جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Intelligent Medical Systems Department

Lecture: (3)

Serial monitor& Sensors II

Subject: Embedded system

Class: Third

Lecturer: Prof.Dr. Mehdi Ebady Manaa

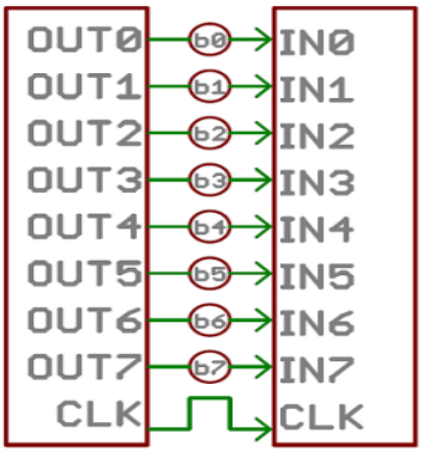
Programmer:- Fatima Hussein Jawad

Study Year: 2024-2025

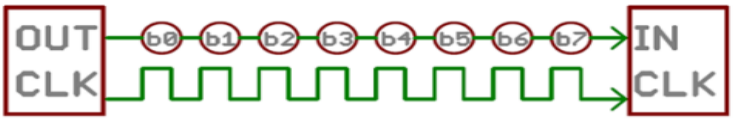
Serial monitor

To make electronic circuits transmit information to each other, you must share a common communication protocol. There are two types of communication:

1. **Parallel communication**
   It is transfers several bits at the same time. Uses a number of wires of eight or sixteen or more, and the data are transmitted in huge waves of values 0 and 1, the bytes are transmitted through each pulse of the clock.
2. **Serial communication**
   It is transferring data bit by bit. These interfaces can work using a few wires up to one wire and usually do not exceed four wires, one bit transfer each pulse clock. Divided into two types:
   a- **Synchronous Serial**: All devices connected to a synchronous serial bus share a common clock, requiring at least one additional wire between connected devices.
   b- **Asynchronous Serial**: Data is transferred without the use of an external clock signal. It is the best way to reduce the number of wires required.



