VIPS
Technical Campus
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

Introduction to Internet of Things (AIDS/AIML309)

By: Dr. Divya Agarwal

Course Overview:

- This course is foundation course around the Internet of Things (IoT). It overviews about the technology used to build these kinds of devices, how they communicate, how they store data, and the kinds of distributed systems needed to support them. Divided into four modules, the first unit explores about theoretical concepts of IoT while second unit is completely programming based to explore IoT sensors and actuators with Arduino. The rest of the syllabus is application oriented based on learning. In general, through this course students will be explored to the interconnection and integration of the physical world and the cyber space. They will be able to design and develop IOT Devices

Course Objective:

- To learn fundamentals of IoT and how to build IoT based systems
- To emphasize on development of Industrial IoT applications
- To recognize the factors that contributed to the emergence of IoT
- To utilize and implement solid theoretical foundation of the IoT Platform and System Design.

Course Outcomes (CO):

- CO1: Ability to understand design flow of IoT based systems
- CO2: Analyse and understand different communication protocols for connecting IoT nodes to server
- CO3: Apply coding concepts to design real-time IoT solutions
- CO4: Develop the state-of-the-art IoT based systems, suitable for real life and Industry applications

Syllabus Contents

- **UNIT- I**

❑**The Internet of Things:** An Overview of what is IoT? Why IoT? Explain the definition and usage of the term "Internet of Things (IOT)" in different contexts. Design Principles for Connected Devices, internet principles: internet communications-An overview, Physical Design of IoT, Logical Design of IoT, IoT standards, IoT generic architecture and IoT protocols. IoT future trends, Understand IoT Applications and Examples. Understand various IoT architectures based on applications. Understand different classes of sensors and actuators. Sensors: sensor terminology, sensor dynamics and specifications. Understand the basics of hardware design needed to build useful circuits using basic sensors and actuators

Syllabus Contents

- **UNIT- II**

❑**Communication protocols and Arduino Programming:** Understand various network protocols used in IoT, Understand various communication protocols (SPI, I2C, UART). Design and develop Arduino code needed to communicate the microcontroller with sensors and actuators, build circuits using IoT supported Hardware platforms such as Arduino, ESP8266 etc., Use of software libraries with an Arduino sketch that allows a programmer to use complicated hardware without dealing with complexity, Learning IoT application programming and build solutions for real life problems and test them in Arduino and Node MCU environments. Understand various wireless Technologies for IoT and its range, frequency and applications.

Syllabus Contents

- **UNIT- III**

❑**Fundamentals of IEEE 802.15.4, Zigbee and 6LOWPAN:** Importance of IEEE 802.15.4 MAC and IEEE 802.15.4 PHY layer in constrained networks and their header format, Importance of Zigbee technology and its applications, use of IPv6 in IoT Environments, Understanding importance of IPv6 and how constrained nodes deal with bigger headers (IPv6). Understand IPv6 over LowPower WPAN (6LoWPAN) and role of 6LoWPAN in wireless sensor network. Various routing techniques in constrained network. Understanding IoT Application Layer Protocols: HTTP, CoAP Message Queuing Telemetry Transport (MeTT).

Syllabus Contents

- **UNIT- IV**

❑**Application areas and Real-time Case Studies:** Role of big data, cloud computing and data analytics in a typical IoT system. Analyze various case studies implementing IoT in real world environment and find out the solutions of various deployment issues. Smart parking system, Smart irrigation system-block diagram, sensors, modules on Arduino and Node MCU

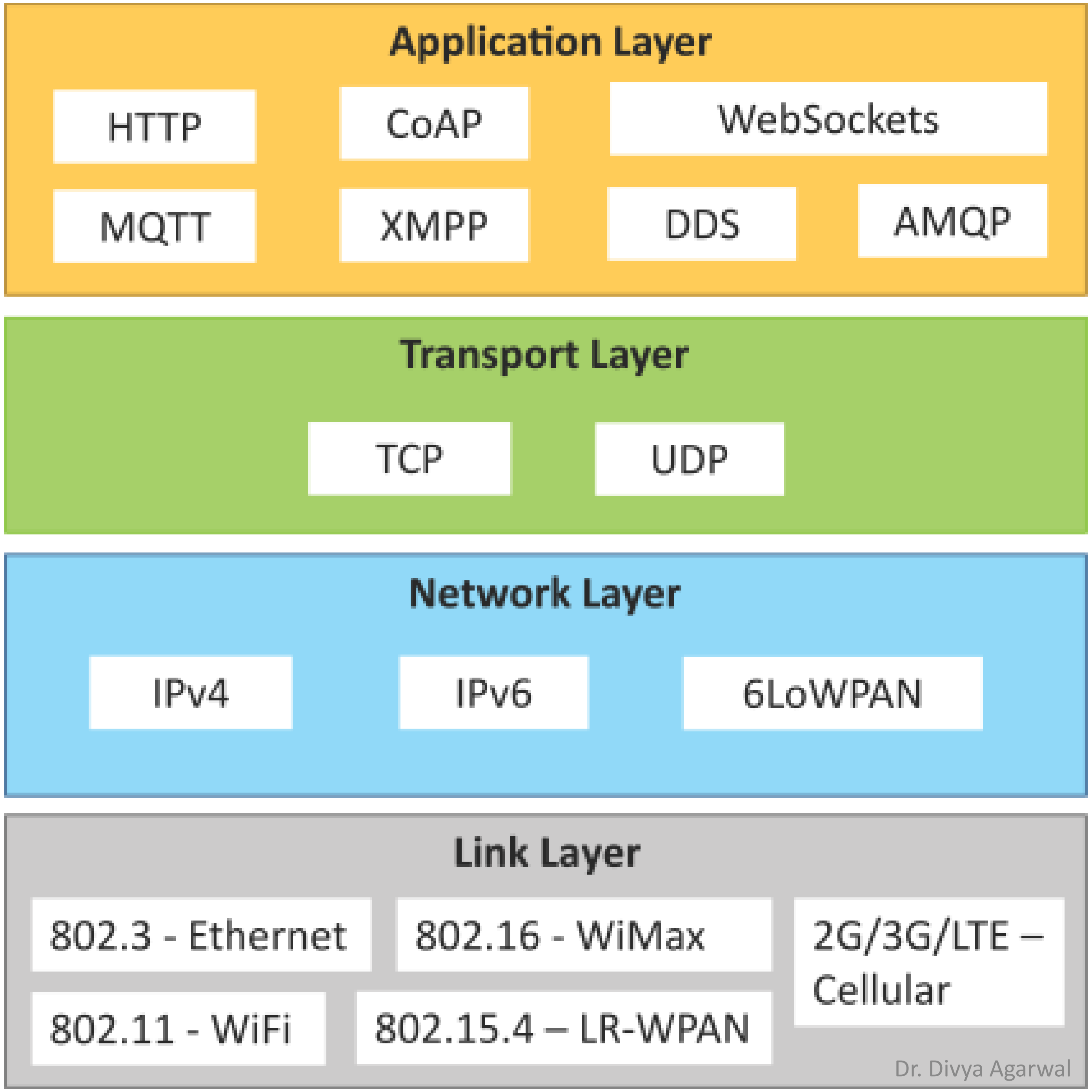
Syllabus Contents

- **UNIT- II**

❑**Communication protocols and Arduino Programming:** Understand various network protocols used in IoT, Understand various communication protocols (SPI, I2C, UART). Design and develop Arduino code needed to communicate the microcontroller with sensors and actuators, build circuits using IoT supported Hardware platforms such as Arduino, ESP8266 etc., Use of software libraries with an Arduino sketch that allows a programmer to use complicated hardware without dealing with complexity, Learning IoT application programming and build solutions for real life problems and test them in Arduino and Node MCU environments. Understand various wireless Technologies for IoT and its range, frequency and applications.

COMMUNICATION PROTOCOLS

- IoT is an aggregation of different networks, such as mobile networks (3G, 4G, CDMA, etc.), WLANs, WSN, and Mobile Adhoc Networks (MANET).
- **Requirement for IoT** - Seamless connectivity.
- **Parameters for good IoT experience:** Network-communication speed, reliability, and connection durability.
- **Challenge:** Interconnecting network of objects with high-speed mobile networks like 5G along with local and urban network communication protocols such as Wi-Fi, Bluetooth, and WiMax is feasible yet challenging



Communication Protocols in different Layers

IoT Architecture and Protocols

- **Link Layer** Determines how data is physically sent over network layer (e.g. copper wire, coaxial cable or a radio wave).

- Determines how packet are coded and signaled by hardware device over medium to which host is attached.

- Link Layer Protocols:-
  1. IEEE 802.3 - Ethernet (wired connection)
  2. 802.11 –Wi-Fi
  3. 802.16—WiMax
  4. 802.15.4 – LR-WPAN
  5. 2G/3G/4G—Mobile communication

IoT Architecture and Protocols

1. **Link Layer**.
   - Its scope is local network connection to which host is attached.
   - Hosts on same link exchange data packets over link layer using link layer protocols.

1. **IEEE 802.3 - Ethernet :** IEEE 802.3 is a collection of wired Ethernet standards for link layer. For example

- 802.3 is standard for 10BASE5 Ethernet that uses co-axial cable as a shared medium,
- 802.3.i is standard for 10BASE-T Ethernet over copper twisted pair connections,
- 802.3.j is standard for 10BASE-F Ethernet over fibre optic connections,
- 802.3ae is standard for 10Gbit/s Ethernet over fiber, etc. Standards provide data rates from 10 Mb/s to 40 Gb/s and higher. Shared medium in Ethernet can be a coaxial cable, twisted pair wire or an optical fiber which carries communication for all devices on Internet, thus data sent by one device can be received by all devices subject to propagation conditions and transceiver capabilities.

Link Layer

2. **IEEE 802.11 –Wi-Fi :** IEEE 802.11 is a collection of wireless LAN (WLAN) communication standards including extensive description of link layer. For example
   - 802.11a operates in 5Ghz band,
   - 802.11b and 802.11g operate in 2.4 GHz band,
   - 802.11n operates in 2.4/5 GHz bands,
   - 802.11 ac operates in 5GHz band and
   - 802.11ad operates in 60 GHz band. Standards provide data rates from 1 Mb/s to 6.75 Gb/s.

Link Layer

3. **IEEE 802.16—WiMax :** IEEE 802.16 is a collection of wireless broadband standards including extensive descriptions for link layer also called WiMax.

- WiMax standards provide data rates from 1.5 Mb/s to 1 Gb/s.
- Recent update provides data rates of 100 Mbit/s for mobile stations and 1Gbit/s for fixed stations.

Link Layer

4. **802.15.4 – LR-WPAN:** IEEE 802.15.4 is collection of standards for Low Rate Wireless PAN (LR-WPAN). Standards form basis of specifications for high level communication protocols such as ZigBee. LR-WPAN standards provide data rates from 40 Kb/s to 250 Kb/s. Standards provide low cost and low speed communication for power constraint devices.
5. **2G/3G/4G—Mobile communication:** There are different generations of mobile communication standards including 2G (GSM and CDMA), 3G (UMTS and CDMA 2000) and 4G (LTE), IoT devices based on these standards can communicate over cellular networks, Data rates range from 9.6 Kb/s (for 2G) to 100 Mb/s (for 4G).

IoT Protocols

SCHOOL OF ENGINEERING AND TECHNOLOGY

2. **Network/Internet Layer** -This layer performs host addressing and packet routing. Datagram contain source and destination addresses which are used to route them from source to destination across multiple networks.

   - **IPv4:** Internet protocol version 4 is most deployed internet protocol, used to identify devices on network using hierarchical addressing scheme. It uses 32 bit address scheme that allows 2^{32} addresses. It is succeeded by IPv6. IP protocols establish connections on packet networks, but do not guarantee delivery of packets. Guaranteed delivery and Data integrity are handled by upper layer protocols such as TCP.

IoT Protocols

- **IPV6:-** New version of internet protocol which uses 128-bits address that allows $2^{\wedge 128}$ or 3.4 X $10^{\wedge 38}$ address.
- **6LoWPAN:- IPv6 over Low-Power Wireless Personal Area Networks (*6LoWPAN)*** brings IP protocol to low power devices which have limited processing capabilities. Operates in 2.4 GHz frequency range and provides data transfer rates of 250 Kb/s. 6LoWPAN works with 802.15.4 link layer protocol and defines compression mechanisms for IPv6 datagrams over IEEE 802.15.4-based networks.

Transport Layer

Provide end to end message transfer capability independent of underlying network.

- Message transfer capability can be set up on connections using handshakes (as in TCP) for without using handshakes (as in UDP) . Provides functions such as error control segmentation, flow control and congestion control.

1. **Transmission Control Protocol (TCP): -** Widely used for data transmission in communication network such as internet. It is a **connection oriented** and **stateful protocol**. While IP protocols deals with sending packets, TCP ensures reliable transmission of packets in order. It provides **error detection capability** so that duplicate packets can be discarded and lost packets are retransmitted. **Flow control capability** ensures that rate at which sender sends data is not too high for receiver to process. **Congestion control** capability helps in avoiding network congestion and congestion collapse which can lead to degradation of network performance.

2. **User Datagram Protocol (UDP): -** Transaction oriented, stateless and connectionless protocol. Useful for time-sensitive applications that have very small data units to exchange and do not want connection setup-overhead. Does not provide guaranteed delivery, ordering of message and duplicate elimination.

IoT Protocols

SCHOOL OF ENGINEERING AND TECHNOLOGY

4. **Application Layer** Defines how application processes (clients and servers), running on different end systems, pass messages to each other. It defines:

- Types of messages, e.g., request messages and response messages.
- Syntax of various message types, i.e., fields in message and how fields are delineated.
- Meaning of information that field is supposed to contain.
- Rules for determining when and how a process sends messages and responds to messages

Application Type	Application-Layer Protocol
Electronic Mail	**Send:** Simple Mail Transfer Protocol (SMTP) **Receive:** Post Office Protocol v3 (POP3)
M2M	CoAP
World Wide Web (WWW)	Hyper Text Transfer Protocol 1.1 (HTTP 1.1)
File Transfer	File Transfer Protocol (FTP) Trivial File Transfer Protocol (TFTP)
Internet Telephony	Proprietary (e.g. Vocaltec)

Application Layer

4. **Application Layer** Defines how application interface with the lower layer protocols to send data over network. It enables process-to-process connections using ports.

- **HTTP:** Hypertext Transfer Protocol (HTTP) forms foundation of WWW.
  - ❑ Includes commands such as GET, PUT, POST, DELETE, HEAD, TRACE, OPTIONS, etc.
  - ❑ Follows **request response model** where client sends requests to server using HTTP commands.
  - ❑ Stateless protocol where each HTTP request is independent of other requests.
  - ❑ HTTP client can be a browser or an application running on client (e.g., an application running on an IoT device, a mobile application or other software.)
  - ❑ Uses Universal Resources Identities (URIs) to identify HTTP resources.

Application Layer

- **Constrained Application Protocol (CoAP):** used for machine-to-machine applications.
  - ❑ Meant for constrained environments with constrained devices and networks. Like, HTTP, CoAP is a web transfer protocol and uses a request-response model, however it runs on top of UDP instead of TCP.
  - ❑ Uses a client-server architecture where clients communicate with servers using connectionless datagrams.
  - ❑ Designed to easily interface with HTTP.
  - ❑ Supports methods such a GET, POST, PUT, and DELETE.

Application Layer

- **WebSocket:** allows full-duplex communications over a single socket connection for sending messages between client and server.
  - ❑ Based on TCP and allows streams of messages to be sent back and forth between client and server while keeping TCP connection open.
  - ❑ Client can be browser, mobile application or an IoT device
- **Message Queue Telemetry Transport (MQTT):** light weight messaging protocol based on publish-subscribe model.
  - ❑ Uses a client-server architecture where clients connects to server and publishes messages to topics on server.
  - ❑ Broker forwards the messages to clients subscribed to topics
  - ❑ Well suited for constrained environments where devices have limited processing and memory resources and network bandwidth is low.

