FUNDAMENTALS OF INTERNET OF THINGS

(III-CSE, SEMESTER-II, R-22)
PREPARED BY-MAGANTI APPARAO

HEAD OF THE DEPARTMENT

ST. MARY'S ENGINEERING COLLEGE

UNIT – II

MACHINE-TO-MACHINE COMMUNICATIONS

- **Difference between IoT and M2M**
- **Interoperability in IoT**
- **Introduction to Arduino Programming**
- **Integration of Sensors and Actuators with Arduino**

INTRODUCTION

Machine to Machine: This is commonly known as Machine to machine communication. It is a concept where two or more than two machines communicate with each other without human interaction using a wired or wireless mechanism.

M2M is a technology that helps the devices to connect between devices without using internet.

Internet of Things: IOT is known as the Internet of Things where things are said to be the communicating devices that can interact with each other using a communication media.

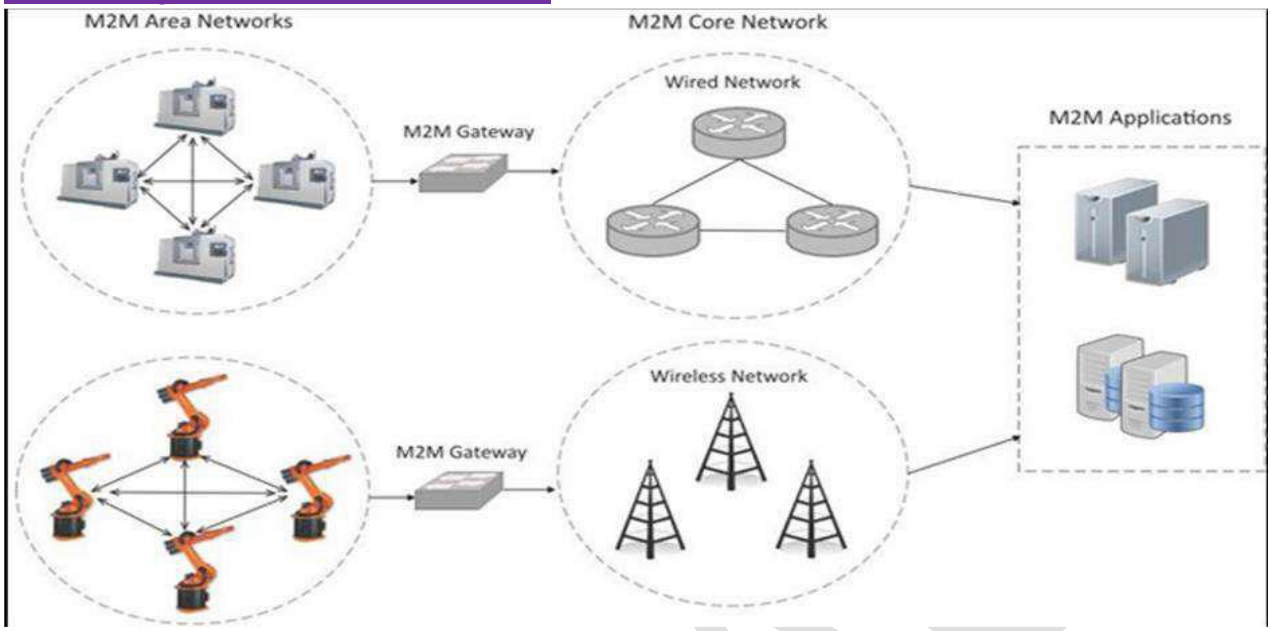
Usually every day some new devices are being integrated which uses IoT devices for its function.

These devices use various sensors and actuators for sending and receiving data over the internet. It is an ecosystem where the devices share data through a communication media known as the internet.

MACHINE-TO-MACHINE (M2M)

Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange. Term which is often synonymous with IoT is Machine-to-Machine (M2M). IoT and M2M are often used interchangeably.

M2M System Architecture



An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.

• Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooth, Modbus, M-Bus, Wireless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.

