**Exp. No : STUDY OF ARDUINO  
Date:**

**Aim:**

To study the features, working, and architecture of an Arduino.

**Component required:**

* Arduino UNO

**Theory:**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

**Why Arduino?**

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

* Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
* Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
* Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
* Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
* Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

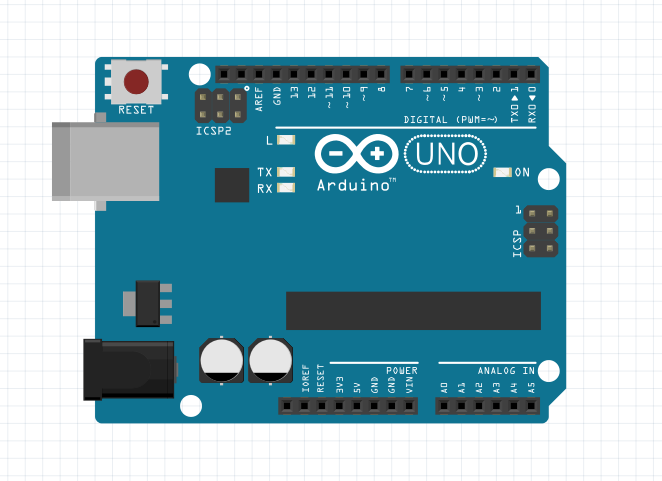
**Arduino UNO**

The Arduino UNO is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 Digital pins, 6 Analog pins, and programmable with the Arduino IDE (Integrated Development Environment) via a type B USB cable. It can be powered by a USB cable or by an external 9 volt battery, though it accepts voltages between 7 and 20 volts.

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform. The ATmega328 on the Arduino Uno comes pre-programmed with a bootloader that allows uploading new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol. The Uno also differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it uses the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

*Technical specifications:*

* Microcontroller: Microchip ATmega328P
* Operating Voltage: 5 Volt
* Input Voltage: 7 to 20 Volts
* Digital I/O Pins: 14 (of which 6 provide PWM output)
* Analog Input Pins: 6
* DC Current per I/O Pin: 20 mA
* DC Current for 3.3V Pin: 50 mA
* Flash Memory: 32 KB of which 0.5 KB used by bootloader
* SRAM: 2 KB
* EEPROM: 1 KB
* Clock Speed: 16 MHz
* Length: 68.6 mm
* Width: 53.4 mm
* Weight: 25 g



**Figure - 2.1:** Arduino UNO

**Pin Details:**

*General pin functions*

* LED: There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
* VIN: The input voltage to the Arduino/Genuino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
* 5V: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), or the VIN pin of the board (7-20V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage the board.
* 3.3V: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
* GND: Ground pins.
* IOREF: This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.
* Reset: Typically used to add a reset button to shields which block the one on the board.[7]

*Special Pin Functions*

Each of the 14 digital pins and 6 Analog pins on the Uno can be used as an input or output, using pin Mode (), digital Write (), and digital Read () functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller.The Uno has 6 analog inputs, labelled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function.

In addition, some pins have specialized functions:

* Serial: pins 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
* External Interrupts: pins 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
* PWM(Pulse Width Modulation) 3, 5, 6, 9, 10, and 11 Can provide 8-bit PWM output with the analogWrite() function.
* SPI(Serial Peripheral Interface): 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
* TWI(Two Wire Interface): A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.
* AREF(Analog REFerence): Reference voltage for the analog inputs.

**Communication**

The Arduino/Genuino Uno has a number of facilities for communicating with a computer, another Arduino/Genuino board, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows serial communication on any of the Uno's digital pins.

**Automatic (Software) Reset**

Rather than requiring a physical press of the reset button before an upload, the Arduino/Genuino Uno board is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the boot loader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened.

**Result:**

The features, working, and architecture of an Arduino have been studied.

**Exp. No.: BLINKING OF AN LED USING ARDUINO**

**Date:**

**Aim:**

To interface an LED with Arduino and to write a program to make the LED blink at defined intervals.