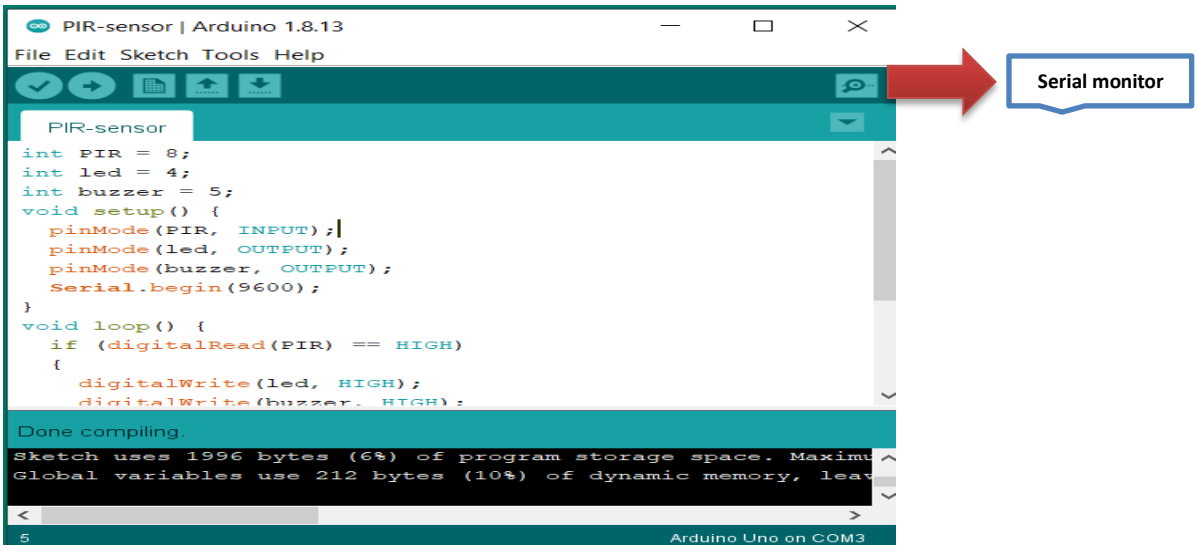
Parallel communication



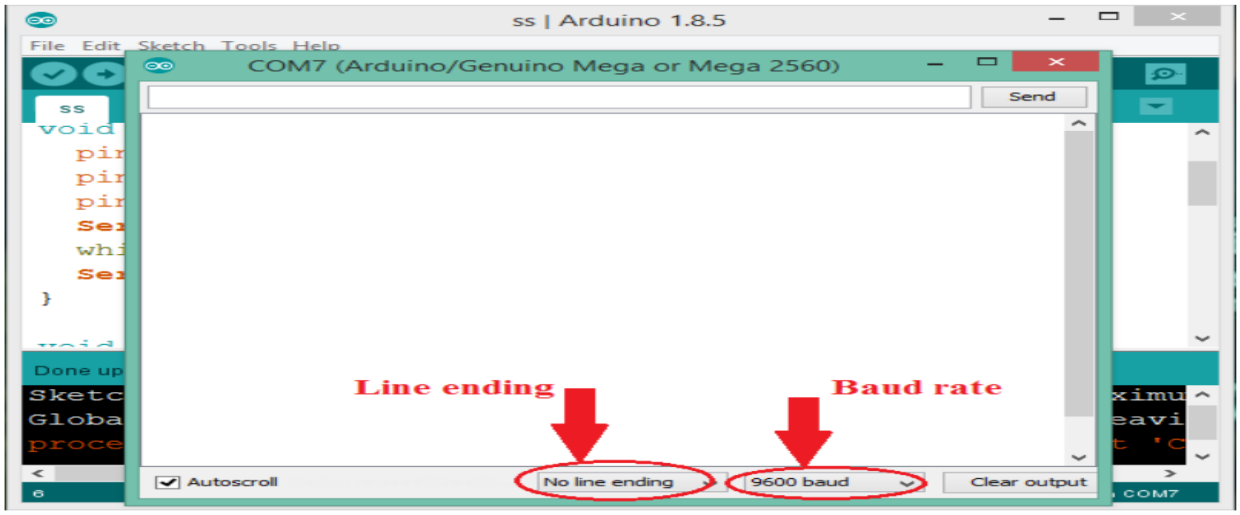
Serial communication

Serial monitor: The Serial Monitor is a separate pop-up window that acts as a separate terminal that communicates by receiving and sending Serial Data. It is the link between the computer and the Arduino.

Serial Data is sent as Asynchronous Serial over a single wire and consists of a series of 1's and 0's sent over the wire. Data can be sent in both directions (In our case on two wires).



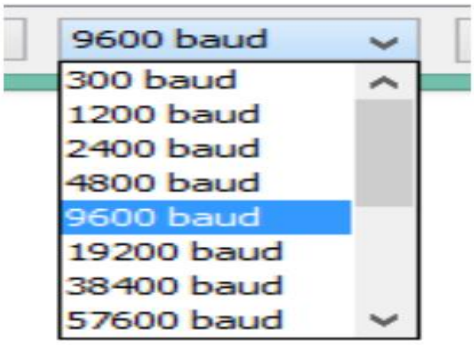
When open the window will find that it contains the following:



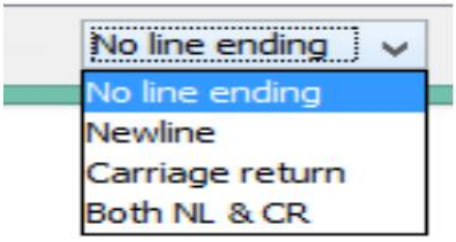
Baud rate: Determines the speed of data transmission over the serial line, expressed in bits / sec (bps).

The Baud rate can be any possible value. The only thing that is required is that both devices work at the same rate. Of the most common baud rates especially in simple devices that do not require high speed is 9600bps. Other standard rates include 300, 1200, 2400, 4800, 19200, 34800, 57600, 115200 and so on...

The higher the baud rate the higher the speed of data transmission / reception, but there are limits to the speed at which data can be reduced.A very high speed will cause errors in the receiving end.



Line ending: Specifies what sends after the value entered to the Arduino by pressing the send button.



Serial monitor Functions

- **Serial.begin():** Sets the data rate in bits per second (baud) for serial data transmission, For communicating between the computer and Arduino.

Serial.begin(speed, config)

Speed Measured by Baud rate

config sets data, parity, and stop bits(by default SERIAL_8N1) .

NOTE: Can put just speed (Serial.begin(speed))so the config will be Automatically default SERIAL_8N1. Ex: Serial.begin(9600);

- **Serial .available():** Get the number of bytes (characters) available for reading from the serial port.

Serial. available():

- **Serial.read():** Reads incoming serial data.

Serial.read():

- **Serial.print:** Prints data to the serial port as human-readable ASCII text.

Serial.print(val)

Serial.print(val, format)

- **Serial.println():** Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline

character (ASCII 10, or '\n'). This command takes the same forms as Serial.print().