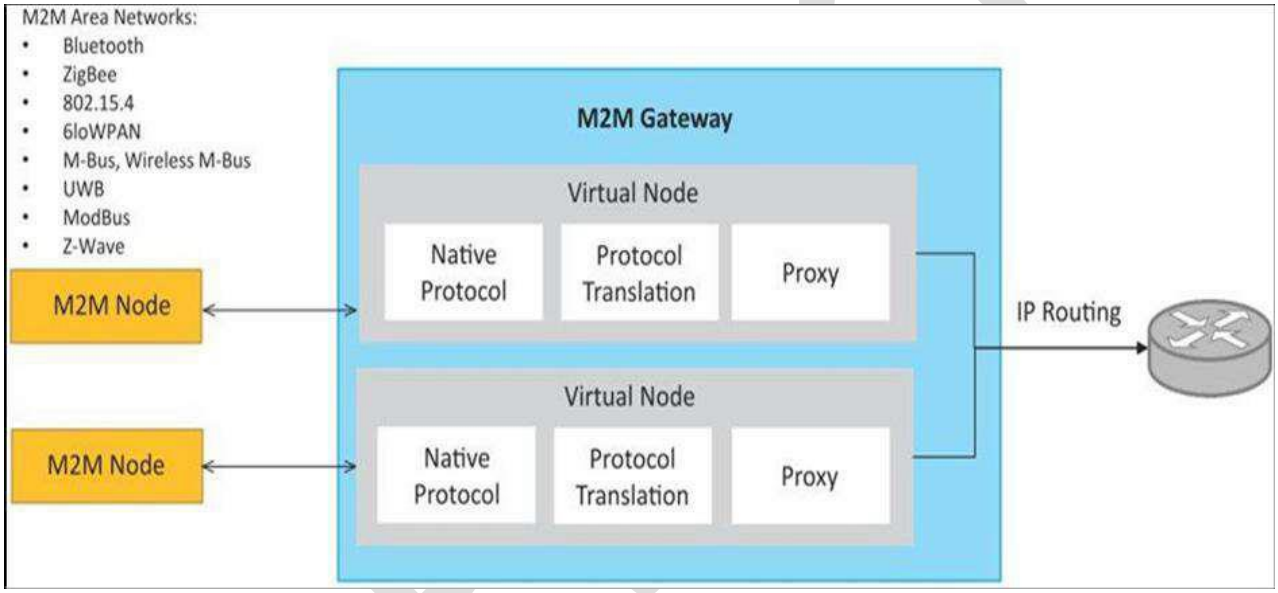
• The communication network provides connectivity to remote M2M area networks. • The communication network can use either wired or wireless networks (IP based).

• M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks M2M gateway.

• Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network.

• To enable the communication between remote M2M area networks, M2M gateways are used.

Block Diagram of an M2M Gateway



The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable IP-connectivity for M2M are networks.

M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol(IP). With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

M2M Features

- Large number of nodes or devices.
- Low cost.
- Energy efficient.
- Small traffic per machine/device.
- Large quantity of collective data.
- M2M communication free from human intervention.
- Human intervention required for operational stability and sustainability.

M2M Applications

- Environmental monitoring
- Civil protection and public safety
- Supply Chain Management (SCM)
- Energy & utility distribution industry (smart grid)
- Intelligent Transport Systems (ITSs)
- Healthcare
- Automation of building
- Military applications
- Agriculture
- Home networks

DIFFERENCES BETWEEN IOT AND M2M

1) Communication Protocols

Commonly uses M2M protocols include ZigBee, Bluetooth, Modbus, M-Bus, Wireless M-Bus etc.,

In IoT uses HTTP, CoAP, Web Socket

2) Machines in M2M Vs Things in IoT:

Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.

3) Hardware Vs Software Emphasis:

The emphasis of M2M is more on hardware with embedded modules, the emphasis of

IoT is more on software.

4) Data Collection &Analysis

M2M data is collected in point solutions and often in on-premises storage infrastructure.

The data in IoT is collected in the cloud (can be public, private or hybrid cloud).

5) Applications

M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premises enterprise applications.

IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

Parameter	**IoT**	**M2M**
Abbreviation	Internet of Things	Machine to Machine
Intelligence	Devices have objects that are responsible for decision making	Some degree of intelligence is observed in this
Connection type used	The connection is via Network and using various communication types.	The connection is a point to point
Communication protocol used	Internet protocols are used such as HTTP, FTP, and Telnet.	Traditional protocols and communication technology techniques are used
Data Sharing	Data is shared between other applications that are used to improve the end-user experience.	Data is shared with only the communicating parties.
Internet	Internet connection is required for communication	Devices are not dependent on the Internet.
Scope	A large number of devices yet scope is large.	Limited Scope for devices.
Business Type used	Business 2 Business(B2B) and Business 2 Consumer(B2C)	Business 2 Business (B2B)
Basis of	IoT	M2M
Open API support	Supports Open API integrations.	There is no support for Open APIs
Examples	Smart wearables, Big Data and Cloud, etc.	Sensors, Data and Information, etc.

INTEROPERABILITY IN IOT

Interoperability is defined by IEEE as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged.

In IoT interoperability can be defined as the ability of two systems to communicate and share services with each other.

The interoperability issues in IoT can be seen from different perspectives due to heterogeneity. Heterogeneity is not a new concept nor restricted to a domain.

Even in the physical world there are many types of heterogeneities for example, people speak dissimilar languages, but they can still communicate with each other through a translator (human/tools) or by using a common language.

Similarly, the diverse elements comprising IoT (devices, communication, services, applications, etc.) should seamlessly cooperate and communicate with each other to realize the full potential of IoT ecosystem.

Why Interoperability is Important in Context of IoT?

Physical objects can interact with any other physical objects and can share their information.

Any device can communicate with other devices anytime from anywhere.

Machine to Machine communication(M2M), Device to Device Communication (D2D), Device to Machine Communication (D2M).

Seamless device integration with IoT network.

Different wireless communication protocols such as ZigBee (IEEE 802.15.4), Bluetooth (IEEE 802.15.1), GPRS, 6LowPAN, and Wi-Fi (IEEE 802.11).