**Components required:**

* PC or laptop
* Arduino UNO
* 220 resistor

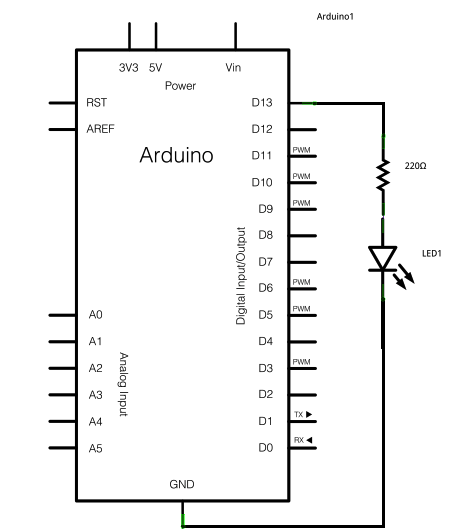


* LED
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



* Connect the longer leg of the LED, that is the anode, to the digital pin 13 in the Arduino, through a 220 resistor.



* Connect the shorter leg of the LED, that is the cathode, to the ground (GND) pin on the Arduino.

**Procedure:**

1. Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
2. Open Aurdino IDE and open a new editor.
3. Enter the required code save it as a “ino” file.
4. In the “Tools” menu go to the “Port” tab and select the appropriate port.
5. Upload the program into the Arduino and verify the output.

**Program:**

**To make the built-in LED blink:**

void setup() {  
  // initialize digital pin LED\_BUILTIN as an output.  
  pinMode(LED\_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(LED\_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)  
  delay(1000);                       // wait for a second  
  digitalWrite(LED\_BUILTIN, LOW);    // turn the LED off by making the voltage LOW  
  delay(1000);                       // wait for a second  
}

**To make the externally connected LED blink:**

int led = 13;

void setup() {  
  // initialize digital pin LED\_BUILTIN as an output.  
  pinMode(led, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)  
  delay(1000);                       // wait for a second  
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW  
  delay(1000);                       // wait for a second  
}

**Result:**

A program has been written to make the built-in and externally connected LED blink and the output verified.

**Exp. No.: FADING OF AN LED USING ARDUINO**

**Date:**

**Aim:**

To interface an LED with Arduino and to write a program to make the LED fade.

**Components required:**

* PC or laptop
* Arduino UNO
* 220 resistor

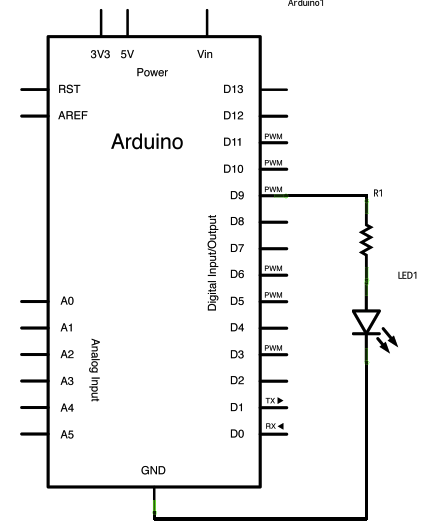


* LED
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



* Connect the longer leg of the LED, that is the anode, to the digital pin 9 in the Arduino, through a 220 resistor.



* Connect the shorter leg of the LED, that is the cathode, to the ground (GND) pin on the Arduino.

**Procedure:**

1. Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
2. Open Arduino IDE and open a new editor.
3. Enter the required code save it as a “ino” file.
4. In the “Tools” menu go to the “Port” tab and select the appropriate port.
5. Upload the program into the Arduino and verify the output.

**Program:**

int led = 9;           // the PWM pin the LED is attached to  
int brightness = 0;    // how bright the LED is  
int fadeAmount = 5;    // how many points to fade the LED by  
// the setup routine runs once when you press reset:  
void setup() {  
  // declare pin 9 to be an output:  
  pinMode(led, OUTPUT);  
}  
// the loop routine runs over and over again forever:  
void loop() {  
  // set the brightness of pin 9:  
  analogWrite(led, brightness);  
  // change the brightness for next time through the loop:  
  brightness = brightness + fadeAmount;  
  // reverse the direction of the fading at the ends of the fade:  
  if (brightness <= 0 || brightness >= 255) {  
    fadeAmount = -fadeAmount;  
  }  
  // wait for 30 milliseconds to see the dimming effect  
  delay(30);  
}

**Result:**

A program has been written to make the LED connected to an Arduino fade and the output verified.