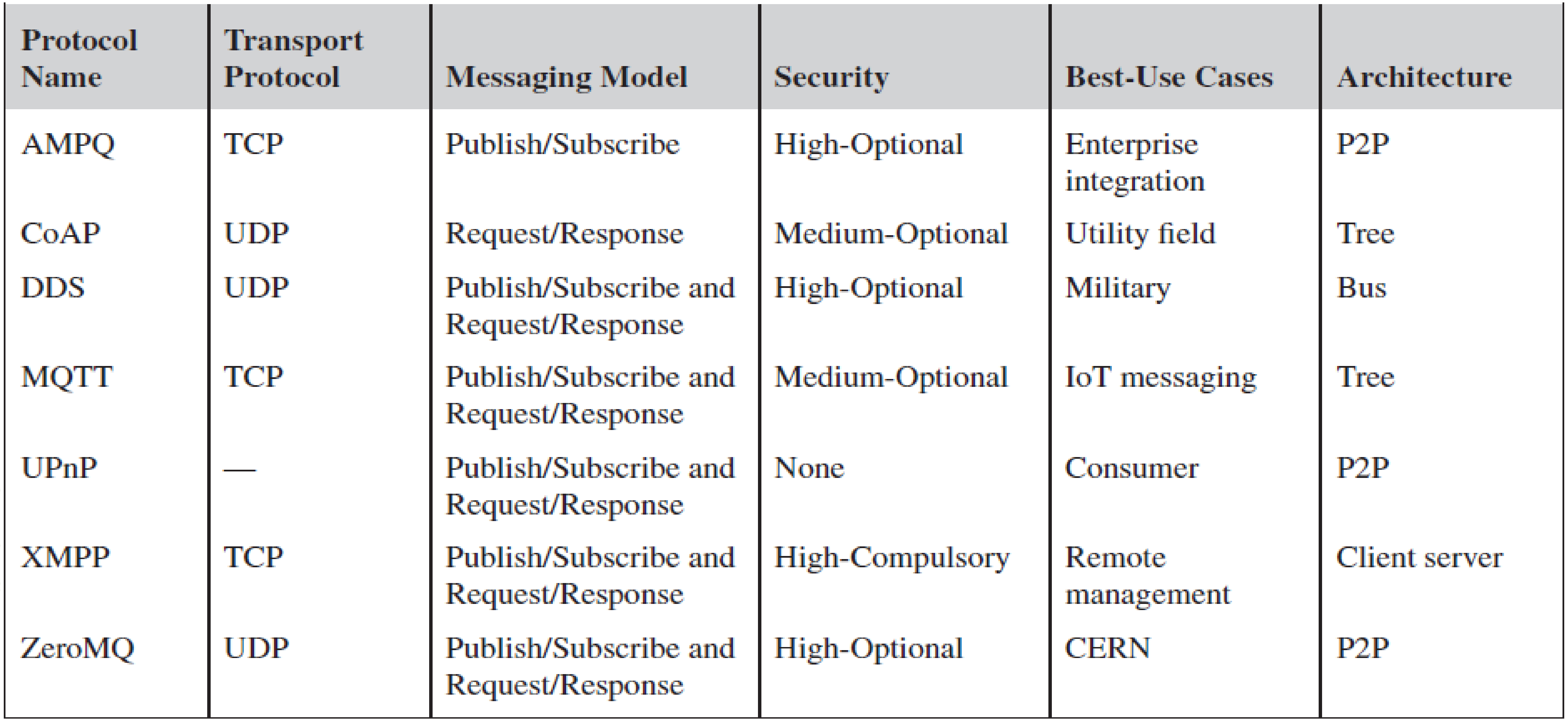
Application Layer

- **Extensible Messaging and Presence Protocol (XMPP):** Protocol for real-time communication and streaming XML data between network entities.
  - ❑ Powers wide range applications including messaging, presence, data syndication, gaming, multi-party chat and voice/video calls.
  - ❑ Allows sending small chunks of XML data from one network entity to another in near real-time.
  - ❑ Decentralized protocol and uses a client-server architecture.
  - ❑ Supports both client-to-server and server-to-server communication paths.
  - ❑ Allows real-time communication between IoT devices.

Application Layer

- **Data –Distribution Service (DDS):** Data-centric middleware standard for device-todevice or machine-to-machine communication.
  - Uses publish-subscribe model where publishers create topics which subscribers subscribe.
  - Publisher- responsible for data distribution; subscriber- responsible for receiving published data.
  - Provides quality-of-service (QoS) control and configurable reliability.
- **Advanced Message Queuing Protocol (AMQP):** Protocol for business messaging.
  - Supports both point-to-point and publisher-subscriber model, routing and queuing
  - AMQP brokers receives messages from publishers and route them to consumers.
  - Publishers publish messages to exchange which then distributes message copies to queues.
  - Messages are either delivered by broker to consumer which have subscribed to queues or consumers can pull messages from queues.

Comparison of IoT Communication Protocols

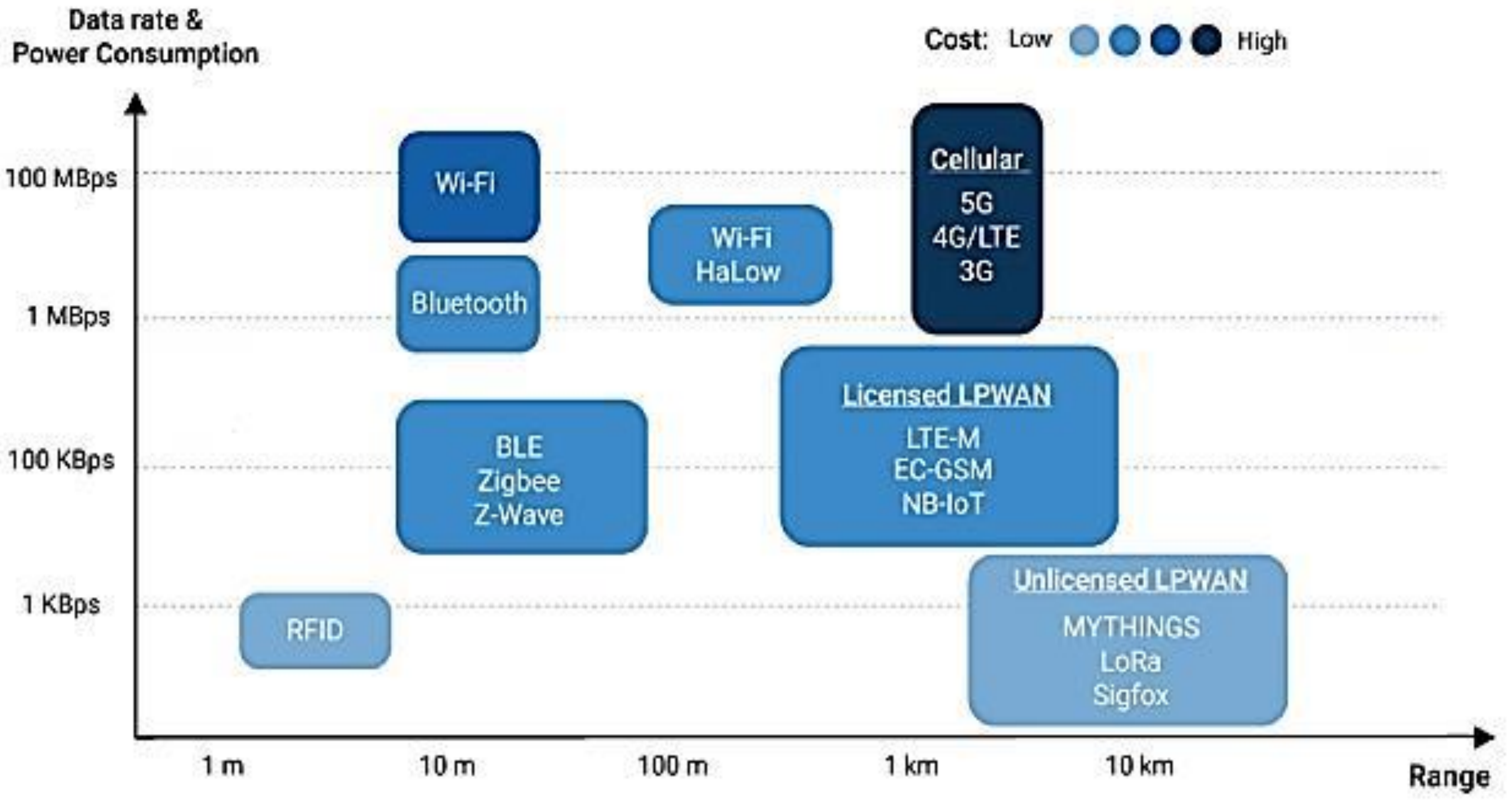


Protocol Name	Transport Protocol	Messaging Model	Security	Best-Use Cases	Architecture
AMPQ	TCP	Publish/Subscribe	High-Optional	Enterprise integration	P2P
CoAP	UDP	Request/Response	Medium-Optional	Utility field	Tree
DDS	UDP	Publish/Subscribe and Request/Response	High-Optional	Military	Bus
MQTT	TCP	Publish/Subscribe and Request/Response	Medium-Optional	IoT messaging	Tree
UPnP	—	Publish/Subscribe and Request/Response	None	Consumer	P2P
XMPP	TCP	Publish/Subscribe and Request/Response	High-Compulsory	Remote management	Client server
ZeroMQ	UDP	Publish/Subscribe and Request/Response	High-Optional	CERN	P2P

Wireless Communication Technology

SCHOOL OF ENGINEERING AND TECHNOLOGY

- Near Field Communication NFC,
- RFID,
- ZigBee,
- Bluetooth (BT),
- RF transceivers and
- RF modules.



Near-Field Communication

- NFC is enhancement of ISO/IEC214443 standard for contact-less proximity-card.
- Short distance (20 cm) wireless communication technology.
- Enables data exchange between cards in proximity and other devices.
- Examples of applications of NFC are proximity-card reader/RFID/IoT/M2M/mobile device, mobile payment wallet, electronic keys for car, house, office entry keys and biometric passport readers.
- NFC devices transmit and receive data at same instance and setup time (time taken to start communication) is 0.1 s.
- Device or its reader can generate RF fields for nearby passive devices such as passive RFID.
- An NFC device can check RF field and detect collision of transmitted signals.
- Device can check collision when received signal bits do not match with transmitted signal bits.

Near-Field Communication

Features of an NFC device are:

- Range of functioning is within 10 to 20 cm.
- Device can communicate with Bluetooth and Wi-Fi devices in order to extend distance from 10 cm to 30 m or higher.
- Device is able to receive and pass data to a Bluetooth connection or standardised LAN or Wi-Fi using information handover functions.
- Device data transfer rates are 106 kbps, 212 kbps, 424 kbps and 848 kbps (bps stands for bit per second, kbps for kilo bit per second).

Near-Field Communication

Three modes of communication are:

1. **Point-to-point (P2P) mode:** Both devices use active devices in which RF fields alternately generate when communicating.
2. **Card-emulation mode:** Communication without interruption for read and write as required in a smart card and smart card reader. FeliCa™ and Mifare™ standards are protocols for reading and writing data on card device and reader, and then reader can transfer information to Bluetooth or LAN.
3. **Reader mode:** Using NFC the device reads passive RFID device. RF field is generated by an active NFC device. This enables passive device to communicate.

Radio Frequency Identification (RFID)

- RFID is an automatic identification method.
- Use Internet.
- RFID usage is, therefore, in remote storage and retrieval of data is done at RFID tags. An RFID device functions as a tag or label, which may be placed on an object.
- Object can then be tracked for movements.
- Object may be a parcel, person, bird or an animal.
- IoT applications of RFID are in business processes, such as parcels, tracking and inventory control, sales log-ins and supply-chain management.

Bluetooth BR/EDR and Bluetooth Low Energy

- Bluetooth devices follow IEEE 802.15.1 standard protocol for L1 (physical cum data-link layer).
- BT devices form a WPAN devices network.
- **Modes:** Bluetooth BR/EDR (Basic Rate 1 Mbps/Enhanced Data Rate 2 Mbps and 3 Mbps) and Bluetooth low energy (BT LE 1Mbps).
- Latest version is Bluetooth v4.2.
- BT LE is called Bluetooth Smart.
- Bluetooth v4.2 (December 2014) provides LE data packet length extension, link layer privacy and secure connections, extended scanner and filter link layer policies and IPSP.
- BT LE range is 150 m at 10 mW power output, data transfer rate is 1 Mbps and setup time is less than 6 s.
- Bluetooth v5, released in June 2016, has increased the broadcast capacity by 800%, quadrupled the range and doubled the speed.
- A device may have provisions for single mode BT LE or dual mode BT BR/EDR (Mbps stands for Million Bits per second).

Bluetooth BR/EDR and Bluetooth Low Energy

Its features are:

- Auto-synchronisation between mobile and other devices when both use BT.
- BT network uses features of self-discovery, self-configuration and self-healing.
- Radio range depending on class of radio; Class 1 or 2 or radios: 100 m, 10 m or 1 m used in device BT implementation.
- Support to NFC pairing for low latency in pairing the BT devices.
- Two modes—dual or single mode devices are used for IoT/M2M devices local area network.
- IPv6 connection option for BT Smart with IPSP (Internet Protocol Support Profile).
- Smaller packets in LE mode.
- Operation in secured as well as unsecured modes (devices can opt for both link-level as well as service-level security or just service level or unsecured level).
- Advanced Encryption Algorithm cryptographic block cipher based on symmetric 128-bit block data (AES-CCM 128) authenticated encryption algorithm for confidentiality and authentication.
- Connection of IoT/M2M/mobile devices using BT EDR device to Internet with 24 Mbps Wi-Fi 802.11 adaptation layer (AMP: Alternative MAC/PHY layer) or BT-enabled wire-bound connection ports or device.

ZigBee IP/ZigBee SE 2.0

- ZigBee devices follow **IEEE 802.15.4 standard protocol** L1 (physical cum data-link layer).
- ZigBee devices form a **WPAN devices network**.
- ZigBee end-point devices form a WPAN of embedded sensors, actuators, appliances, controllers or medical data systems which connect to Internet for IoT applications, services and business processes.
- ZigBee Neighbourhood Area Network (NAN) is a version for a smart grid.
- ZigBee smart energy version 2.0 has energy management and energy efficiency capabilities using an IP network.

Features of ZigBee IP are:

- ❑ Used for low-power, short-range WPAN
- ❑ Can function in six modes—end point, ZigBee-ZigBee devices router, ZigBee network coordinator, ZigBee-IP coordinator, ZigBee-IP router and IP host.
- ❑ ZigBee IP enhancement provisions IPv6 connectivity.
- ❑ Is a Reduced Function Device (RFD) i.e., one that functions for 'sleepy'/ battery-operated device. Sleepy means one that wakes up infrequently, sends data and then goes back to sleep.
- ❑ Supports IPv6 network with 6LoWPAN header compression, connection for Internet communication and control of low power devices, TCP/UDP transport layer and TLSv1.2 public key (RSA and ECC) and PSK cipher suite for end-to-end security protocol, end-to-end means application layer to physical layer.
- ❑ ZigBee router uses reactive and proactive protocols for routing mode, which enable applications in big-scale automation and remote controls.
- ❑ A self-configuring and self-healing dynamic pairing mesh network, supports both multicast and unicast options.