Different wired communication protocols like Ethernet (IEEE 802.3) and Higher Layer LAN Protocols (IEEE 802.1).

Different programming languages used in computing systems and websites such as JavaScript, JAVA, C, C++, Visual Basic, PHP, and Python.

Different hardware platforms such as Crossbow, NI, etc.

Different operating systems.

As an example for sensor node: Tiny OS, SOS, Mantis OS, RETOS, and mostly vendor specific OS.

As an example for personal computer: Windows, Mac, Unix, and Ubuntu.

Different databases: DB2, MSQL, Oracle, PostgreSQL, SQLite, SQL Server, and Sybase

Different data representations.

Different control models.

TYPES OF INTEROPERABILITY

- Device Interoperability
- Networking Interoperability
- Syntactic Interoperability
- Semantic Interoperability
- Platform Interoperability

Device Interoperability

Device interoperability refers to enabling the integration and interoperability of such heterogeneous devices with various communication protocols and standards.

Device interoperability is concerned with

(i) the exchange of information between heterogeneous devices and heterogeneous communication protocols and

(ii) (ii) the ability to integrate new devices into any IoT platform.

Network Interoperability

Network level interoperability deals with mechanisms to enable seamless message exchange between systems through different networks (networks of networks) for end-to-end communication. To make systems interoperable, i.e each system should be able to exchange messages with other systems through various types of networks.

Due to the dynamic and heterogeneous network environment in IoT, the network interoperability level should handle issues such as addressing, routing, resource optimization, security, QoS, and mobility support.

Syntactical Interoperability

Syntactic interoperability refers to interoperation of the format as well as the data structure used in any exchanged information or service between heterogeneous IoT system entities.

Semantic Interoperability

semantic interoperability is defined as “enabling different agents, services, and applications to exchange information, data and knowledge in a meaningful way, on and off the Web”.

Platform Interoperability

Platform interoperability issues in IoT arises due to the availability of diverse operating systems (OSs), programming languages, data structures, architectures and access mechanisms for things and data.

Developers need to obtain extensive knowledge of the platform specific APIs and information models of each different platform to be able to adapt their applications from one platform to another.

Interoperability Handling Approaches in IoT

To improve the state of IoT interoperability, researchers have leveraged numerous approaches and technologies which we refer to interoperability handling approaches

1.Adapters/gateways

2.Virtual networks/ overlay-based solutions

3.Networking technologies

IP based approaches

Software-defined networking (SDN)

Network function virtualization

Fog computing

4.Service oriented architecture (SOA)

5.Open API

6.Semantic web technologies

7.Open standard

INTRODUCTION TO ARDUINO PROGRAMMING

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Open source based electronic programmable board (micro controller) and software(IDE).

Accepts analog and digital signals as input and gives desired output. No extra hardware required to load a program into the controller board.

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.

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.

Arduino simplifies the process of working with microcontrollers, but it offers some advantages

 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. .

 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

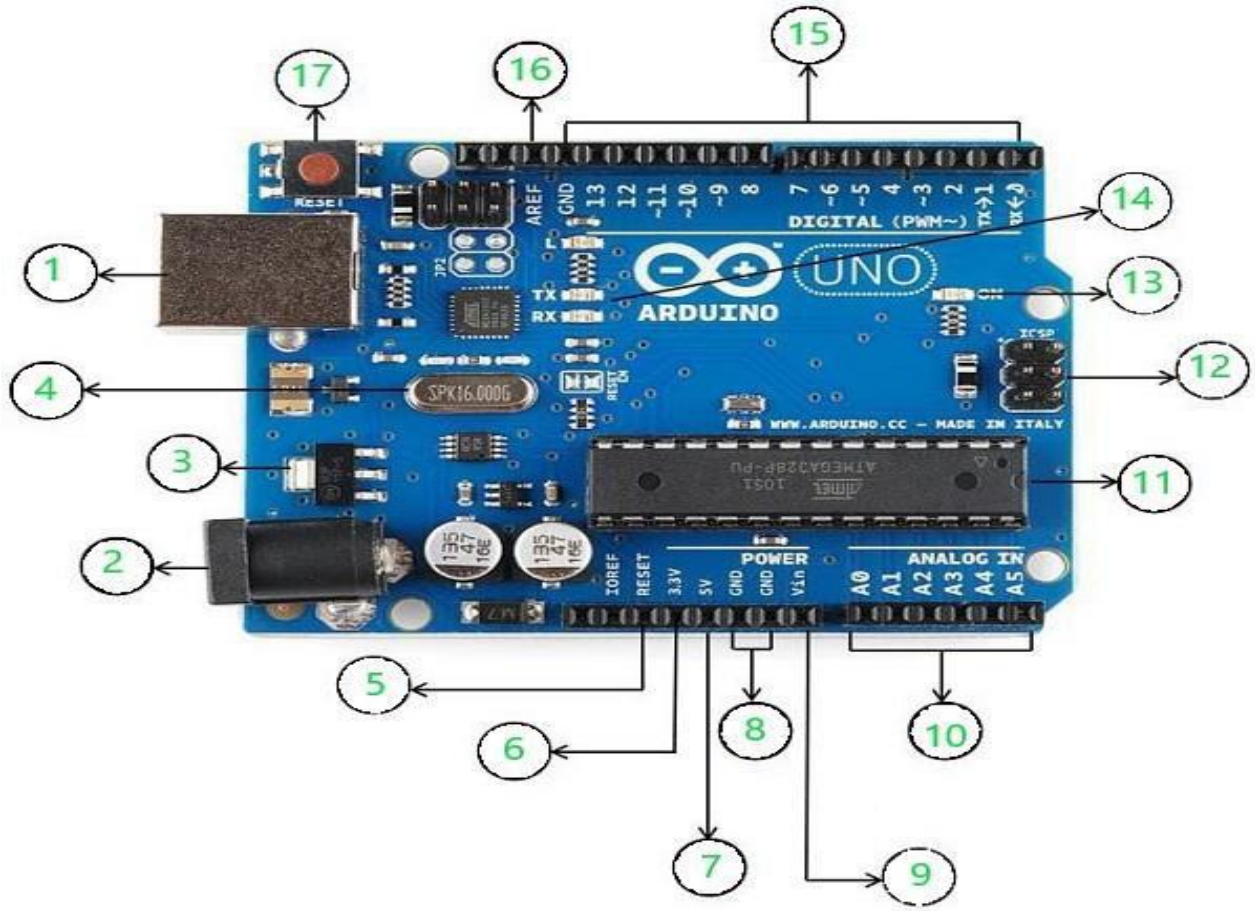
people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