**Exp. No.: INTERFACING A WATER-LEVEL SENSOR WITH AN ARDUINO**

**Date:**

**Aim:**

To interface a water-level sensor with an Arduino and to write a program to measure the water level.

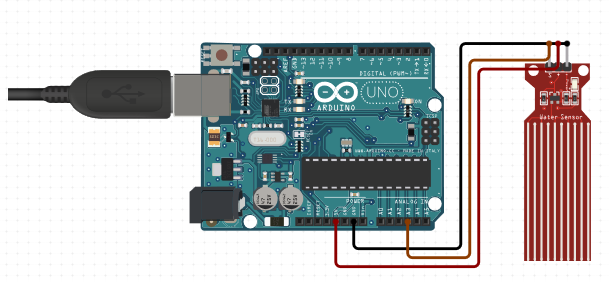
**Components required:**

* PC or laptop
* Arduino UNO
* Water-level sensor
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



* Connect the GND pin on the water-level sensor to that on the Arduino.
* Connect the SIG pin on the sensor to analog pin A3 of the Arduino.
* Connect the VCC pin on the sensor to the 5V pin on the Arduino.

**Procedure:**

1. Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
2. Open Arduino IDE and open a new editor.
3. Enter the required code save it as an “ino” file.
4. In the “Tools” menu go to the “Port” tab and select the appropriate port.
5. Upload the program into the Arduino and verify the output on the serial monitor in Arduino IDE.

**Program:**

int sensorPin = A3;

int sensorValue = 0;

int value;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(sensorPin, INPUT);

}

void loop() {

// put your main code here, to run repeatedly:

//sensorValue = analogRead(sensorPin);

value = analogRead(sensorPin);

if (value<=480){

Serial.println("Water level: 0mm - Empty!");

}

else if (value>480 && value<=530){

Serial.println("Water level: 0mm to 5mm");

}

else if (value>530 && value<=615){

Serial.println("Water level: 5mm to 10mm");

}

else if (value>615 && value<=660){

Serial.println("Water level: 10mm to 15mm");

}

else if (value>660 && value<=680){

Serial.println("Water level: 15mm to 20mm");

}

else if (value>680 && value<=690){

Serial.println("Water level: 20mm to 25mm");

}

else if (value>690 && value<=700){

Serial.println("Water level: 25mm to 30mm");

}

else if (value>700 && value<=705){

Serial.println("Water level: 30mm to 35mm");

}

else if (value>705){

Serial.println("Water level: 35mm to 40mm");

}

delay(2000);

}

**Output:**

**Result:**

A water-level sensor was interfaced with an Arduino and a program to measure the water level was written and executed.

**Exp. No.: INTERFACING AN ULTRSONIC SENSOR WITH AN ARDUINO**

**Date:**

**Aim:**

To interface an ultrasonic sensor with an Arduino and to write a program to measure the distance of an object from the sensor.

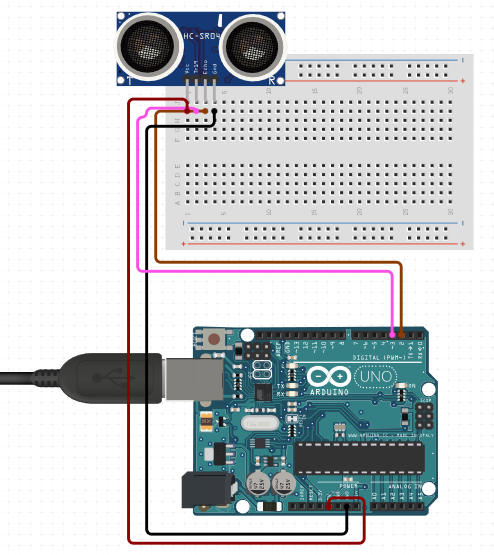
**Components required:**

* PC or laptop
* Arduino UNO
* Ultrasonic sensor HCSR04
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



* Connect the ECHO pin on the HCSR04 sensor to digital pin 2 of the Arduino.
* Connect the ground (GND) of the sensor to GND of the Arduino.
* Connect the TRIG pin of the sensor to digital pin 3 of the Arduino.
* Connect the VCC pin of the sensor to the 5V pin on the Arduino.