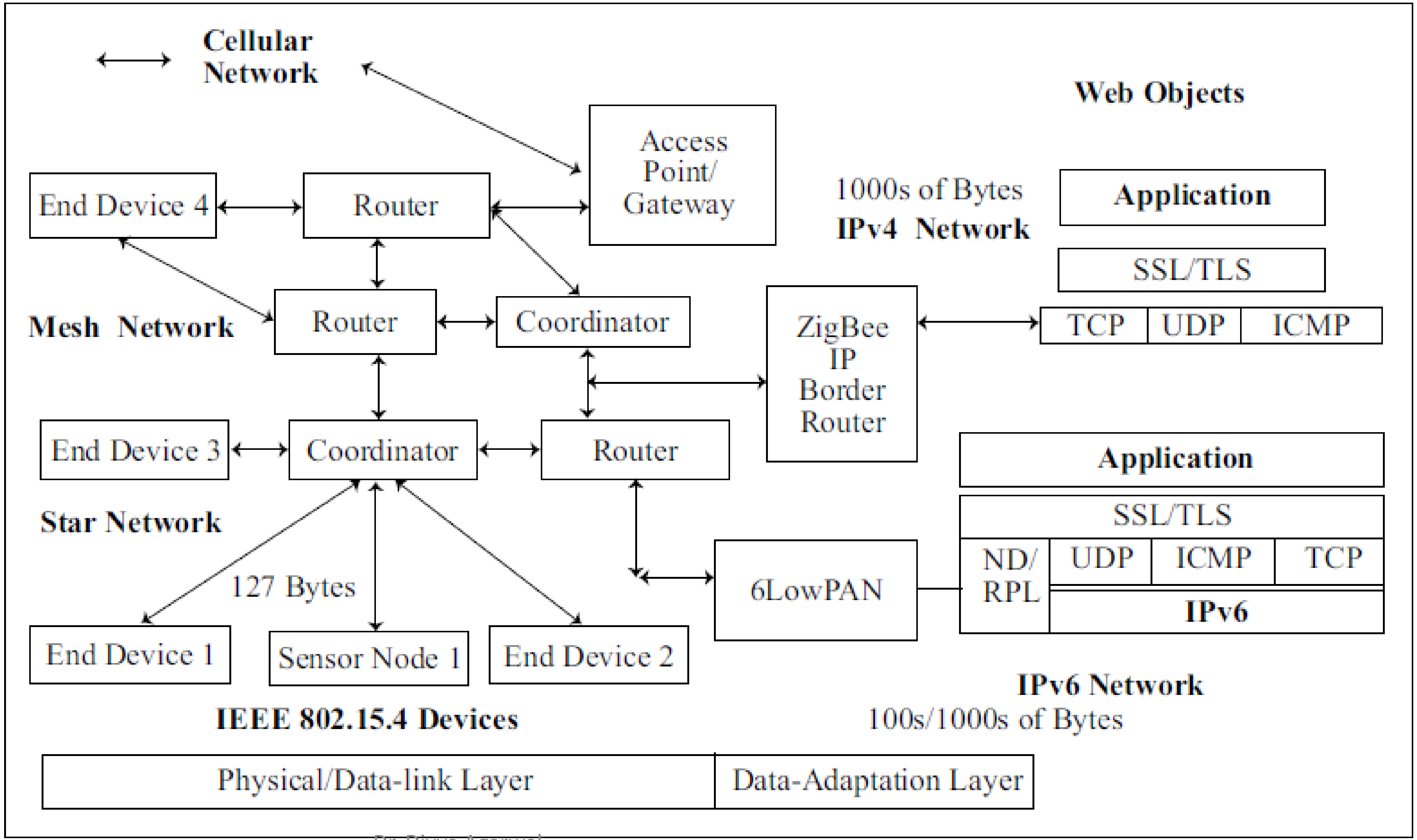
Features of ZigBee IP are:

- ❑ Multicast forwarding to support multicast Domain Name System (mDNS) based service discovery (SD)
- ❑ Support to pairing of coordinator with end-point devices and routers, providing bigger network using multiple star topology and inter-PAN communications
- ❑ Support to sensor nodes and sensor (or appliances) network integration, sensor and appliances devices configured as router or end-devices
- ❑ Low latency (< 10 ms) link layer connection
- ❑ Range is 10–200 m, data transfer rate is 250 kbps, low power operation
- ❑ ISM band frequencies direct sequence spread spectrum 16-channel radio, and provide link level security using AES-CCM-128
- ❑ Includes RFD in ZigBee SE 2.0
- ❑ ZigBee NAN is for devices which are used for smart-metering, distribution automation devices and smart grid communication profile. NAN enables a utility’s last-mile at HAN (Home Area Network), outdoor access network that connects smart meters to WAN (wide area network) gateways.

Features of ZigBee IP are:

Figure shows

- Three end devices, two routers, one sensor node connected to coordinator ZigBee devices forming a star network.
- One end device, two routers and one coordinator forming a mesh network.
- Mesh network router connects to an AP/gateway, which in turn connects to a cellular network.



Features of a ZigBee network are:

- ❑ Router in star network connects to 6LoWPAN, which connects an IEEE 802.15.4 devices network to IPv6 network.
- ❑ 1000s of byte communicate between the network layer and IoT web objects.
- ❑ 127 B communication between the adaptation layer IEEE 802.15.4 devices at single data transfer.
- ❑ IETF ND (Neighbour Discovery), ROLL (Routing Over Low power Loss Network), RPL routing, IPv6/IPv4 network, TCP/UDP/ICMP transport, SSL/TLS security layer protocols for communication between web object/application and ZigBee devices.

Wi-Fi

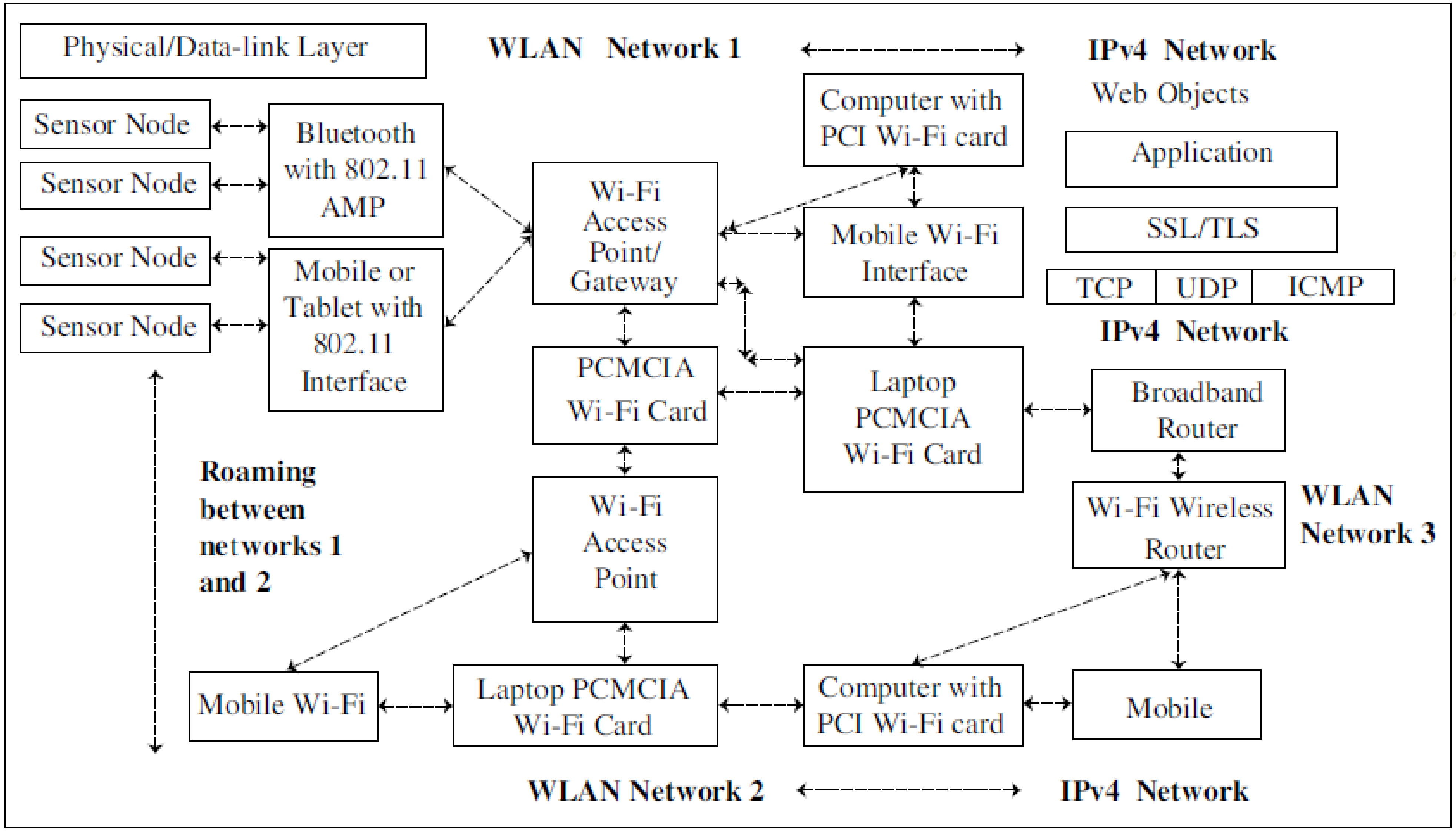
- An interface technology that uses IEEE 802.11 protocol and enables WLANs.
- Wi-Fi devices connect enterprises, universities and offices through home AP/public hotspots.
- Wi-Fi connects distributed WLAN networks using Internet.
- Automobiles, instruments, home networking, sensors, actuators, industrial device nodes, computers, tablets, mobiles, printers and many devices have Wi-Fi interface.
- **Issues of Wi-Fi interfaces APs and routers:** are higher power consumption, interference and performance degradation.
- Wi-Fi interfaces connect within themselves or to an AP or wireless router using Wi-Fi PCMCIA or PCI card or built-in circuit cards and through: **Base station (BS) or AP**
- WLAN transceiver or BS can connect one/many wireless devices simultaneously to Internet.

Wi-Fi

- Peer-to-peer nodes without access point: Client devices within Independent Basic Service Set (IBSS) network can communicate directly with each other. It enables fast and easy setting of an 802.11 network.
- Peer to multipoint nodes with Basic Service Sets (BSSs) using one in-between AP point or distributed BSSs connect through multiple APs.
- Each BSS is a Service Set Identifier (SSID)

Wi-Fi

Figure shows three WLAN networks (BSSs) for sensor device nodes, mobiles, tablets, laptops, computers and Internet connectivity of WLAN networks with IP4 networks (here dashed lines represent wireless connectivity and solid lines represent wired connectivity).



Wi-Fi

Wi-Fi interfaces, access points, routers features are as follows:

- Generally used are 2.4 GHz IEEE 802.11b adapters or 5 GHz (802.11a or 802.11g) or 802.11n or other 802.11 series protocols.
- Interfaces use 2.4 GHz or 5 GHz antenna
- Offers mobility and roaming
- Have easy installation simplicity and flexibility
- Coverage range is 30 m to 125 m
- Used in room having limited-coverage 802.11a which coexists with b, coexists with b and g
- Uses **802.11b** in wider coverage range because that is unaffected by walls and is meant for hotspots for public usage having range data rate 11 Mbps (802.11b) within 30 m
- Uses **802.11g** for high data rates up to 54 Mbps, and 802.11n for very high 600 Mbps, using multiple antennas to increase data rates
- Interoperable with wireless as well as wired infrastructure which ensures compatibility and enables easier access and hides complexity when enabling the wireless access to data, media and streams, and applications and services.
- Provides security, integrity and reliability

RF Transceivers and RF Modules

- RF transmitters, receivers, and transceivers are simplest RF circuits.
- Transceiver transmits RF from one end and receives from other end, but internally has an additional circuit, which separates signals from both ends.
- An oscillator generates RF pulses of required active duty cycle and connects to a transmitter.
- BT, ZigBee, and Wi- Fi radios deploy ISM band transceivers
- IoT/M2M applications deploy ISM band RF modules with transceivers/ transmitter/ receiver.
- A number of systems use RF modules for applications needing wireless connectivity; for example, security, telemetry, telematics, fleet management, home automation, healthcare, automobiles wireless tire pressure monitors, back-up cameras and GPS navigation service, payment wallet, RFID and maintenance.

Elements of RF technology

- **RF interface/physical layer**, RF signals transmit between nodes or endpoints, i.e. sensors, actuators, controllers and a gateway where signals are received. Physical layer specifications consist of signal aspects and characteristics, including frequencies, modulation format, power levels, transmitting and receiving mode and signalling between end-point elements.
- **RF network architecture** includes overall system architecture, backhaul, server and bidirectional end-devices with radio duty cycling in applications. Radio duty cycling means managing active intervals, transmission and receiving schedule, and time-intervals actions on an event during active intervals and actions during inactive (sleep) intervals using RF Integrated Circuits (RFICs).

GPRS/GSM Cellular Networks-Mobile Internet

- IoT/M2M communication gateway can access a Wireless WAN (WWAN).
- Network access may use a GPRS cellular network or new generation cellular network for Internet access.
- Mobile phone provisions for a USB wired port, BT and Wi-Fi connectivity.
- Wireless connectivity for Internet uses data connectivity using GSM, GPRS, UMTS/LTE and WiMax services of a mobile service provider or Wi-Fi using a modem.
- Phone, provisions for number of sensors also; for example, acceleration, GPS and proximity

Wireless USB

- Wireless USB is a wireless extension of USB 2.0 and it operates at ultra-wide band (UWB) 5.1 GHz to 10.6 GHz frequencies.
- It is for short-range personal area network (high speed 480 Mbps 3 m or 110 Mbps 10 m channel).
- FCC recommends a host wire adapter (HWA) and a device wire adapter (DWA), which provides wireless USB solution.
- Wireless USB also supports dual-role devices (DRDs).
- Device can be a USB device as well as limited capability host.

Differences between NFC, BT LE, ZigBee and WLAN protocols

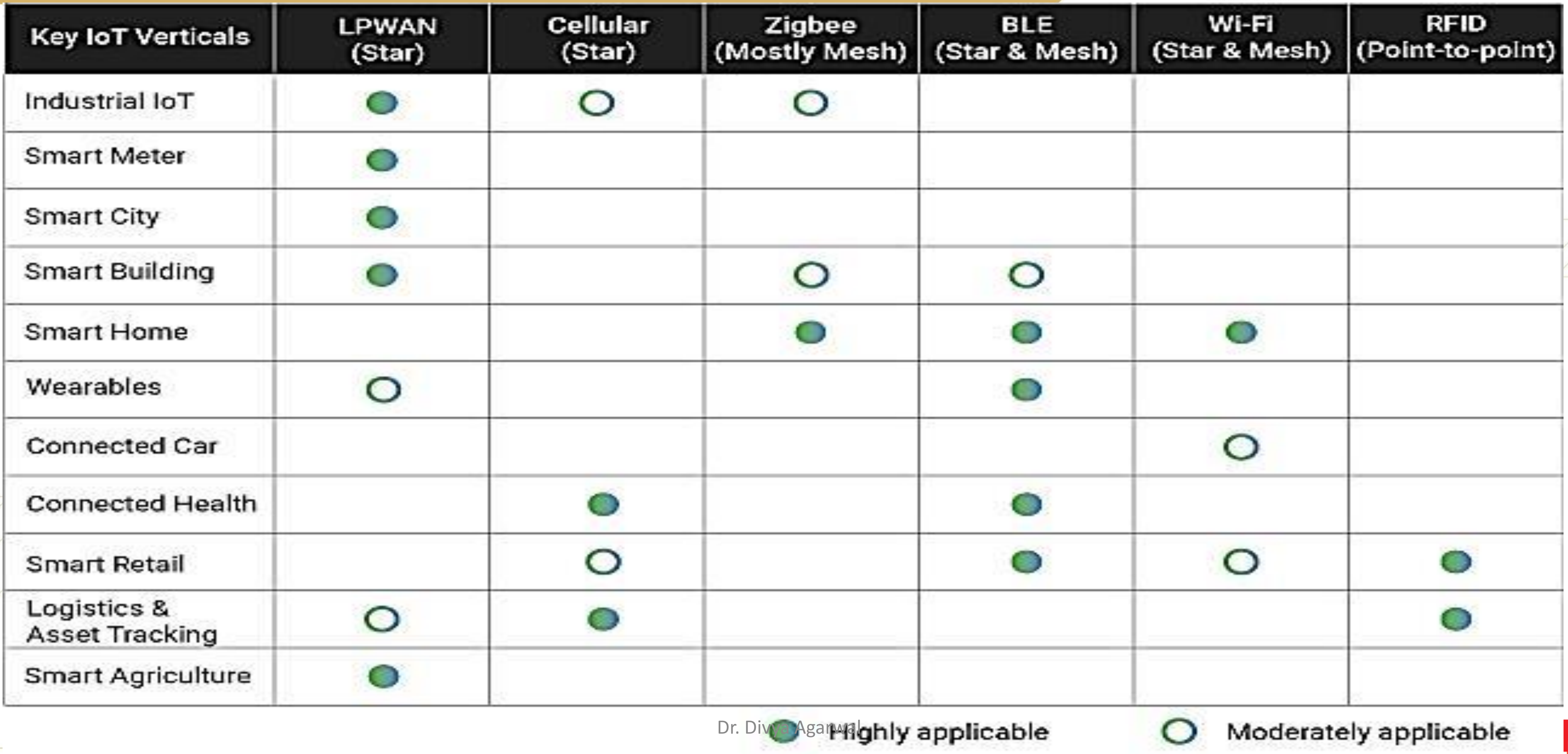
Property	NFC	BT LE	ZigBee IP	WLAN 802.11
IEEE Protocol		802.15.1	802.15.4	802.11z
Physical Layer	848, 424, 212, 106 kbps	2.4 GHz (LE-DSSS)	2.4 GHz or 915 MHz, 868MHz and 433 MHz DSSS MAC layer CSMA/CA	2.4 GHz Two PHY layers MAC layer CSMA/CD
Data Transfer Rate	106 kbps	1 Mbps	250 kbps (2.4 GHz, 40 kbps 915 MHz, 20 kbps 868.3 MHz	11 Mbps/54 Mbps
Form Factor and Range	10–20 cm	Small	Small 10 m to 200 m	Bigger
Protocol Stack		Small in LE	127B	Bigger than WPAN devices
Power Dissipation	Very low	Lower than ZigBee, much lower than WLAN 802.11	2 mW Router and 0.1 mW for end device; much lower than WLAN 802.11	Much Higher than ZigBee
Set up/ Connection/ Disconnection Intervals	0.1s	3s Connection time < 3 ms	20 ms Connection time < 10 ms	
Security		AES-CCM-128	AES-CCM-128	WEP