 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.

Types of Arduino Boards

- Arduino boards based on ATMEGA328 microcontroller
- Arduino boards based on ATMEGA32u4 microcontroller
- Arduino boards based on ATMEGA2560 microcontroller
- Arduino boards based on AT91SAM3X8E microcontroller

AURDINO BOARD AND ITS COMPONENTS



Feature	Value
Operating Voltage	5V
Clock Speed	16MHz
Digital I/O	14
Analog Input	6
PWM	6
UART	1
Interface	USB via ATMega16U2

- Power Supply: USB or power barrel jack

- Voltage Regulator
- LED Power Indicator
- Tx-Rx LED Indicator
- Output power, Ground
- Analog Input Pins
- Digital I/O Pins

Programming with The Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus.

It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

Arduino IDE is an open source software that is used to program the Arduino controller board It can be downloaded from Arduino's official website and installed into PC.

Set Up

1. Power the board by connecting it to a PC via USB cable
2. Launch the Arduino IDE
3. Set the board type and the port for the board
4. TOOLS -> BOARD -> select your board
5. TOOLS -> PORT -> select your port

Programs written using Arduino Software (IDE) are called **sketches**.

1. These sketches are written in the text editor and are saved with the **file extension. ino**

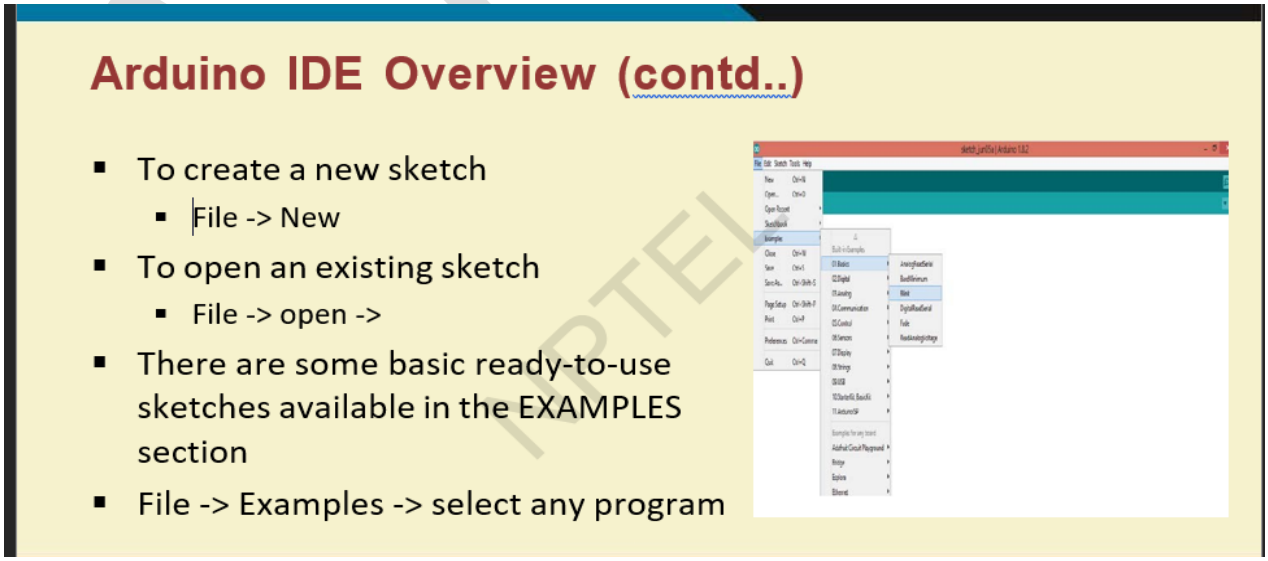
2. The editor has features for cutting/pasting and for searching/replacing text.