Serial.println(val)

Serial.println(val, format)

Val: The value to be printed.

(EX: Serial.print("Hello World")→ gives Hello world)

Format (optional): The formula to convert the value for it and print it.

(EX: Serial.print(33, HEX)→ gives 21, Serial.print(33, BIN)→ gives 100001, Serial.print(33.7845,2)→ gives 33.78).

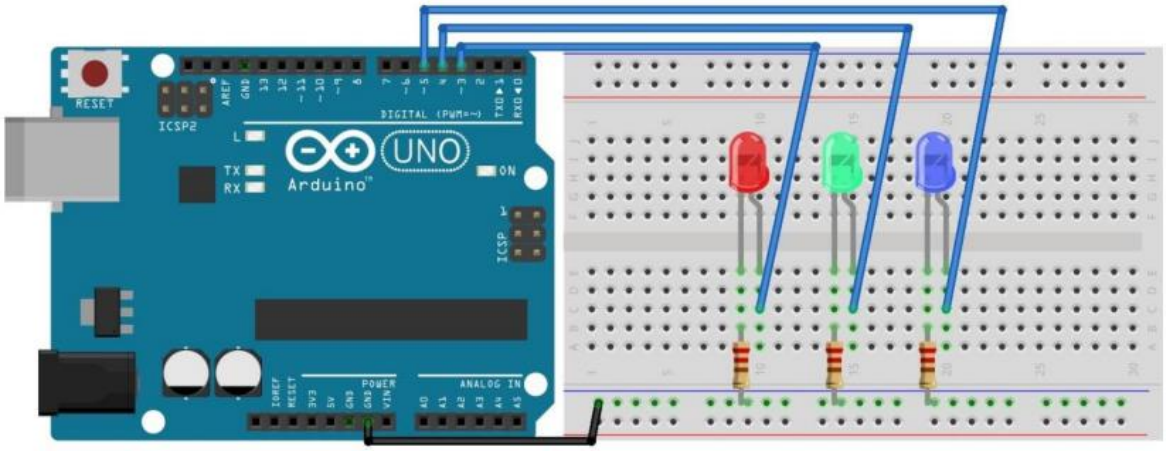
Other Functions: If (Serial), availableForWrite(), end(), find(), findUntil(), flush(), parseFloat(), parseInt(), peek(), readBytes(), readBytesUntil(), readString(), readStringUntil(), setTimeout(), write(), serialEvent().

Example1

(Turn on Leds by serial monitor)

Requirements: Arduino, BreadBoard, 3Resistor, 3 Led, wires.

Connection map:



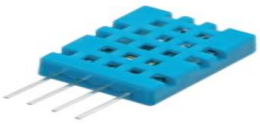
Code:

```
int led1 = 3;
int led2 = 4;
int led3 = 5;
void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter LED Number 3,4,5 to opened up & 0 to Close all");
}
void loop() {
  if (Serial.available()) {
    char c = Serial.read();
    if (c == '3') {
      digitalWrite(led1, HIGH);
      digitalWrite(led2, LOW);
      digitalWrite(led3, LOW);
    }
    else if (c == '4') {
      digitalWrite(led2, HIGH);
      digitalWrite(led1, LOW);
      digitalWrite(led3, LOW);
    }
    else if (c == '5') {
      digitalWrite(led3, HIGH);
```

```
      digitalWrite(led1, LOW);
      digitalWrite(led2, LOW);
    }
    else if (c == '0') {
      digitalWrite(led3, LOW);
      digitalWrite(led1, LOW);
      digitalWrite(led2, LOW);
    }
  }
}
```

Buzzer: A device that converts electrical power to audible sound. Typical uses of buzzers include alarm devices, timers and so on.

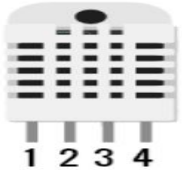
DHT11 sensor: stand for Digital Humidity & Temperature sensor. Used for measure temperature and humidity. There is another type of it known as DHT22.