Differences between NFC, BT LE, ZigBee and WLAN protocols

SCHOOL OF ENGINEERING AND TECHNOLOGY

Property	NFC	BT LE	ZigBee IP	WLAN 802.11
Applications	Payment wallet, short Distance Communication	WPAN, IoT/M2M devices, widely present in mobiles and tablets and, need addition circuit in sensors, actuators, controllers and IoT Devices	WPAN, wider presence in sensors, actuators, controllers, automobile and medical electronic and IoT devices connectivity using IPv6, 6loWPAN, ROLL, RPL and TLSv1.2	WLAN and WWAN network tablet, desktops, mobiles, devices with PCMCIA interface, home networking, Easy IPv4 connectivity
Network	Point to point between active and Passive Devices	Star topology, peer-to-peer piconet expended by interpiconets data transactions and Synchronisation	Low power, mesh or peer-to peer star networks using end devices, coordinator, router, ZigBee IP border router	LAN topology IBSS, BSS and distributed BSSs for WWLAN widely used for Internet connectivity of mobiles, tablets, Desktops
Network Characteristics	P2P, card emulation and reader mode passive neighbour Activation	Self-configuring, self-healing, self-discovery	Self-configuring, self-healing, self-discovery	Scalable, interoperability, security, integrity and reliability
Broadcast/ Multicast/ Unicast	Unicast	Unicast	Unicast/ Multicast	Unicast

Differences between NFC, BT LE, ZigBee and WLAN protocols



Key IoT Verticals	LPWAN (Star)	Cellular (Star)	Zigbee (Mostly Mesh)	BLE (Star & Mesh)	Wi-Fi (Star & Mesh)	RFID (Point-to-point)
Industrial IoT	●	○	○			
Smart Meter	●					
Smart City	●					
Smart Building	●		○	○		
Smart Home			●	●	●	
Wearables	○			●		
Connected Car					○	
Connected Health		●		●		
Smart Retail		○		●	○	●
Logistics & Asset Tracking	○	●				●
Smart Agriculture	●					

● Highly applicable ○ Moderately applicable

Serial Communication Basics

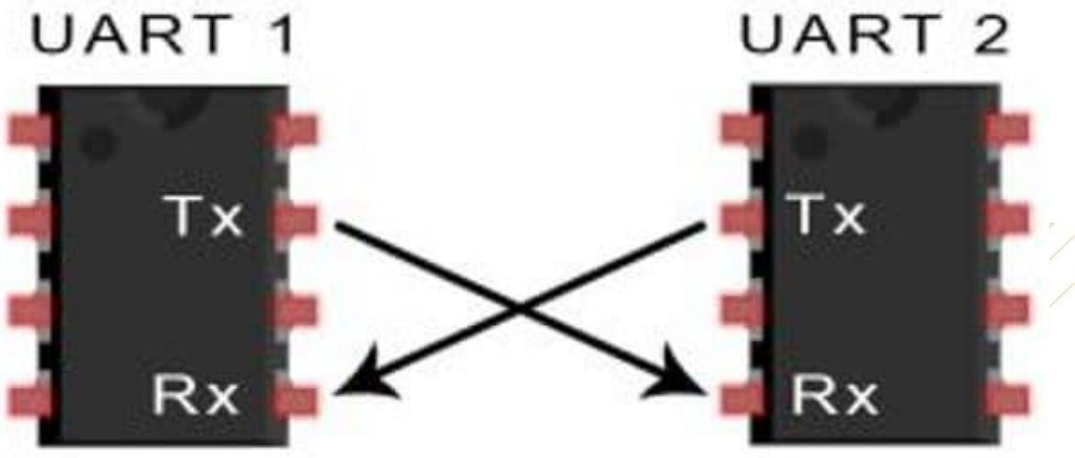
- Communication between electronic devices is like communication between humans.
- Both sides need to speak same language.
- In electronics, these languages are called communication protocols.
- Types of communication protocols are SPI,I2C,UART,USB.
- SPI, I2C, and UART are quite a bit slower than protocols like USB, Ethernet, Bluetooth, and Wi-Fi, but they're a lot simpler and use less hardware and system resources.
- **SPI, I2C, and UART** are ideal for communication between microcontrollers and between microcontrollers and sensors where large amounts of high speed data don't need to be transferred.

Serial Communication Basics

- Serial communication protocols can be categorized as Synchronous and Asynchronous protocols.
- In synchronous communication, data transmission and receiving is a continuous stream at a constant rate.
- Synchronous communication requires clock of transmitting device and receiving device synchronized.
- Examples of synchronous communication are: I2C, SPI etc.
- In case of asynchronous communication, transmission of data requires no clock signal and data transfer occurs intermittently rather than steady stream.
- Handshake signals between transmitter and receiver are important in asynchronous communications.
- Examples of asynchronous communication are Universal Asynchronous Receiver Transmitter (UART), USB, CAN etc.
- Synchronous and asynchronous communication protocols are well-defined standards and can be implemented in either hardware or software.

UART COMMUNICATION PROTOCOL

- Stands for **Universal Asynchronous Reception and Transmission (UART)**
- Allows host to communicate with auxiliary device.
- UART supports bi-directional, asynchronous and serial data transmission.
- Has two data lines, one to transmit (T_X) and another to receive (R_X), which are used to communicate through digital pin 0, digital pin 1.



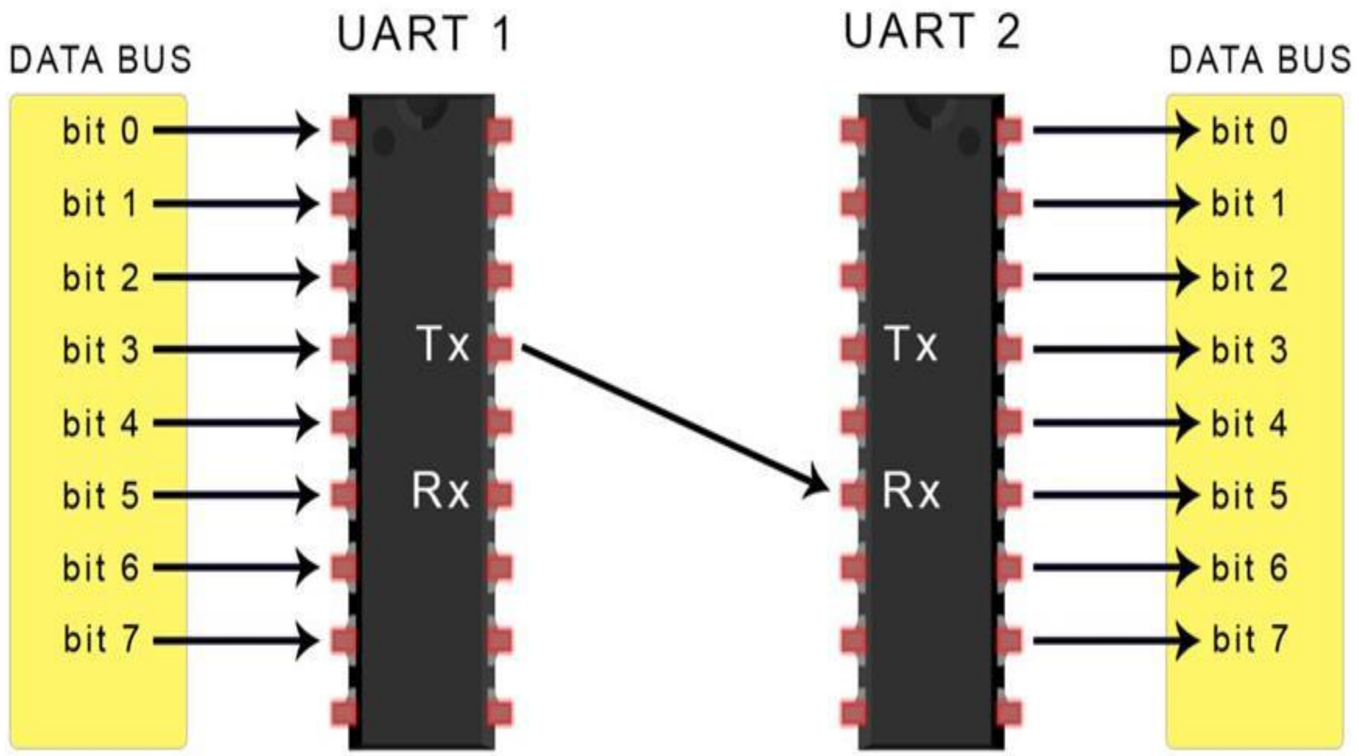
- T_X and R_X are connected between two devices. (eg. USB and computer)
- Can handle synchronization management issues between computers and external serial devices.
- UART can operate between devices in 3 ways:
  - ❑ **Simplex =** data transmission in one direction
  - ❑ **Half-duplex =** data transmission in either direction but not simultaneously
  - ❑ **Full-duplex =** data transmission in both directions simultaneously

UART COMMUNICATION PROTOCOL

- UART data transmission speed is referred to as BAUD Rate and is set to 115,200 by default
- Both UARTs must operate at about same baud rate. If difference of BAUD rate is more than 10%, timing of bits may be off and render data unusable.
- User must ensure UARTs are configured to transmit and receive from same data packet.
- Hardware for UART can be a circuit integrated on microcontroller or a dedicated IC which is in contrast to SPI or I2C, which are just communication protocols.
- UART is most simple and commonly used Serial Communication techniques.
- UART is being used in many applications like GPS Receivers, Bluetooth Modules, GSM and GPRS Modems, Wireless Communication Systems, RFID based applications etc.

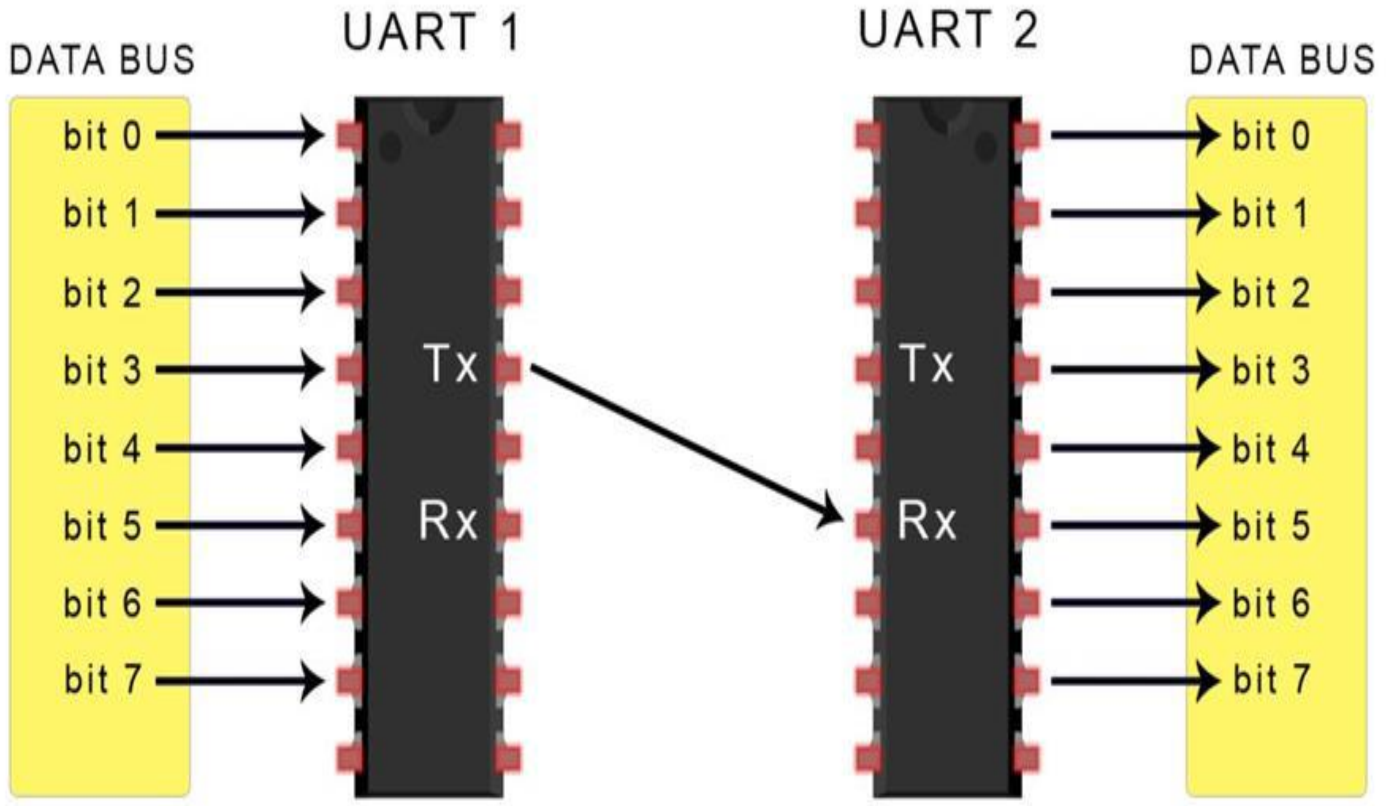
HOW UART WORKS

- Once connected, data flows from T_X of transmitting UART to R_X of receiving UART.
- **UART requires No clocks**
- UART that transmits data receives data from a data bus.
- Data bus sends data to UART by devices like CPU, memory, or microcontroller.
- Data is transferred from data bus to transmitting UART in parallel form.
- After transmitting, UART gets parallel data from data bus, it adds a start bit, a parity bit, and a stop bit, creating data packet as it has No CLOCK.