3.The message area gives feedback while saving and exporting and also displays errors.

4.The console displays text output by the Arduino Software (IDE), including complete error messages and other information.

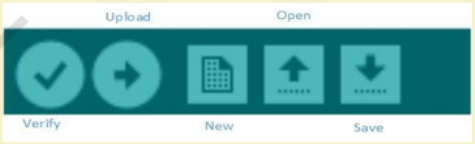
5.The bottom right-hand corner of the window displays the configured board and serial port.

6.The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.



Arduino IDE Overview (contd..)

- Verify: Checks the code for compilation errors
- Upload: Uploads the final code to the controller board
- New: Creates a new blank sketch with basic structure
- Open: Opens an existing sketch
- Save: Saves the current sketch



Arduino IDE Overview (contd..)

- Serial Monitor: Opens the serial console
- All the data printed to the console are displayed here

Sketch Structure

- A sketch can be divided into two parts:
  - Setup ()
  - Loop()
- The function setup() is the point where the code starts, just like the main() function in C and C++
- I/O Variables, pin modes are initialized in the Setup() function
- Loop() function, as the name suggests, iterates the specified task in the program

- The arduino pins can be configured to act as input or output pins using the pinMode() function

```
Void setup ()
        {
        pinMode (pin , mode);
        }
```

Pin- pin number on the Arduino board
Mode- INPUT/OUTPUT

Supported Datatype

- Arduino supports the following data types-

Void	Long
Int	Char
Boolean	Unsigned char
Byte	Unsigned int
Word	Unsigned long
Float	Double
Array	String-char array
String-object	Short

Operators: An operator is a symbol that tells the compiler to perform specific mathematical or logical functions

- Arithmetic Operators: =, +, -, *, /, %
- Comparison Operator: ==, !=, <, >, <=, >=
- Boolean Operator: &&, ||, !
- Bitwise Operator: &, |, ^, ~, <<, >>,
- Compound Operator: ++, --, +=, -=, *=, /=, %=, |=, &=

Arrays

- Collection of elements having homogenous datatype that are stored in adjacent memory location.
- The conventional starting index is 0.
- Declaration of array:

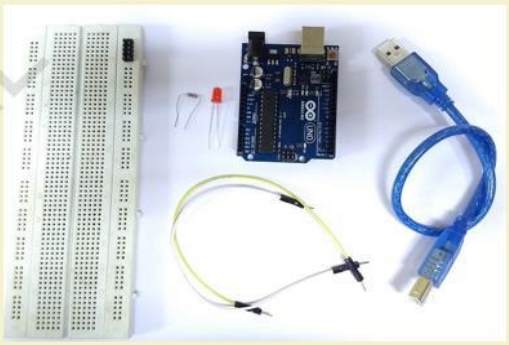
  <Datatype> array_name[size];

  Ex: int arre[5];

PROGRAMS WITH AURDINO

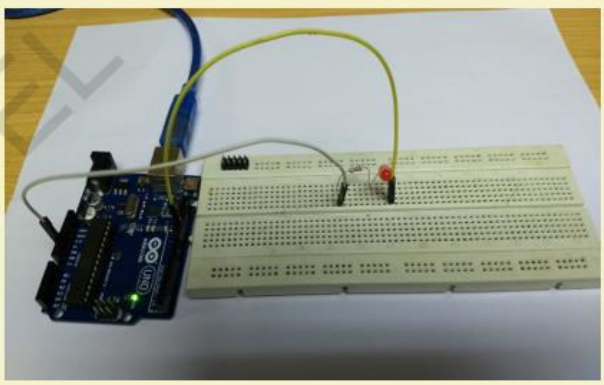
Example- Blinking LED

- Requirement:
  - Arduino controller board, USB connector, Bread board, LED, 1.4Kohm resistor, connecting wires, Arduino IDE
- Connect the LED to the Arduino using the Bread board and the connecting wires
- Connect the Arduino board to the PC using the USB connector
- Select the board type and port
- Write the sketch in the editor, verify and upload.



Example- Blink (contd..)

Connect the positive terminal of the LED to digital pin 12 and the negative terminal to the ground pin (GND) of Arduino Board



Example- Blink (contd..) image setup

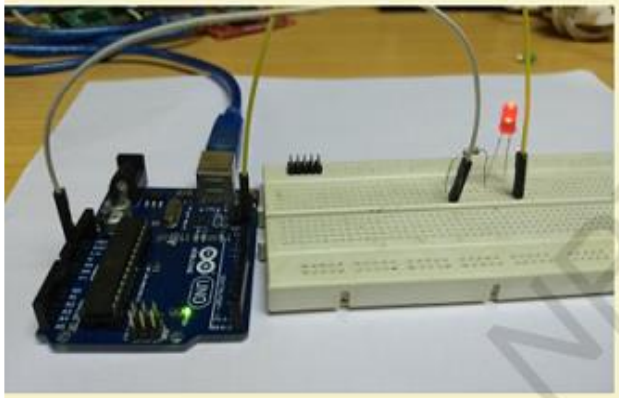
```
void setup() {
   pinMode(12, OUTPUT); // set the pin mode
}
void loop() {
   digitalWrite(12, HIGH);   // Turn on the LED
   delay(1000);
   digitalWrite(12, LOW);  //Turn of the LED
   delay(1000);
}
```

Example- Blink (contd..)