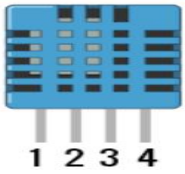


DHT11

DHT22

DHT11		DHT22
0 - 50°C / ± 2°C	Temperature Range	-40 - 125 °C / ± 0.5 °C
20 - 80% / ± 5%	Humidity Range	0 - 100 % / ± 2-5%
1Hz (one reading every second)	Sampling Rate	0.5 Hz (one reading every two seconds)
15.5mm x 12mm x 5.5mm	Body Size	15.1mm x 25mm x 7.7mm
3 - 5V	Operating Voltage	3 - 5V
2.5mA	Max Current During Measuring	2.5mA





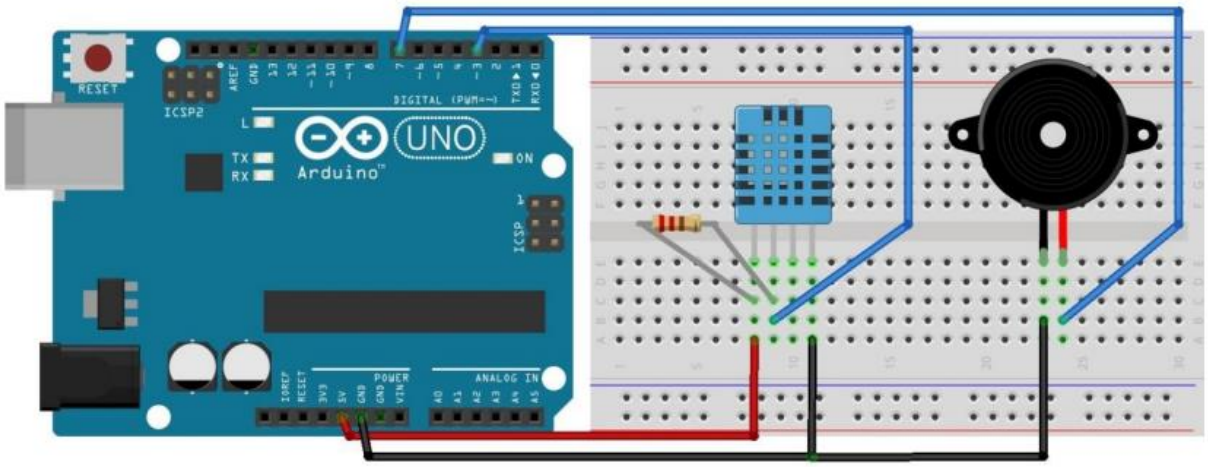
1 = VCC
2 = DATA
3 = NC
4 = GND

Example

(Measurement of temperature and humidity)

Requirements: Arduino, BreadBoard, Resistor, dht11 Sensor,Buzzer, wires.

Connection map:



Code:

(*first must add dht11 library - can download it from github)

```
#include <DHT11.h>

const int dht11 = 3;  // pin number the DHT11 sensor is connected
const int buzzer = 7; // pin number the buzzer is connected to
DHT11 DHT(dht11);     // pass pin number when initializing the object

void setup() {
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
}

void loop() {
```

```
// Read temperature and humidity
  float temperature = DHT.readTemperature();
  float humidity = DHT.readHumidity();

  if (temperature >= 23) {
    Serial.println("Warning!");
    tone(buzzer, 1000);
    delay(1000);
    noTone(buzzer);

    Serial.print("Temperature: C°= ");
    Serial.println(temperature);

    int k = temperature + 273.15; // convert to Kelvin
    Serial.print("k= ");
    Serial.println(k);

    int f = temperature * 1.8 + 32; // convert to Fahrenheit
    Serial.print("f= ");
    Serial.println(f);

    Serial.print("Humidity = ");
    Serial.println(humidity);
    Serial.println("------------------------");
    delay(3000);
  }
}
```

Code Explanation:

1. Variable definitions:

dht11: Refers to the pin number where the DHT11 sensor is connected.
buzzer: Refers to the pin number where the buzzer is connected.

DHT: An object from the DHT11 library that handles data reading from the sensor.

2. setup() function:

The pinMode function is used to set the buzzer pin as an output.

Serial.begin(9600) initializes serial communication to display results on the Serial Monitor.

3. loop() function:

The code reads temperature and humidity using DHT.readTemperature() and DHT.readHumidity() functions.

If the temperature is greater than or equal to 23°C:

The buzzer is activated using tone(buzzer, 1000), with a tone frequency of 1000 Hz for one second, followed by noTone(buzzer) to stop the buzzer.

The temperature and humidity values are printed to the Serial Monitor.

The temperature is converted to Kelvin and Fahrenheit and the converted values are printed.

Temperature conversions:

Kelvin: The conversion is done by adding 273.15 to the Celsius temperature.

Fahrenheit: The conversion is done by multiplying the Celsius temperature by 1.8 and then adding 32.

Delays:delay(1000) and delay(3000) are used to add pauses between actions to prevent rapid repetition of the loop.

Practical Applications:

This code can be used in environmental control systems or as an alert system when the temperature exceeds a certain threshold, such as in home or industrial applications