**Procedure:**

1. Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
2. Open Arduino IDE and open a new editor.
3. Enter the required code save it as an “ino” file.
4. In the “Tools” menu go to the “Port” tab and select the appropriate port.
5. Upload the program into the Arduino and verify the output on the serial monitor in Arduino IDE.

**Program:**

// Include Libraries

#include "Arduino.h"

#include "NewPing.h"

// Pin Definitions

#define HCSR04\_PIN\_TRIG 3

#define HCSR04\_PIN\_ECHO 2

// Global variables and defines

// object initialization

NewPing hcsr04(HCSR04\_PIN\_TRIG,HCSR04\_PIN\_ECHO);

// define vars for testing menu

const int timeout = 10000; //define timeout of 10 sec

char menuOption = 0;

long time0;

// Setup the essentials for your circuit to work. It runs first every time your circuit is powered with electricity.

void setup()

{

// Setup Serial which is useful for debugging

// Use the Serial Monitor to view printed messages

Serial.begin(9600);

while (!Serial) ; // wait for serial port to connect. Needed for native USB

Serial.println("start");

menuOption = menu();

}

// Main logic of your circuit. It defines the interaction between the components you selected. After setup, it runs over and over again, in an eternal loop.

void loop()

{

if(menuOption == '1') {

// Ultrasonic Sensor - HC-SR04 - Test Code

// Read distance measurment from UltraSonic sensor

int hcsr04Dist = hcsr04.ping\_cm();

delay(1001);

Serial.print(F("Distance: ")); Serial.print(hcsr04Dist); Serial.println(F("[cm]"));

}

if (millis() - time0 > timeout)

{

menuOption = menu();

}

}

// Menu function for selecting the components to be tested

// Follow serial monitor for instructions

char menu()

{

Serial.println(F("\nWhich component would you like to test?"));

Serial.println(F("(1) Ultrasonic Sensor - HC-SR04"));

Serial.println(F("(menu) send anything else or press on board reset button\n"));

while (!Serial.available());

// Read data from serial monitor if received

while (Serial.available())

{

char c = Serial.read();

if (isAlphaNumeric(c))

{

if(c == '1')

Serial.println(F("Now Testing Ultrasonic Sensor - HC-SR04"));

else

{

Serial.println(F("illegal input!"));

return 0;

}

time0 = millis();

return c;

}

}

}

**Output:**

**Result:**

An ultrasonic sensor has been interfaced with the Arduino and the distance of an object from the sensor has been measured.

**Exp. No.:** TO USE A BUZZER (OR PIEZO SPEAKER) WITH ARDUINO

**Date:**

**Aim:**

To interface an buzzer sensor with an Arduino.

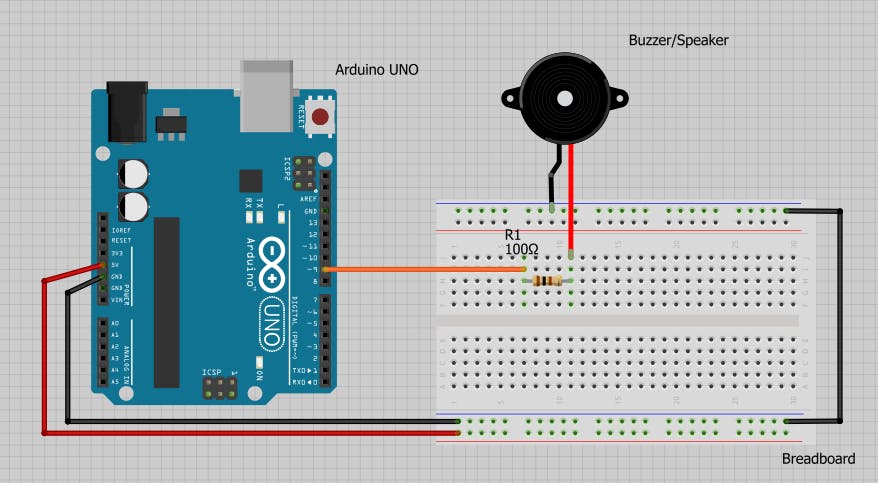
**Components required:**

* PC or laptop
* Arduino UNO
* Buzzer
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