Set the pin mode as output which is connected to the led, pin 12 in this case.

Use digitalWrite() function to set the output as HIGH and LOW

Delay() function is used to specify the delay between HIGH-LOW transition of the output

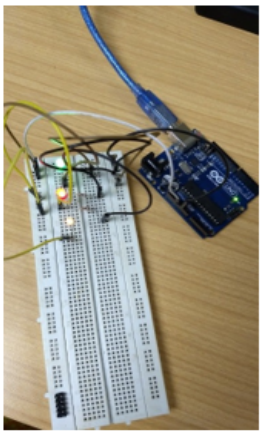


- Connect he board to the PC
- Set the port and board type
- Verify the code and upload, notice the TX – RX led in the board starts flashing as the code is uploaded.

Example: Traffic Control System

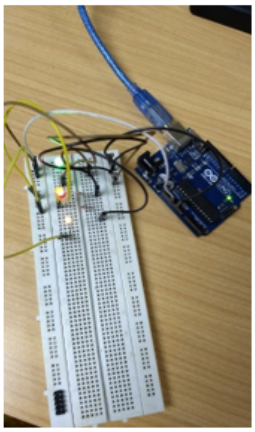
Requirement:

- Arduino Board
- 3 different color LEDs
- 330 Ohm resistors
- Jumper wires



Connection:

- Connect the positive terminals of the LEDs to the respective digital output pins in the board, assigned in the code.
- Connect the negative terminals of the LEDs to the ground



Sketch

```
//LED pins
int r = 2;
int g = 3;
int y = 4;
void setup()
{
      Serial.begin(9600);
      pinMode(r, OUTPUT); digitalWrite(r,LOW);
      pinMode(g, OUTPUT); digitalWrite(g,LOW);
      pinMode(y , OUTPUT); digitalWrite(y, LOW);
}
```

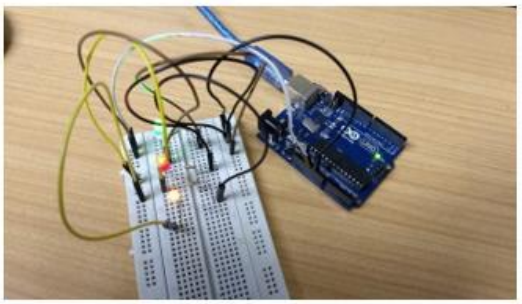
```
void traffic()
{
      digitalWrite(g, HIGH);
      Serial.println("Green LED: ON, GO");
      // delay of 5 seconds
      delay(5000);
      digitalWrite(g, LOW);
      digitalWrite(y, HIGH);
      Serial.println("Green LED: OFF ; Yellow LED: ON, WAIT");
      delay(5000);

            digitalWrite(y, LOW);
            digitalWrite(r, HIGH);
            Serial.println("Yellow LED: OFF ; Red LED: ON, STOP");
            delay(5000);                              // for 5 seconds
            digitalWrite(r, LOW);
            Serial.println("All OFF");
      }

   void loop()
   {
         traffic  ();
         delay (10000);
   }
```

Output:

- Initially, all the LEDs are turned off
- The LEDs are turned on one at a time with a delay of 5 seconds
- The message is displayed accordingly
- Figure showing all the LEDs turned on



INTEGRATING SENSORS AND ACTUATORS WITH AURDINO

SENSORS

Basic electronic Device

Convert a physical quantity/ measurements into electrical signals can be analog or digital

Types of Sensors

Some commonly used sensors:

- Temperature
- Humidity
- Compass
- Light
- Sound
- Accelerometer

Sensor Interface with Arduino

Digital Humidity and Temperature Sensor (DHT)

PIN 1,2,3,4 (from left to right)

- PIN 1-3.3V-5V Power supply
- PIN 2- Data
- PIN 3-Null
- PIN 4- Ground

DHT Sensor Library

 Arduino supports a special library for the DHT11 and DHT22 sensors

 Provides function to read the temperature and humidity values from

the data pin

dht. read Humidity ()