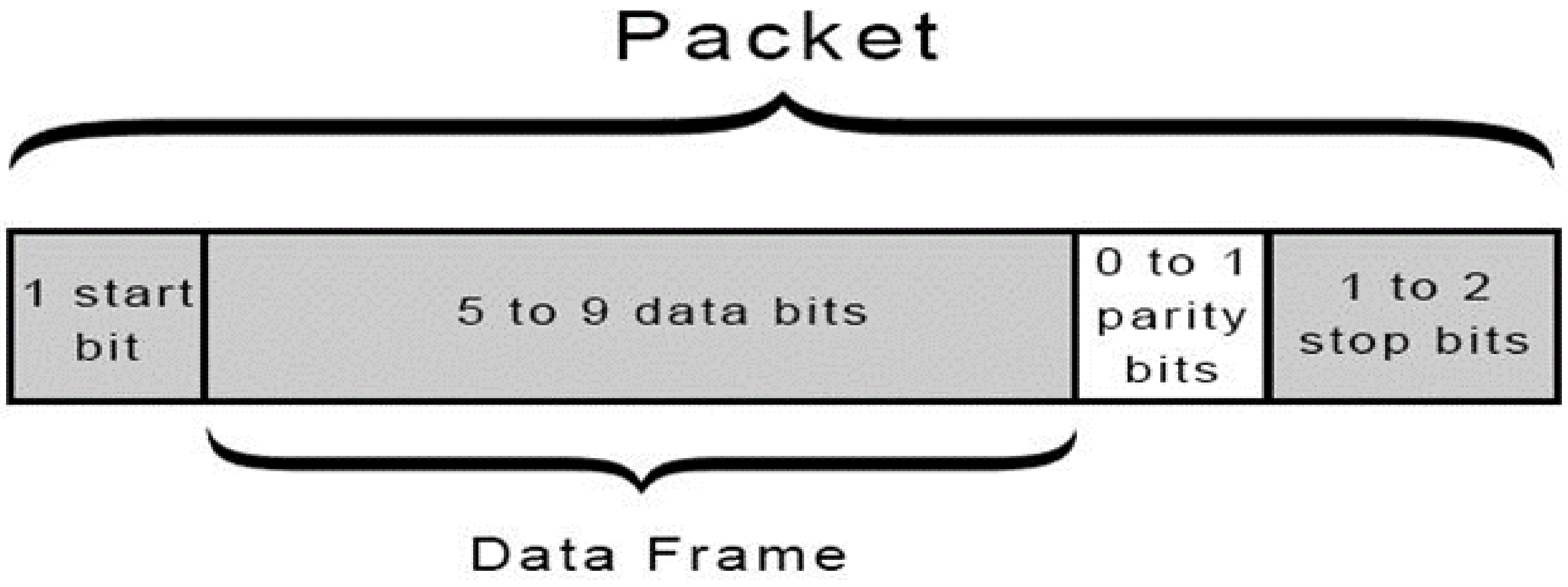


HOW UART WORKS

- Next, data packet is output serially, bit by bit at T_x pin.
- Receiving UART reads data packet bit by bit at its R_x pin.
- Receiving UART then converts data back into parallel form and removes start bit, parity bit, and stop bits.
- Finally, receiving UART transfers data packet in parallel to data bus on receiving end.



HOW UART WORKS



- In UART transmitted data is organized into *packets*.
- Each packet contains 1 start bit, 5 to 9 data, an optional *parity* bit, and 1 or 2 stop bits:

HOW UART WORKS

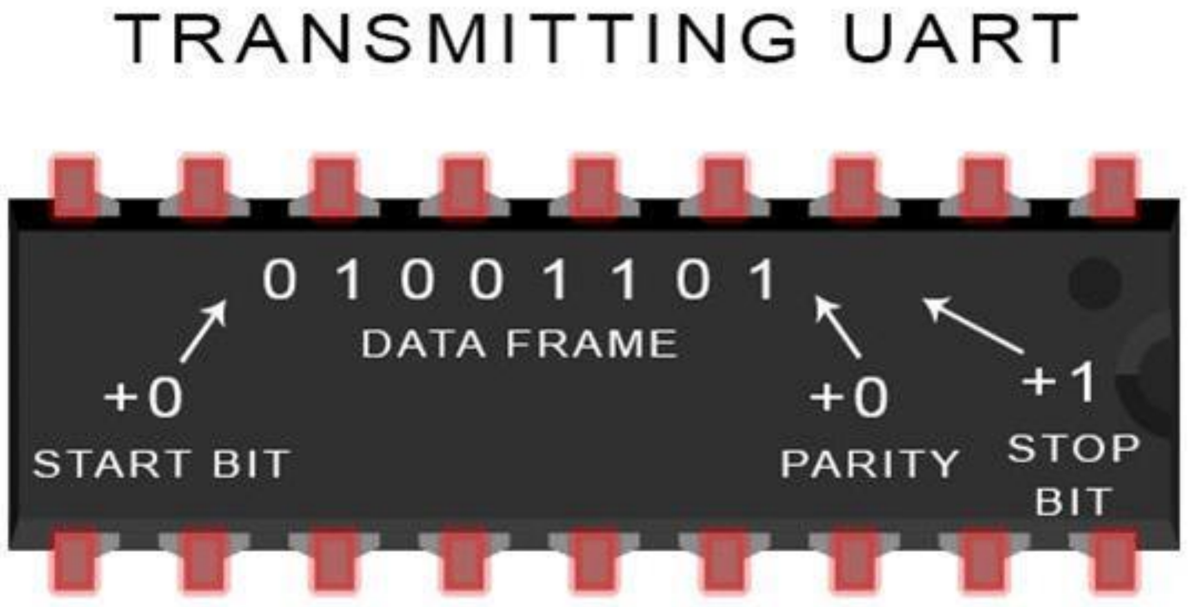
1. **START BIT**
   - UART data transmission line is normally set high meaning not transmitting data.
   - To start transfer of data, transmitting UART pulls transmission line from high to low for one clock cycle.
   - When receiving UART detects high to low voltage transition, it begins reading bits in data frame at frequency of baud rate.
2. **DATA FRAME:**
   - Contains actual data being transferred.
   - Can be 5 bits to 9 bits long if a parity bit is used.
   - If no parity bit is used, data frame can be 8 bits long.
   - In most cases, data is sent with LSB first.

HOW UART WORKS

3. **PARITY**
   - Parity describes evenness or oddness of a number.
   - Parity bit is a way for receiving UART to tell if any data has changed during transmission.
   - Bits gets changed due to EM radiation, mismatched baud rates, or long distance data transfers.
   - After receiving UART reads data frame and counts number of bits with a value of 1 and checks if total is an even or odd number.
   - If parity bit is 0 (even parity), 1 bits in data frame should total to an even number.
   - If parity bit is 1 (odd parity), 1 bits in data frame should total to an odd number.
   - When parity bit matches data, UART knows that transmission was free of errors.
   - But if parity bit is 0, and total is odd; or parity bit is 1, and total is even, UART knows that bits in data frame have changed.
4. **STOP BITS**
   - Marks end of data packet.
   - Usually is two bits long but often only one bit is used.
   - In order to end transmission, UART maintains data line at high voltage (1).

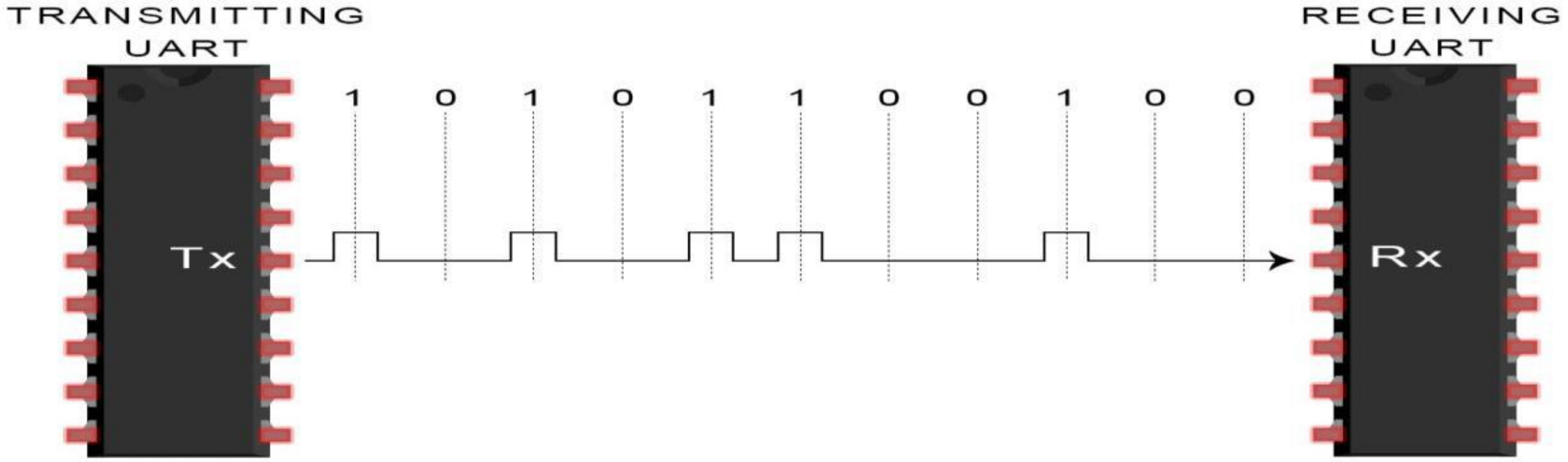
STEPS OF UART TRANSMISSION

1. Transmitting UART receives data in parallel from data bus
2. Transmitting UART adds start bit, parity bit, and stop bit(s) to data frame



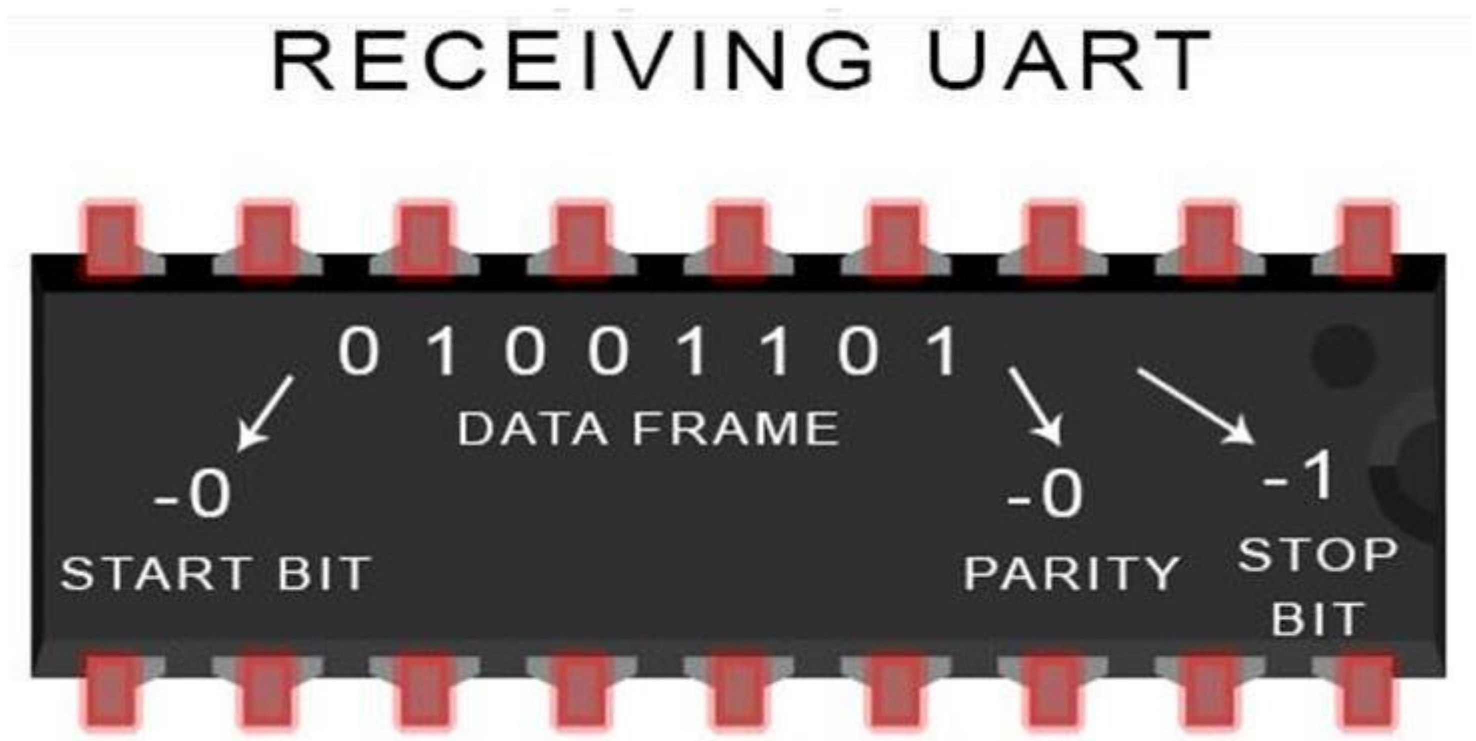
STEPS OF UART TRANSMISSION

3. Entire packet is sent serially from transmitting UART to receiving UART. Receiving UART samples data line at pre-configured baud rate



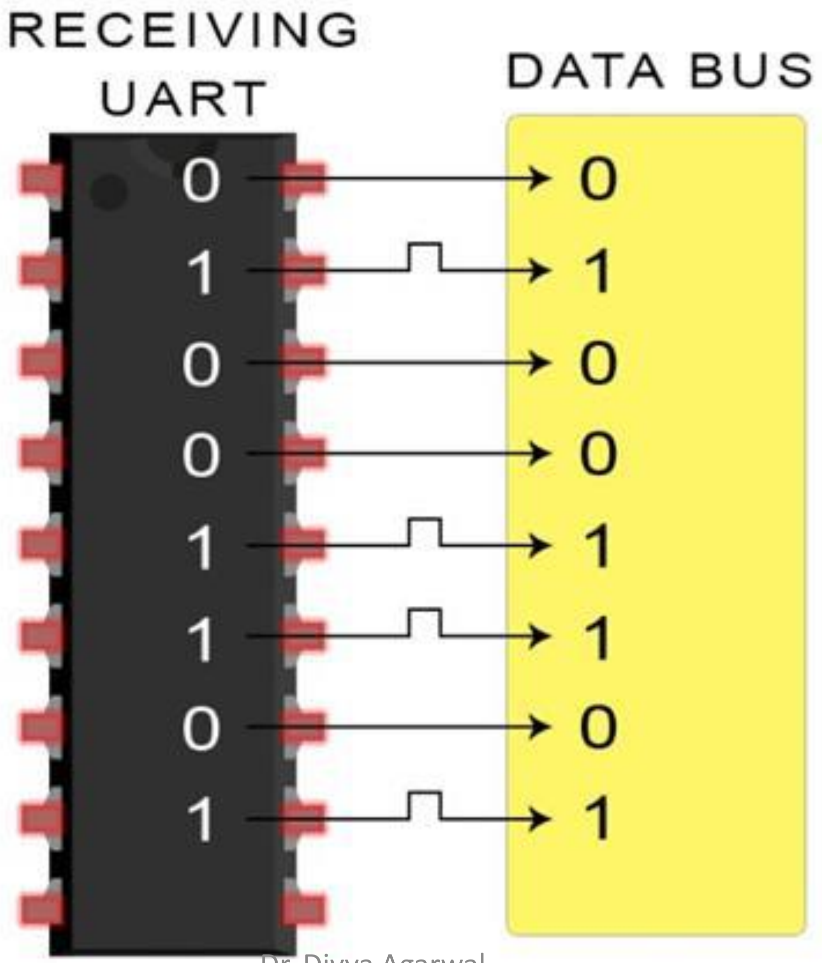
STEPS OF UART TRANSMISSION

4. Receiving UART discards start bit, parity bit, and stop bit from data frame



STEPS OF UART TRANSMISSION

5. Receiving UART converts serial data back into parallel and transfers it to data bus on receiving end

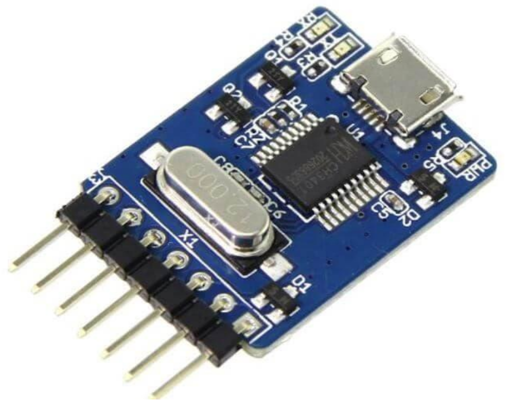


Advantages and Disadvantages of UARTs

- No communication protocol is perfect, but UARTs are pretty good at what they do.
- **ADVANTAGES**
  - Only uses two wires
  - No clock signal is necessary
  - Has a parity bit to allow for error checking
  - Structure of data packet can be changed as long as both sides are set up for it
  - Well documented and widely used method
- **DISADVANTAGES**
  - Size of data frame is limited to a maximum of 9 bits
  - Doesn’t support multiple slave or multiple master systems
  - Baud rates of each UART must be within 10% of each other

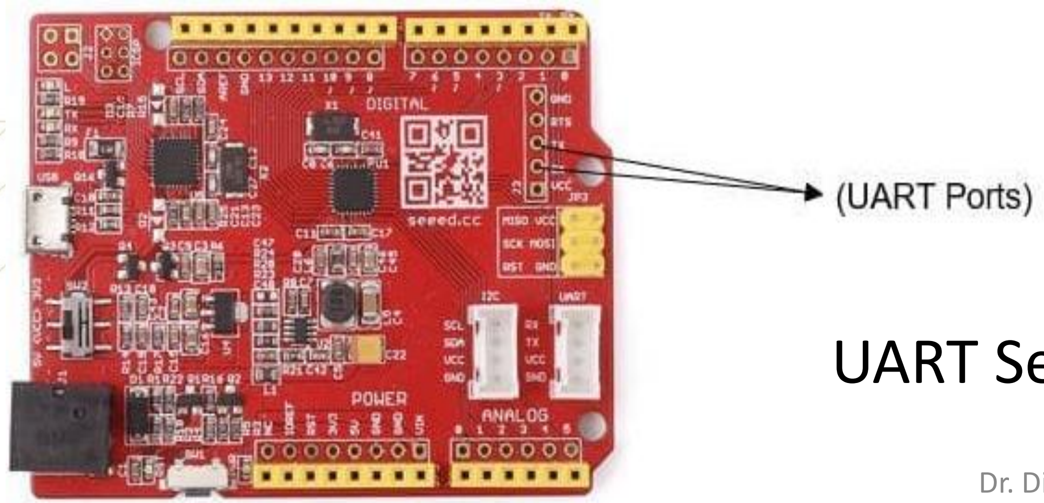
Examples of UARTs

USB CP2102
Serial Converter



FT232r USB UART / USB to UART 5V

All Arduino boards have at least one serial port (UART) which communicates on digital pins 0 (R_X) and 1 (T_X) as well with computer via USB.