**Procedure:**

1. Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
2. Open Arduino IDE and open a new editor.
3. Enter the required code save it as an “ino” file.
4. In the “Tools” menu go to the “Port” tab and select the appropriate port.
5. Upload the program into the Arduino and verify the output on the serial monitor in Arduino IDE.

**Program:**

const int buzzer = 9; //buzzer to arduino pin 9

void setup(){

pinMode(buzzer, OUTPUT); // Set buzzer - pin 9 as an output

}

void loop(){

tone(buzzer, 1000); // Send 1KHz sound signal...

delay(1000); // ...for 1 sec

noTone(buzzer); // Stop sound...

delay(1000); // ...for 1sec

}

**Result:**

**Exp. No.: MQ-6 GAS SENSOR INTERFACING WITH ARDUINO**

**Date:**

**Aim:**

To interface a MQ-6 Gas sensor with an Arduino.

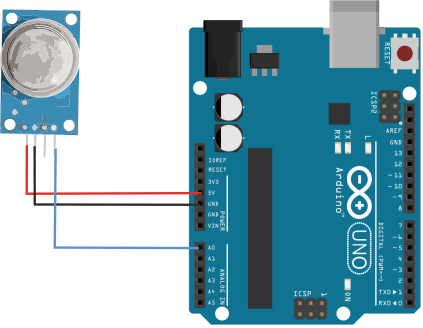
**Components required:**

* PC or laptop
* Arduino UNO
* MQ-6 GAS SENSOR
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



VCC - 5V

GND - GND

S   - Analog pin0

**Procedure:**

* Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
* Open Arduino IDE and open a new editor.
* Enter the required code save it as an “ino” file.
* In the “Tools” menu go to the “Port” tab and select the appropriate port.
* Upload the program into the Arduino and verify the output on the serial monitor in Arduino IDE.

**Program:**

void setup() {

// initialize serial communication at 9600 bits per second:

Serial.begin(9600);

}

// the loop routine runs over and over again forever:

void loop() {

// read the input on analog pin 0:

int sensorValue = analogRead(A0);

// print out the value you read:

Serial.println(sensorValue);

delay(1000);

}

**Result:**

**Exp. No.:** LCD INTERFACING WITH THE ARDUINO MODULE

**Date:**

**Aim:**

To interface a LCD with an Arduino.