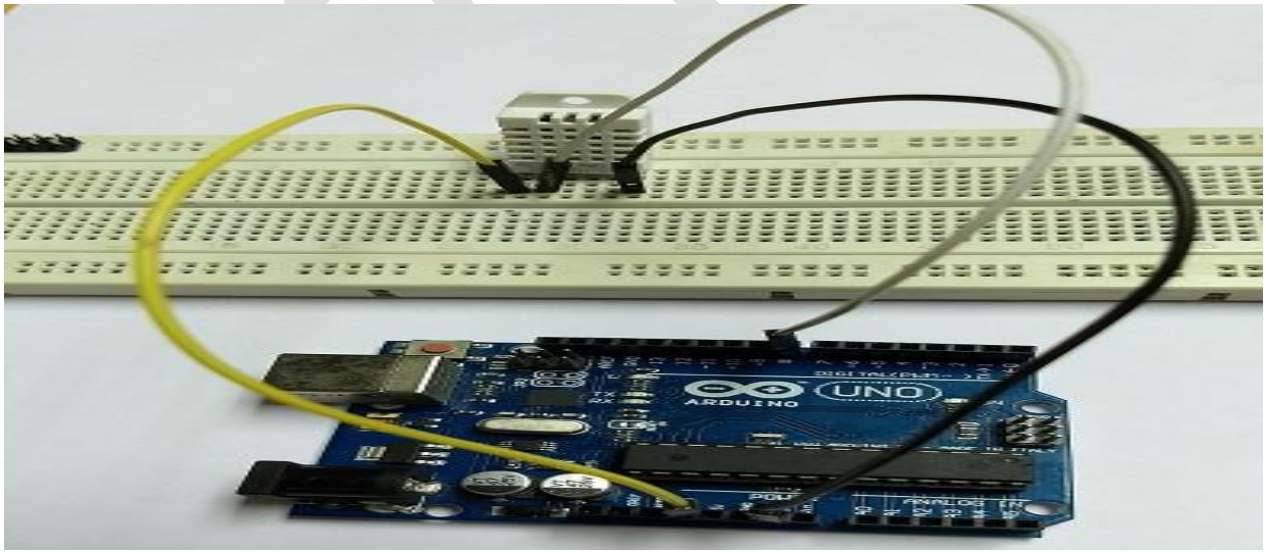
dht. read Temperature ()

Connection

 Connect pin 1 of the DHT to the 3.3 V supply pin in the board

 Data pin (pin 2) can be connected to any digital pin, here 12

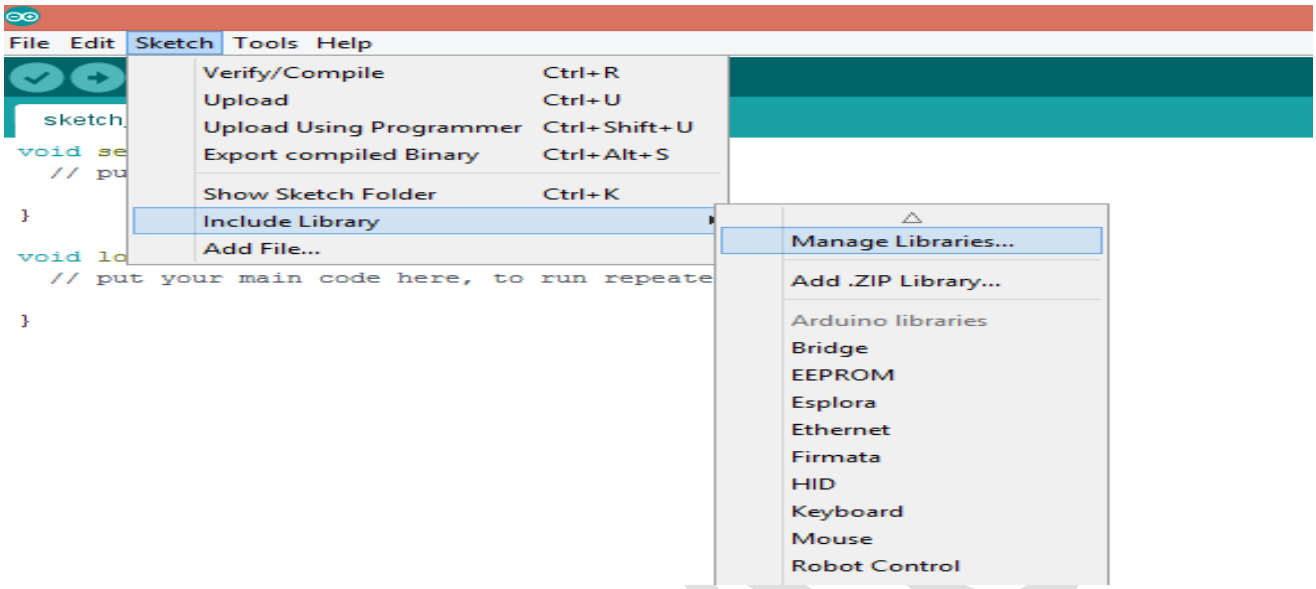
 Connect pin 4 to the ground (GND) pin of the board



Sketch: DHT_Sensor

Install the DHT Sensor Library

- Go to sketch -> Include Library -> Manage Library



Search for DHT Sensor

- Select the "DHT sensor library" and install it

```
#include <DHT.h>;
DHT dht(8, DHT22);    //Initialize DHT sensor
float humidity;       //Stores humidity value
float temperature;    //Stores temperature
value
void setup()
{
 Serial.begin(9600);
 dht.begin();
}
```