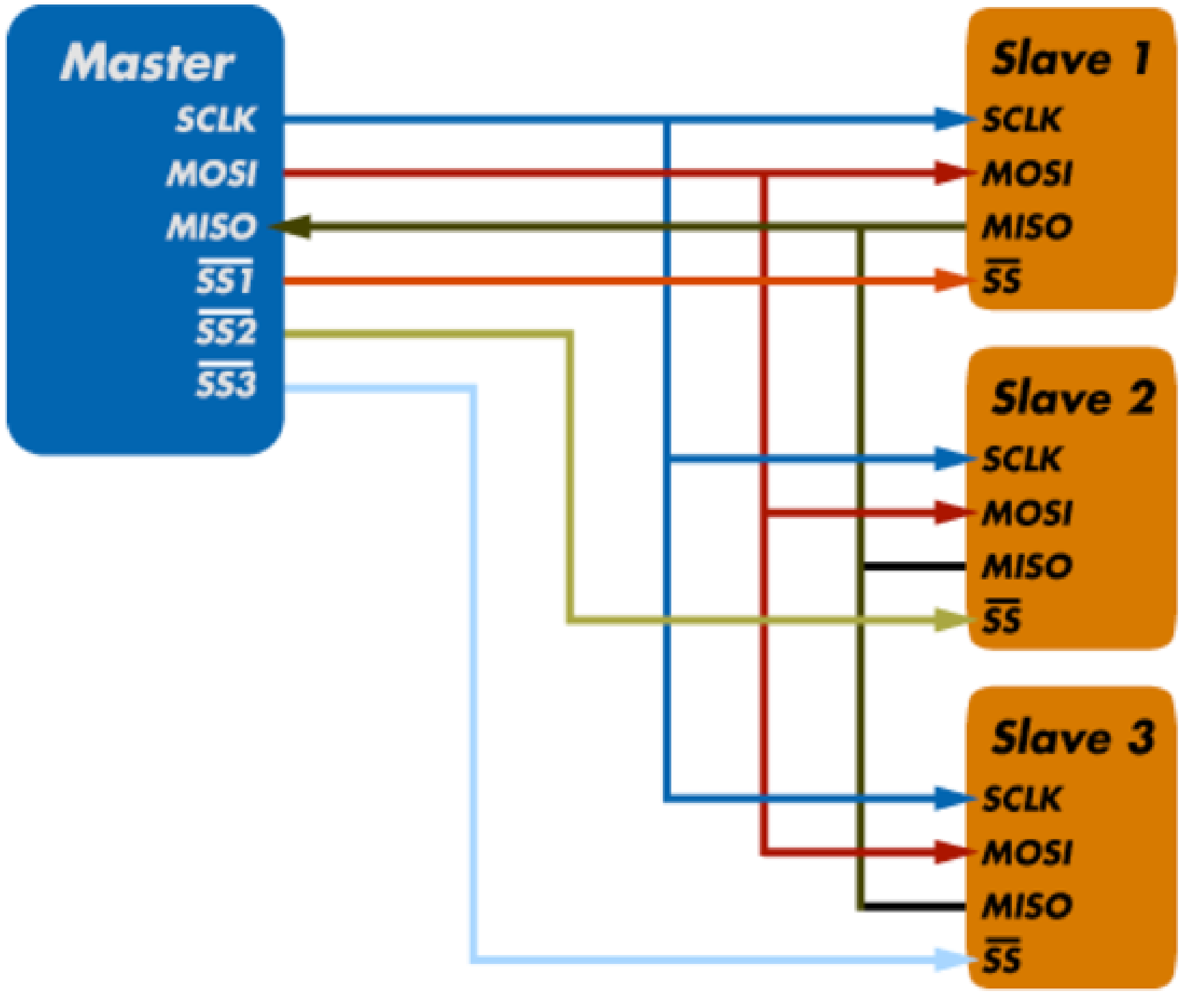
UART Seeeduino V4.2

Serial Peripheral Interface - SPI

- SPI is a common communication protocol used by many different devices.
- Data is transferred without interruption i.e., any number of bits can be sent or received in a continuous stream.
- With I^2C and UART, data is sent in packets, limited to a specific number of bits. Start and stop conditions define beginning and end of each packet, so data is interrupted during transmission.
- Devices communicating via SPI are in a master-slave relationship.
- Master is controlling device (usually a microcontroller), while slave (usually a sensor, display, or memory chip) takes instruction from master.
- Operates at full-duplex where data can be sent and received simultaneously.
- Used in places where speed is important. (eg. SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all use SPI to communicate with microcontrollers)

SPI Communication Protocol

- Operates using a master-slave paradigm that includes at least four signals:
- **MOSI (Master Output/Slave Input) –** Line for master to send data to slave.
- **MISO (Master Input/Slave Output) –** Line for slave to send data to master
- **SCLK (Clock) –** Line for clock signal.
- **SS/CS (Slave Select/Chip Select) –** Line for master to select which slave to send data to.
- SCLK, MOSI, and MISO signals are shared by all devices on the bus.
- SCLK signal is generated by master device for synchronization, while MOSI and MISO lines used for data exchange.

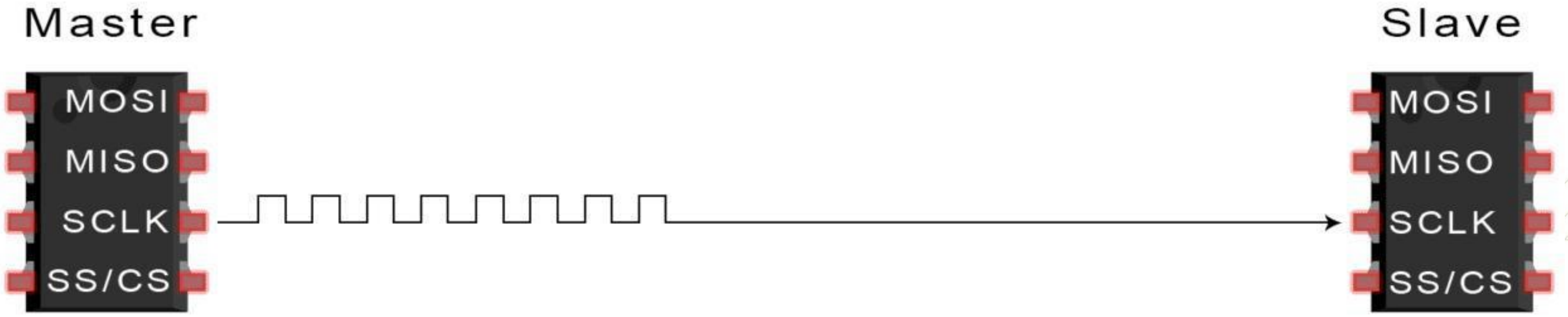


SPI Working

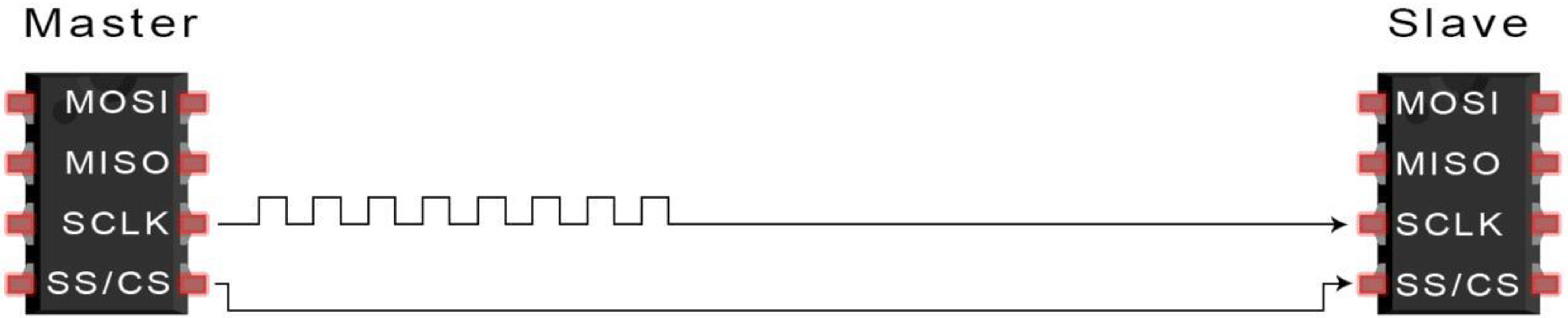
- SPI communicate with 2 ways:
  1. Selecting each device with a separate **Chip Select line.**
  2. **Daisy chaining** where each device is connected to or through its data out to data in line of next.
- There is no limit to number of SPI device that can be connected.
- In point-to-point communication, SPI interface does not require addressing operations and is full-duplex communication, which is simple and efficient.
- **SPI Working Protocol:** SPI communicates via 4 ports which are:
  1. MOSI – Master Data Output, Slave Data Input
  2. MISO – master data input, slave data output
  3. SCLK – clock signal, generated by master device
  4. NSS – Slave enabled signal, controlled by master device, some ICs will be labelled as CS (Chip select)

Steps of SPI Data Transmission

1. Master outputs clock signal

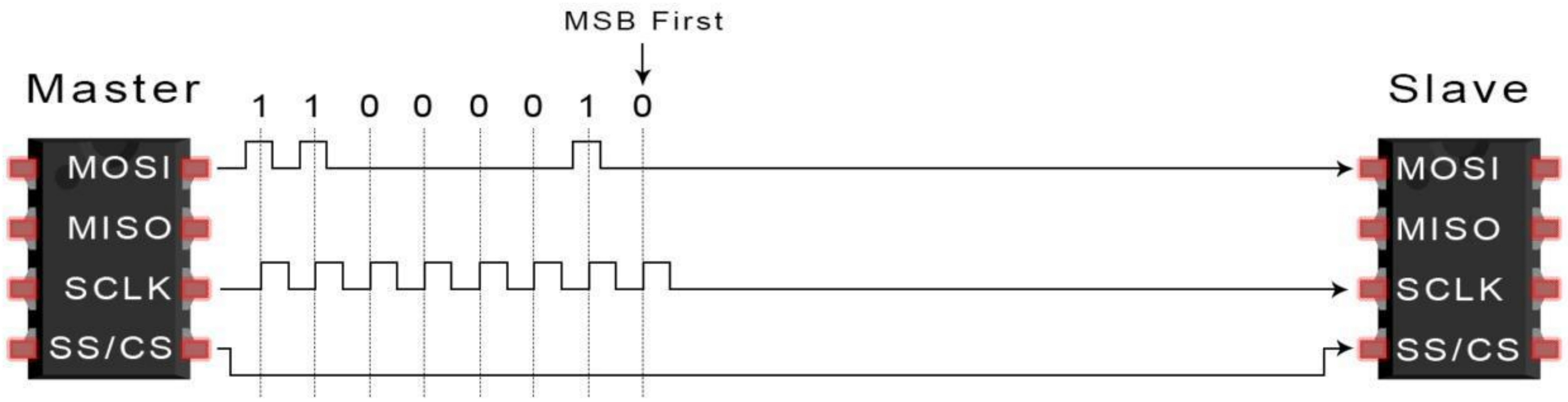


2. Master switches SS/CS pin to a low voltage state, which activates slave

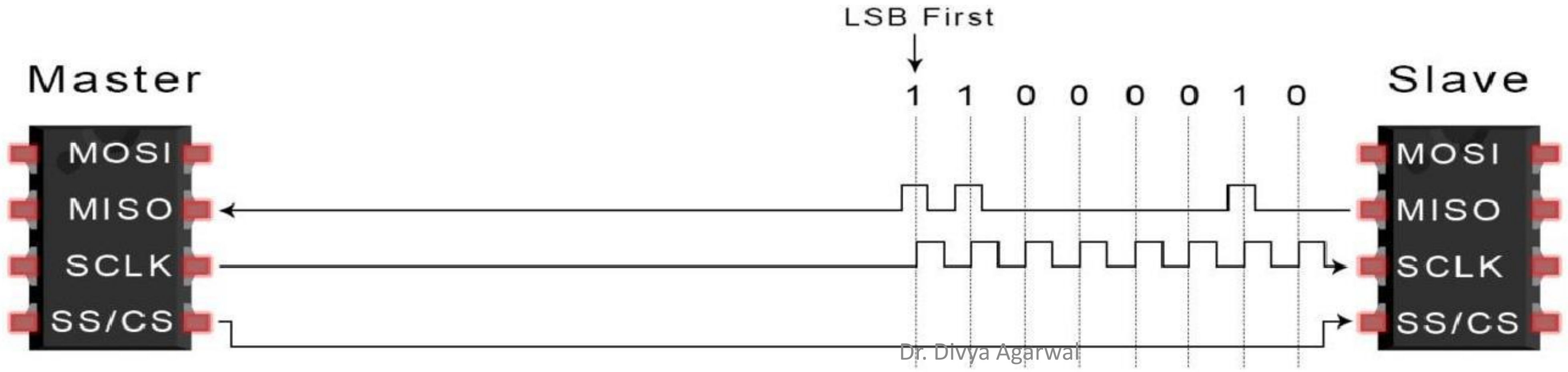


Steps of SPI Data Transmission

3. Master sends data one bit at a time to slave along MOSI line. Slave reads bits as they are received



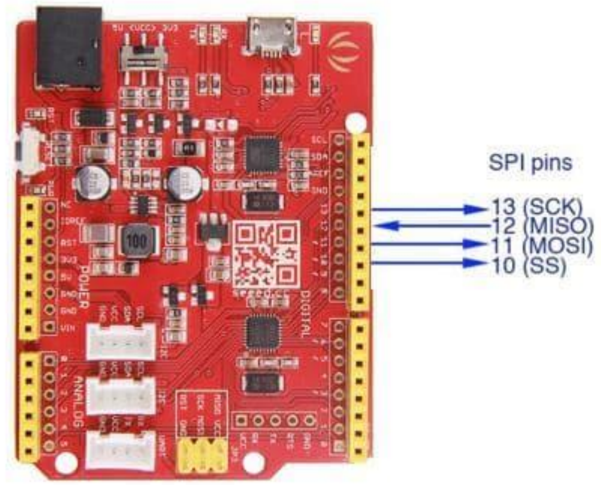
4. If response is needed, slave returns data one bit at a time to master along MISO line. Master reads bits as they are received



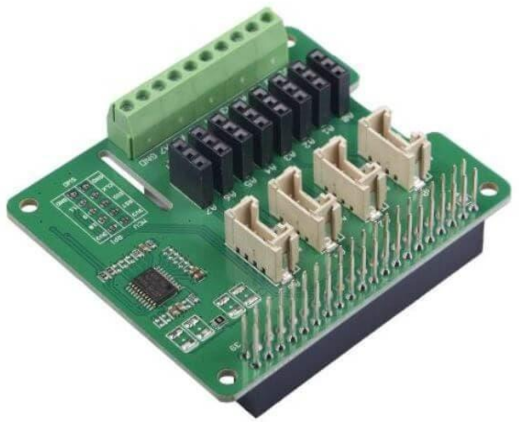
Advantages and Disadvantages of SPI

- **ADVANTAGES**
  - ❑ No start and stop bits, so data can be streamed continuously without interruption
  - ❑ No complicated slave addressing system like I^{2}C
  - ❑ Higher data transfer rate than I^{2}C (almost twice as fast)
  - ❑ Separate MISO and MOSI lines, so data can be sent and received at same time
- **DISADVANTAGES**
  - ❑ Uses four wires (I^{2}C and UARTs use two)
  - ❑ No acknowledgement that data has been successfully received (I^{2}C has this)
  - ❑ No form of error checking like parity bit in UART
  - ❑ Only allows for a single master.