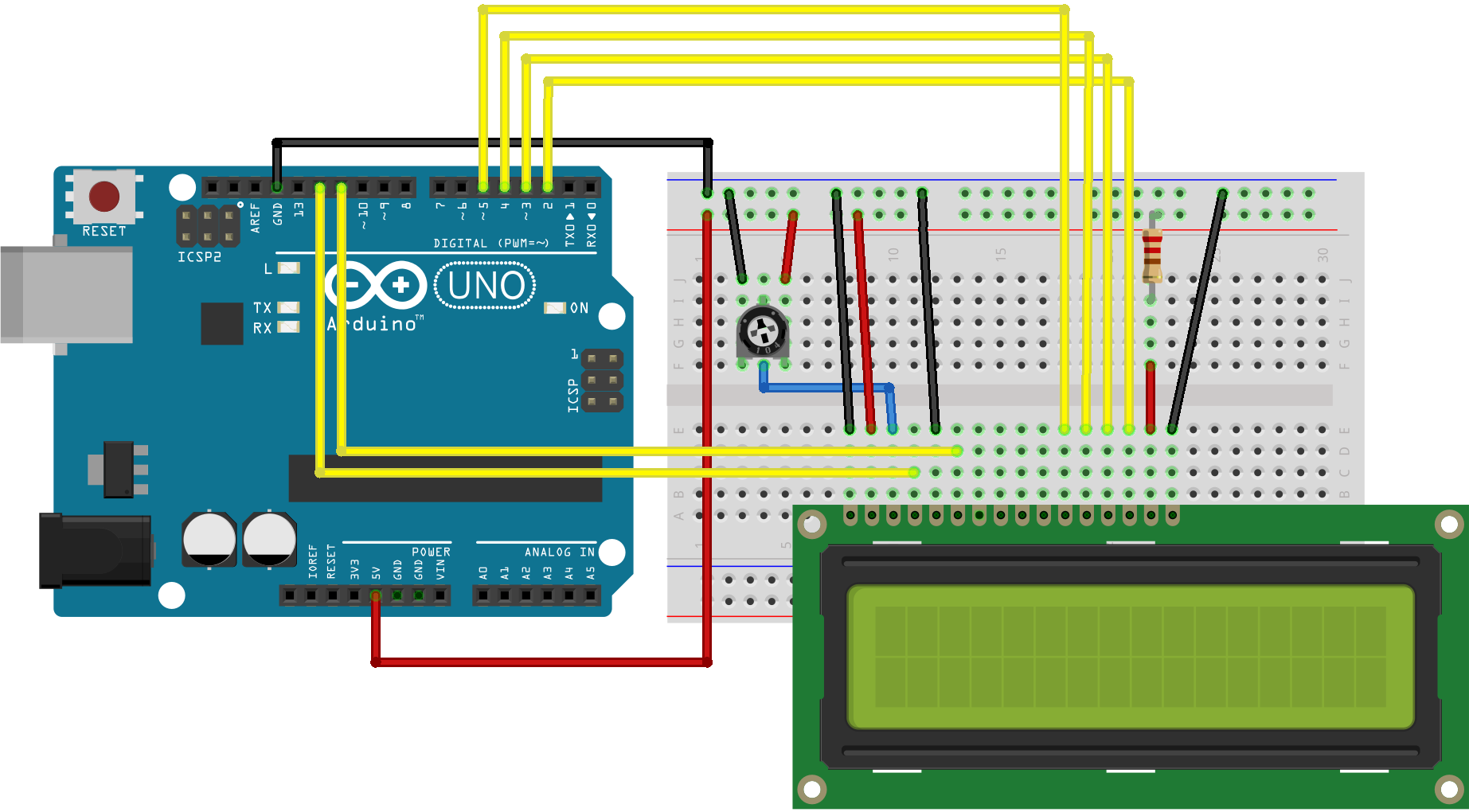
**Components required:**

* PC or laptop
* Arduino UNO
* LCD module
* Breadboard
* Connecting wires

**Software required:**

* Arduino IDE

**Connections:**



* LCD RS pin to digital pin 12
* LCD Enable pin to digital pin 11
* LCD D4 pin to digital pin 5
* LCD D5 pin to digital pin 4
* LCD D6 pin to digital pin 3
* LCD D7 pin to digital pin 2
* Additionally, wire a 10k pot to +5V and GND, with it's wiper (output) to LCD screens VO pin (pin3). A 220 ohm resistor is used to power the backlight of the display, usually on pin 15 and 16 of the LCD connector

**Procedure:**

* Connect the components on a breadboard as mentioned above and connect the Arduino to the PC/laptop.
* Open Arduino IDE and open a new editor.
* Enter the required code save it as an “ino” file.
* In the “Tools” menu go to the “Port” tab and select the appropriate port.
* Upload the program into the Arduino and verify the output on the serial monitor in Arduino IDE.

**Program:**

#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins

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

void setup() {

// set up the LCD's number of columns and rows:

lcd.begin(16, 2);

// Print a message to the LCD.

lcd.print("hello, world!");

}

void loop() {

// set the cursor to column 0, line 1

// (note: line 1 is the second row, since counting begins with 0):

lcd.setCursor(0, 1);

// print the number of seconds since reset:

lcd.print(millis() / 1000);

}

**Result:**