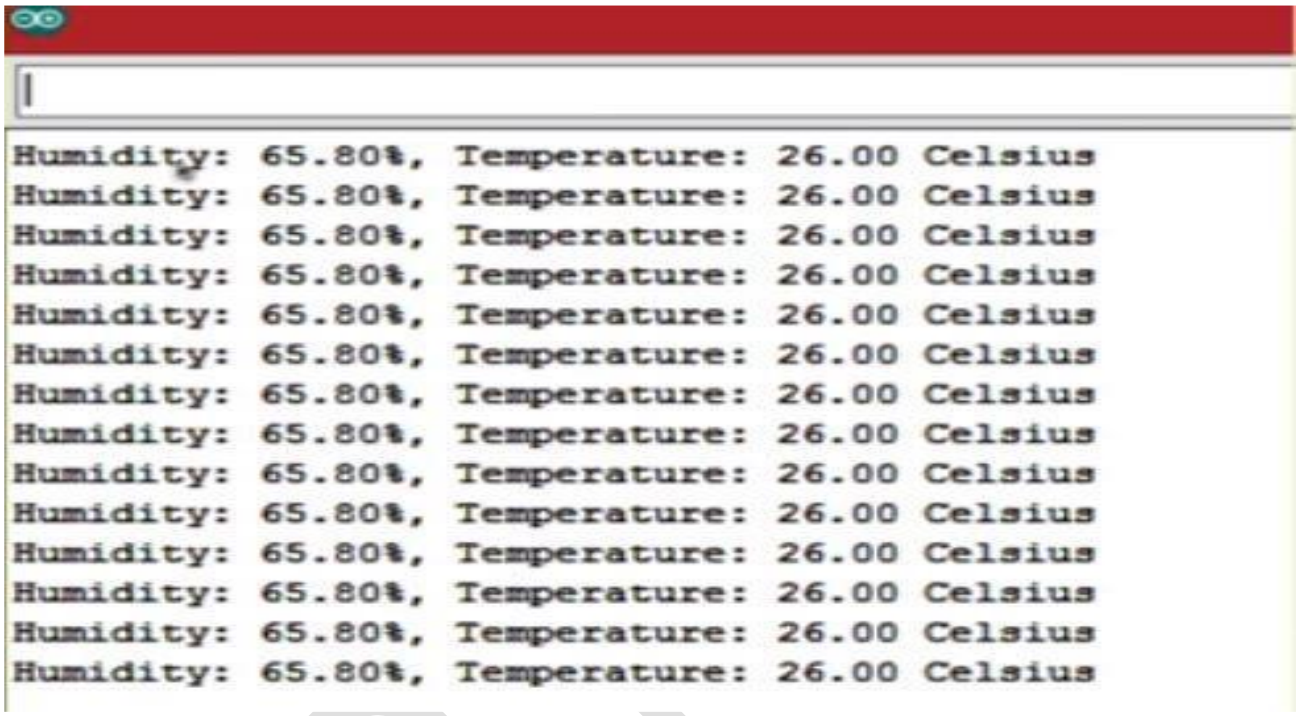
```
void loop()
{
   //Read data from the sensor and store it to variables
humidity and temperature
   humidity = dht.readHumidity();
   temperature= dht.readTemperature();
   //Print temperature and humidity values to serial
monitor
   Serial.print("Humidity: ");
   Serial.print(humidity);
   Serial.print("%, Temperature: ");
   Serial.print(temperature);
   Serial.println(" Celsius");
   delay(2000); //Delay of 2 seconds
```

Connect the board to the PC

- Set the port and board type
- Verify and upload the cod

Output

The readings are printed at a delay of 2 seconds as specified by the delay () function



ACUTATORS

Types of Motor actuators

1.Servo Motor

2.Stepper Motor

3.Hydraulic Motor

4.Ac motor and so on

Servo Motor:

Servo Motor is a high Precision motor which provides rotary motion 0 to 180 degree

The 3 wires in the servo motor are

Black or the darkest one is ground Red is power supply and Yellow is for signal pin

Servo Library on Arduino

Arduino provides separate library SERVO to operate the servo motor.

Create an instance of servo to use it in the sketch.

Syntax: Servo my servo

Sketch: SERVO_ACTUATOR

```
#include <Servo.h>
//Including the servo library for the program
int servoPin = 12;

Servo ServoDemo; // Creating a servo object
void setup() {
  // The servo pin must be attached to the servo before it can be used
  ServoDemo.attach(servoPin);
}

void loop(){
  //Servo moves to 0 degrees
  ServoDemo.write(0);
  delay(1000);

  // Servo moves to 90 degrees
  ServoDemo.write(90);
  delay(1000);

  // Servo moves to 180 degrees
  ServoDemo.write(180);
  delay(1000);
}
```

Here Servo Demo is the instance of Servo.

Write () takes the degree value and rotates the motor accordingly.

Output:

The motor turns 0, 90 and 180 degrees with a delay of 1 second each.