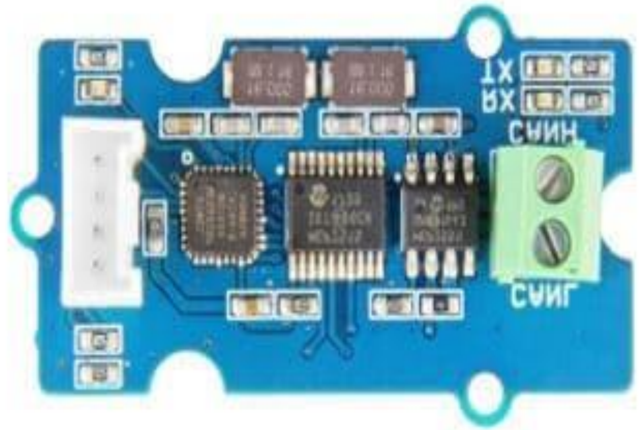
Examples of SPIs in Microcontrollers



SPI Seeeduino V4.2

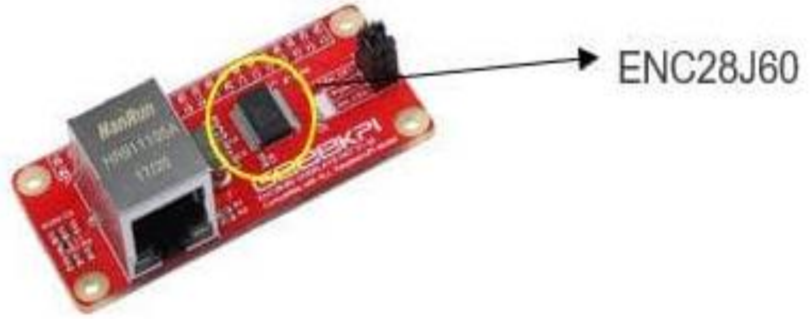


MCP 3008 / Grove I2C ADC



Serial CAN-BUS Module based on MCP2551 and MCP2515

ENC28J60 OVERLAYS HAT for Raspberry pi

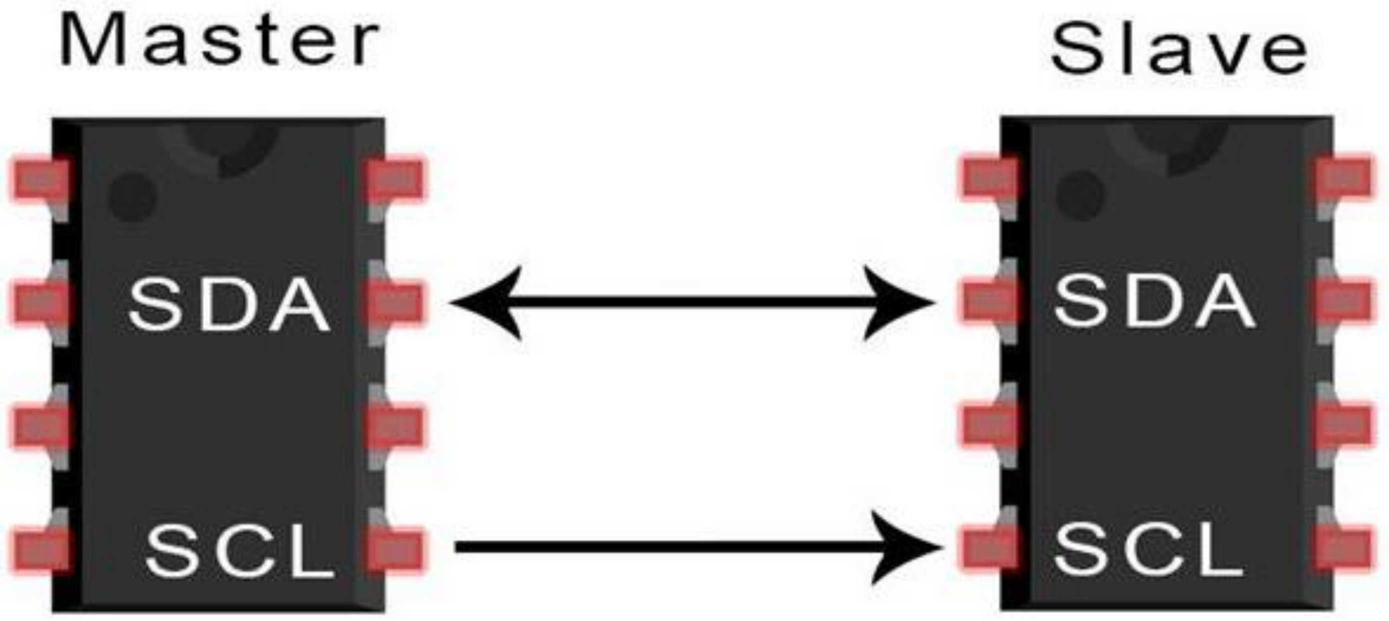


I2C Communication Protocol

- I2C stands for **Inter-Integrated Circuit** also known as **Two Wired Interface(TWI).**
- Bus interface connection protocol incorporated into devices for serial communication similar to UART.
- Originally designed by Philips Semiconductor in 1982.
- Widely used protocol for short-distance communication.
- Not used for PC-device communication but instead with modules and sensors.
- Simple, bidirectional two-wire synchronous serial bus and requires only two wires to transmit information between devices connected to bus.
- I2C uses an address system and a shared bus
- Many different devices can be connected using same wires and all data are transmitted on a single wire
- Speed of I2C is affected by data speed, wire quality and external noise

Working of I2C Communication Protocol

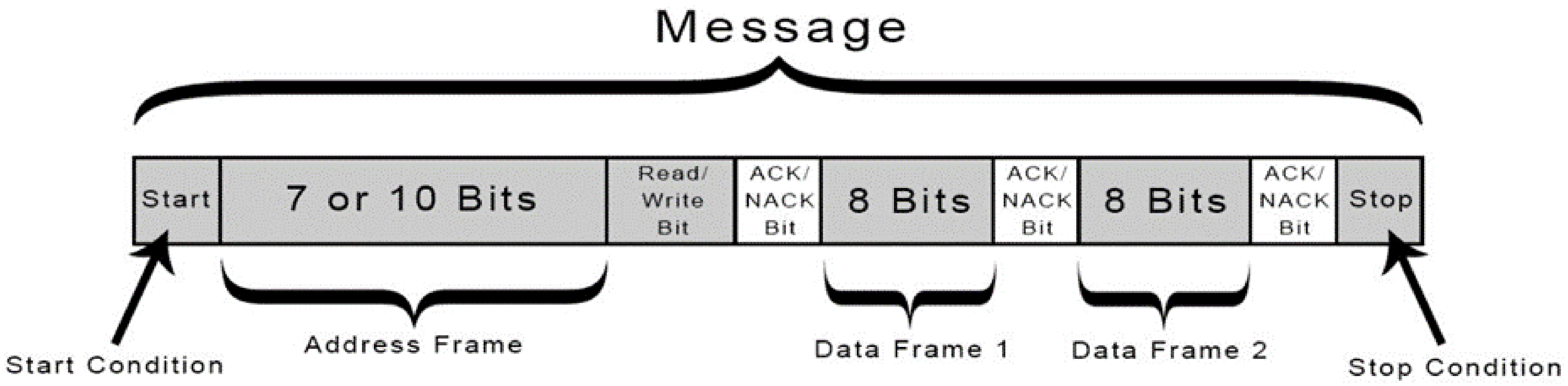
- Uses only 2 bi-directional open-drain lines for data communication called SDA and SCL.
- Serial Data (SDA) – Transfer of data takes place through this pin (data line).
- Serial Clock (SCL) – To carry clock signal for synchronizing transmission.
- Both these lines are pulled high.
- I2C operates in 2 modes –
  1. Master mode
  2. Slave mode



Working of I2C Communication Protocol

- I2C operates in 2 modes –
  1. Master mode
  2. Slave mode
- Each data bit transferred on SDA line is synchronized by a high to low pulse of each clock on SCL line.
- Master device initiates bus transfer of data and generates a clock to open transferred device and any addressed device is considered a slave device.
- Relationship between master and slave devices, transmitting and receiving on bus is not constant and depends on direction of data transfer at time.
- If master wants to send/receive data to/from slave, it must first address slave.

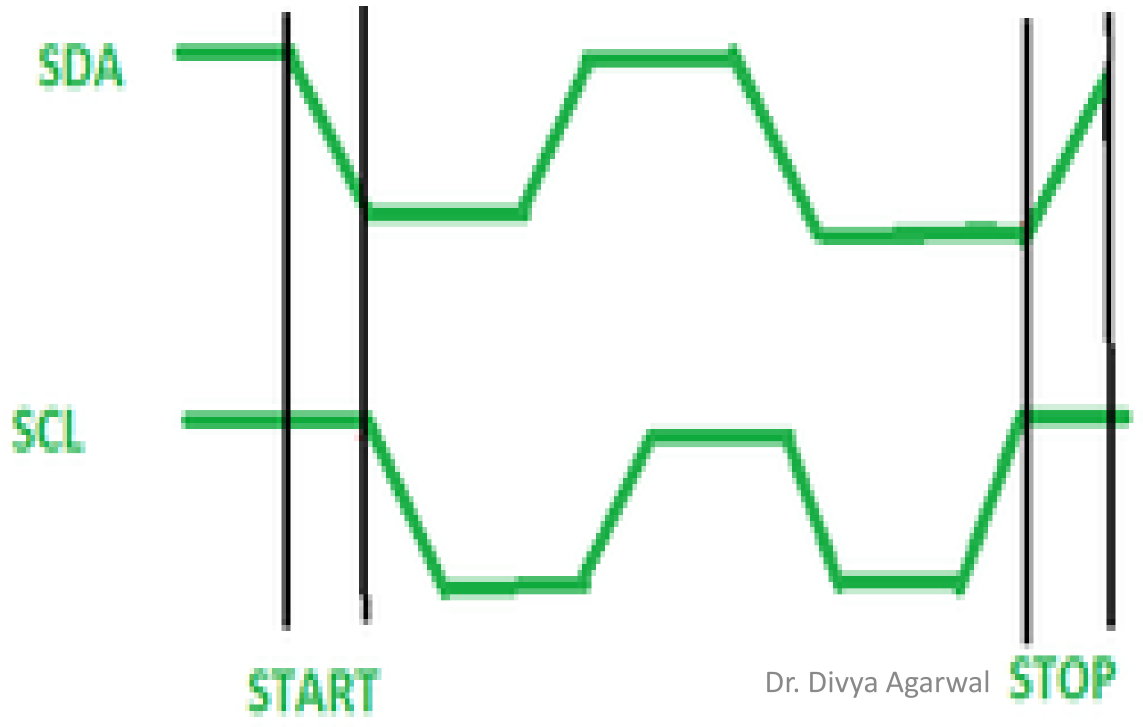
I2C Packet Format



- Here, data is transmitted in form of packets.
- Packets are 9 bits long = first 8 bits are put in SDA line + 1 bit for ACK/NACK
- START condition plus address packet plus one more data packet plus STOP condition collectively form a complete Data transfer.

I2C Packet Format

- Data line can not change when clock line is high
- **Start and Stop Conditions :** START and STOP can be generated by keeping SCL line high and changing level of SDA. For **START** condition SDA is changed from high to low while for **STOP** condition SDA goes from low to high while keeping SCL high.

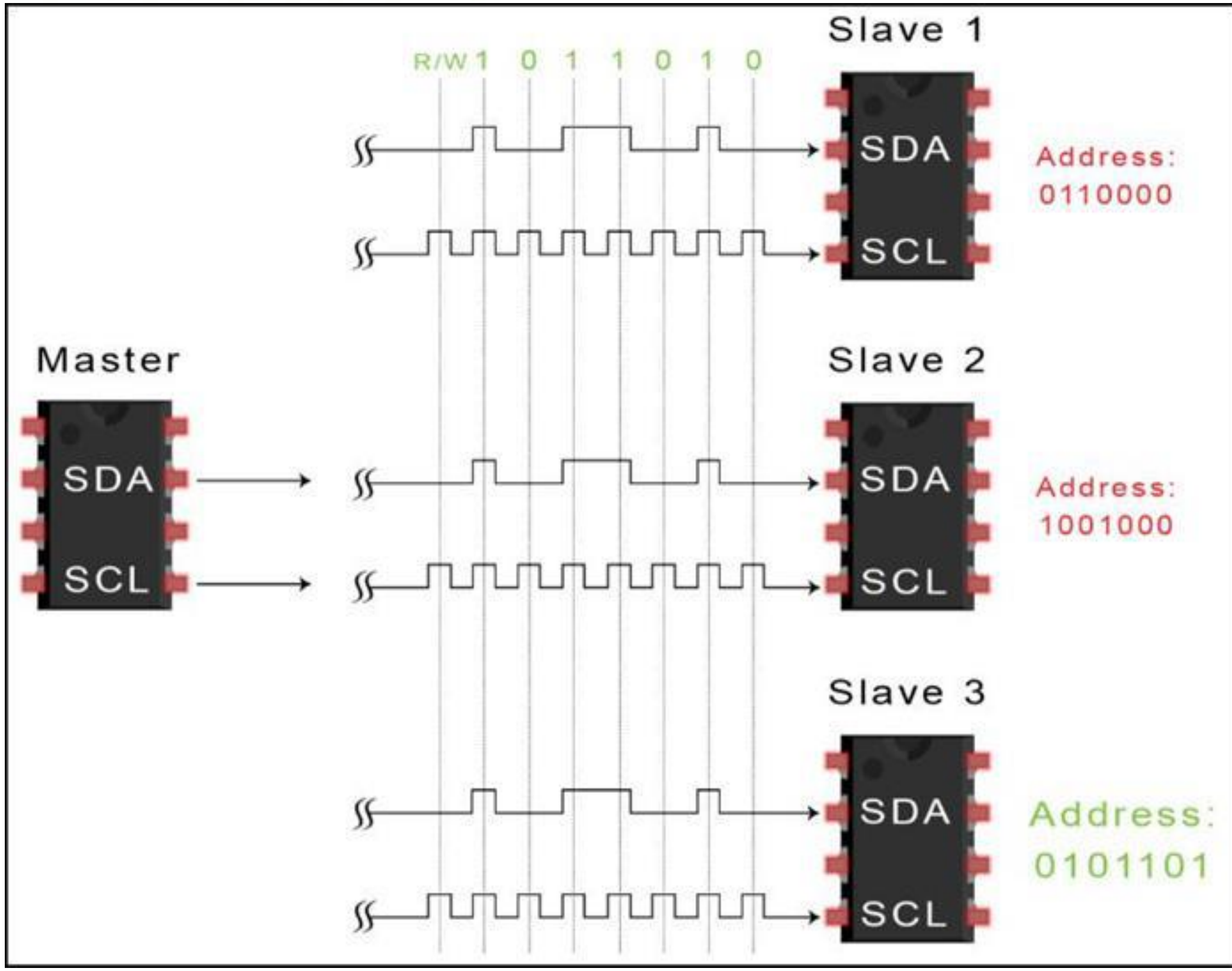


I2C Packet Format

- **Repeated Start Condition :** Between each start and stop condition pair, bus is considered as busy and no master can take control of bus. If master tries to initiate a new transfer and does not want to release bus before starting new transfer, it issues a new START condition called a REPEATED START condition.
- **Read/Write Bit :** High Read/Write bit indicates that master is sending data to slave, whereas low Read/Write bit indicates that master is receiving data from slave.
- **ACK/NACK Bit :** After every data frame, follows an ACK/NACK bit. If data frame is received successfully then ACK bit is sent to sender by receiver.
- **Addressing :** Address frame is first frame after start bit. Slave address with which master wants to communicate is sent by master to every slave connected with it. Slave then compares its own address with this address and sends ACK.

