and guidance to users through the mobile app.

Top of Form

# **Real-time traffic monitoring system**





# **Benefits of Managing**

A real-time traffic monitoring system can greatly assist commuters in making optimal route decisions and contribute to improving overall traffic flow. Here's how it works:

1. **Real-Time Traffic Data**: The system continuously collects and updates real-time traffic data from various sources such as traffic cameras, vehicle sensors, and incident reports. This data includes information on traffic congestion, road closures, accidents, and vehicle speeds.
2. **Mobile App Integration**: Commuters can access this real-time data through a mobile app specifically designed for this purpose. The app provides a user-friendly interface that displays the current traffic conditions on a map or list, allowing users to see congestion levels and incidents.
3. **Route Recommendations**: The mobile app uses the real-time data to provide commuters with route recommendations. It can suggest alternative routes to avoid congested areas, accidents, or road closures. These recommendations are based on current traffic conditions and can help commuters reach their destinations more quickly and efficiently.
4. **Navigation Assistance**: The app can provide turn-by-turn navigation assistance, guiding users through the recommended routes. This ensures that commuters stay on the optimal path, even in unfamiliar areas.
5. **Incident Alerts**: The system can send real-time alerts to the mobile app when incidents occur, such as accidents or road closures. Commuters are notified promptly and can adjust their routes accordingly. This helps prevent additional congestion caused by drivers approaching the incident area.
6. **User Reports and Feedback**: The system often allows users to report incidents they encounter, such as accidents, debris on the road, or potholes. This user-generated data can be invaluable for other commuters and traffic authorities, enhancing the overall effectiveness of the system.
7. **Environmental Impact**: Some systems also provide information on the environmental impact of traffic, such as air quality and greenhouse gas emissions. Commuters who are environmentally conscious can make route decisions that reduce their impact on the environment.
8. **Optimizing Traffic Flow**: As more commuters use the system to make informed route decisions, it helps distribute traffic more evenly across road networks. This, in turn, can reduce congestion in areas that are often heavily affected by traffic jams.
9. **Traffic Management**: Traffic management authorities can also use the data collected by the system to adjust traffic signal timings, optimize traffic flow at intersections, and respond to incidents more effectively. This leads to a better-controlled traffic environment.

In summary, a real-time traffic monitoring system assists commuters by providing up-to-date information on traffic conditions, suggesting optimal routes, and offering navigation guidance. By making informed decisions, commuters can avoid traffic jams and incidents, which ultimately leads to less congestion and smoother traffic flow on the road network. Additionally, it helps reduce stress and frustration for commuters and contributes to a more efficient and sustainable transportation system.

Top of Form