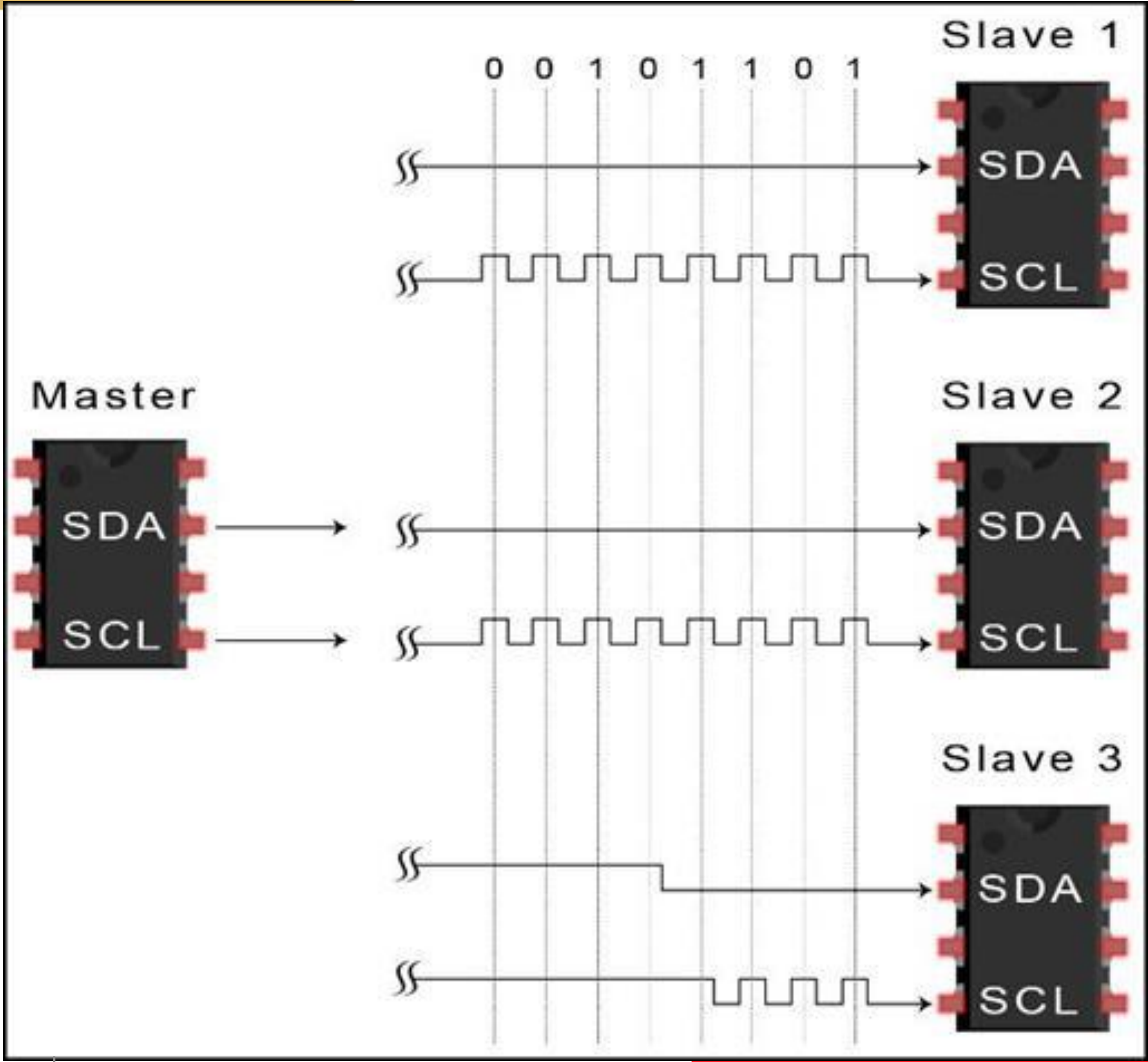
STEPS OF I2C DATA TRANSMISSION

1. Master sends start condition to every connected slave by switching SDA line from high voltage to low voltage level before switching SCL line from high to low:
2. Master sends each slave 7 or 10 bit address of slave it wants to communicate with, along with read/write bit
3. Each slave compares address sent from master to its own address. If address matches, slave returns an ACK bit by pulling SDA line low for one bit else sets SDA line high.



STEPS OF I2C DATA TRANSMISSION

4. Master sends or receives data frame
5. After each data frame has been transferred, receiving device returns another ACK bit to sender to acknowledge successful receipt of frame
6. To stop data transmission, master sends a stop condition to slave by switching SCL high before switching SDA high



Advantages and Disadvantages of I2C

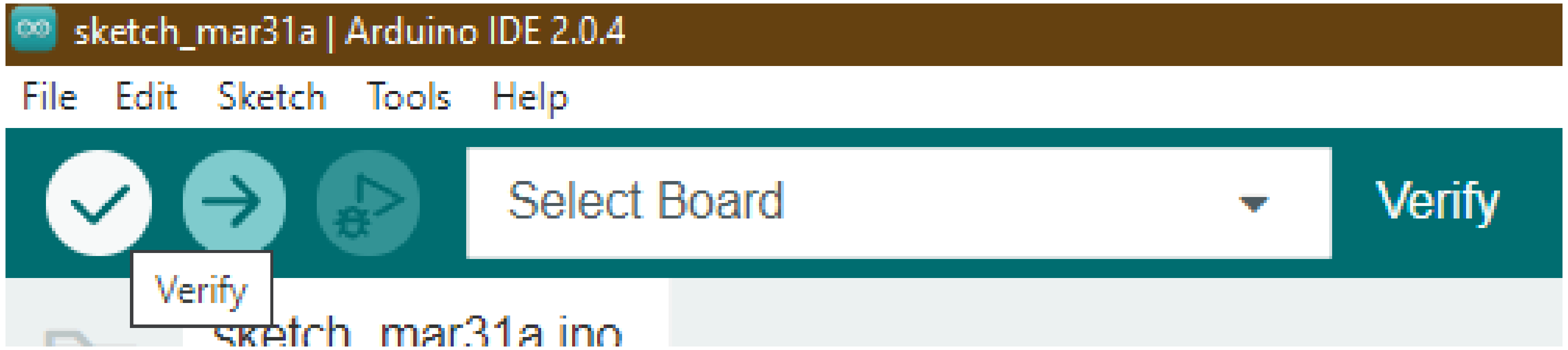
- **ADVANTAGES**
  - ❑ Only uses two wires
  - ❑ Supports multiple masters and multiple slaves
  - ❑ ACK/NACK bit gives confirmation that each frame is transferred successfully
  - ❑ Hardware is less complicated than with UARTs
  - ❑ Well known and widely used protocol
- **DISADVANTAGES**
  - ❑ Slower data transfer rate than SPI
  - ❑ The size of the data frame is limited to 8 bits
  - ❑ More complicated hardware needed to implement than SPI

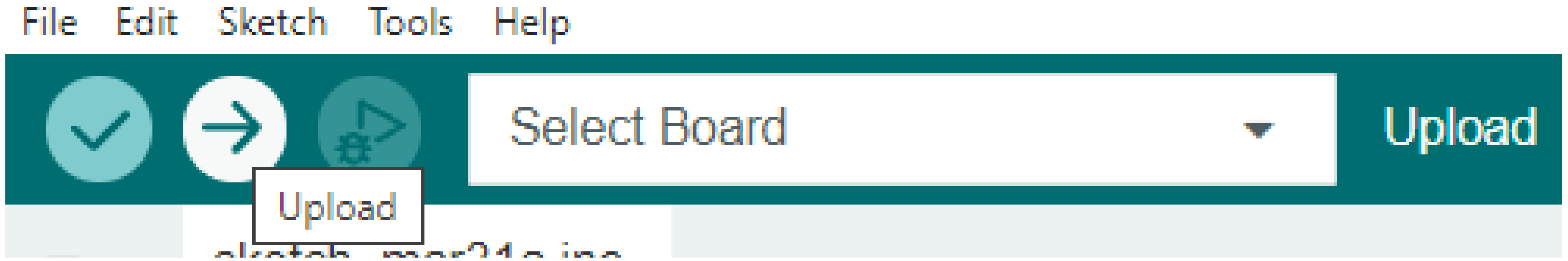
ARDUINO IDE OVERVIEW:

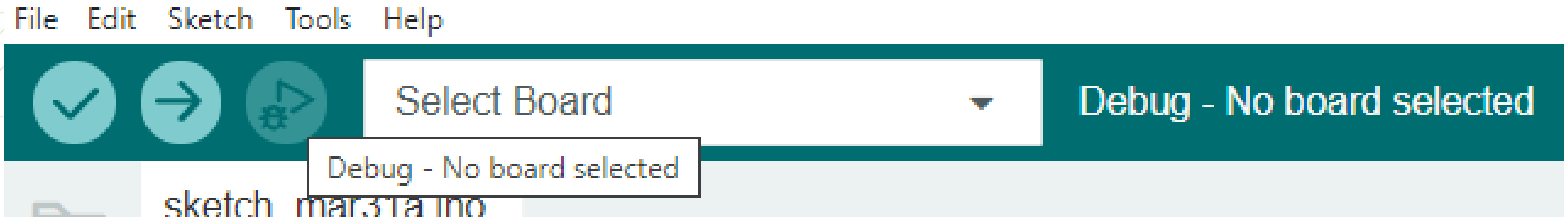
- **Program coded in Arduino IDE is called a SKETCH**

1. To create a new sketch **File -> New**
2. To open an existing sketch **File -> open ->**
3. To open ready-to-use sketches **File -> Examples -> select any program**
4. **Verify:** Checks code for compilation errors
5. **Upload:** Uploads final code to controller board
6. **New:** Creates a new blank sketch with basic structure
7. **Open:** Opens an existing sketch
8. **Save:** Saves current sketch
9. **Serial Monitor:** Opens serial console

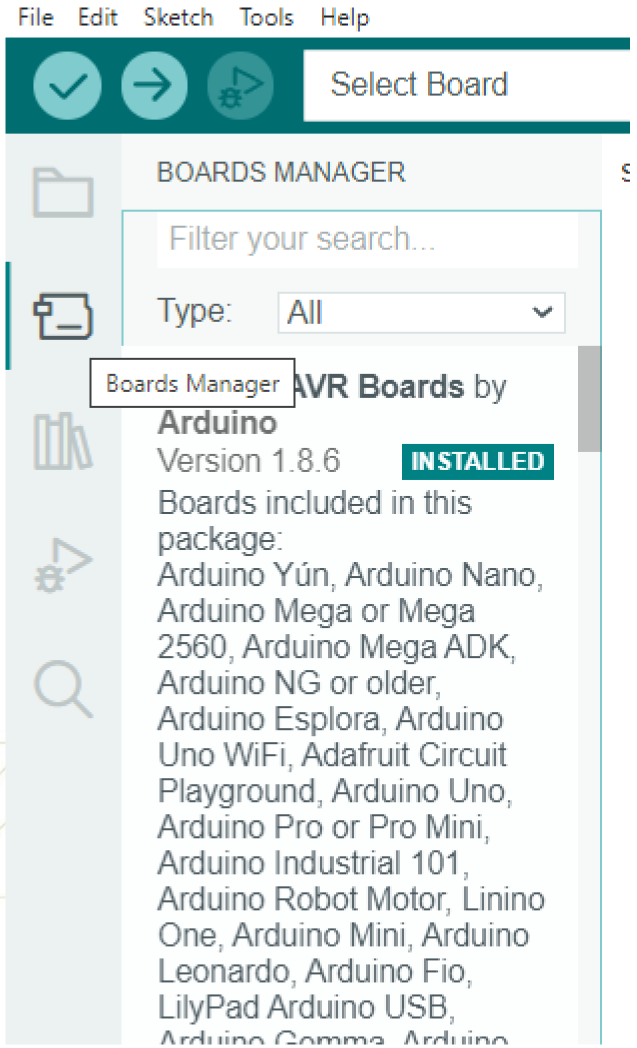
ARDUINO IDE OVERVIEW:


sketch_mar31a | Arduino IDE 2.0.4
File Edit Sketch Tools Help
Select Board
Verify
Verify
sketch_mar31a.ino

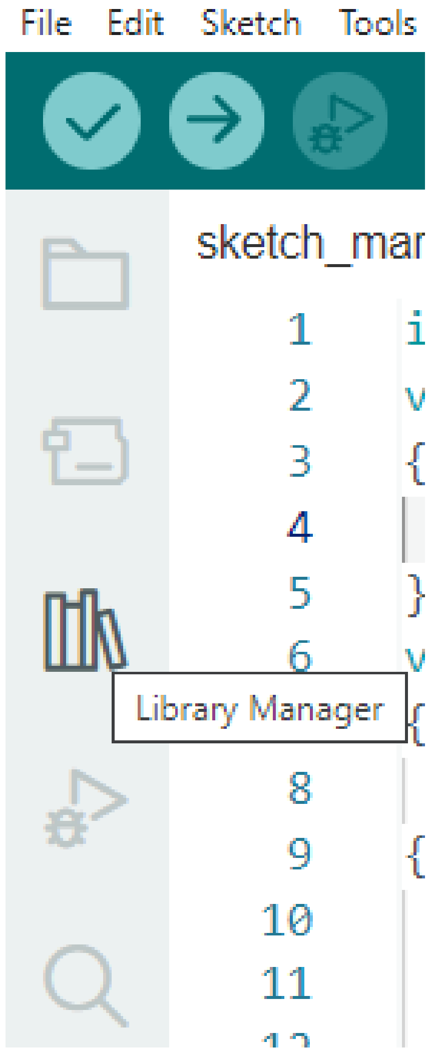


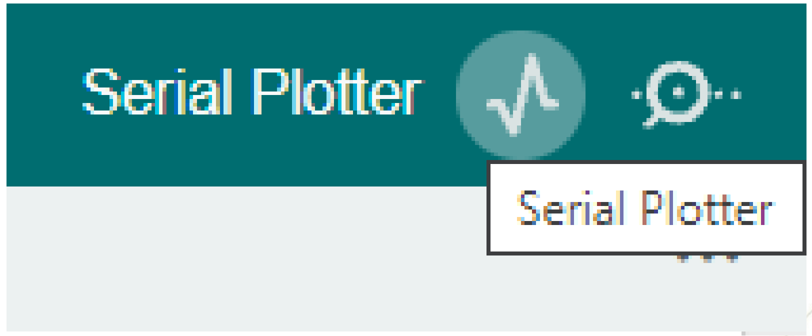

File Edit Sketch Tools Help
Select Board
Debug - No board selected
Debug - No board selected
sketch_mar31a.ino

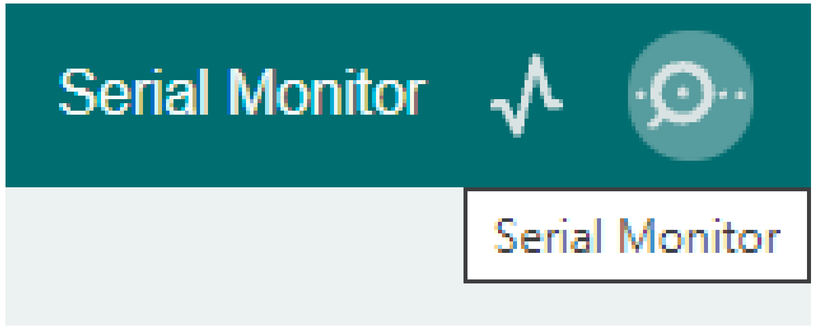
ARDUINO IDE OVERVIEW:



AVR Boards by Arduino
Version 1.8.6 INSTALLED
Boards included in this package:
Arduino Yún, Arduino Nano, Arduino Mega or Mega 2560, Arduino Mega ADK, Arduino NG or older, Arduino Esplora, Arduino Uno WiFi, Adafruit Circuit Playground, Arduino Uno, Arduino Pro or Pro Mini, Arduino Industrial 101, Arduino Robot Motor, Linino One, Arduino Mini, Arduino Leonardo, Arduino Fio, LilyPad Arduino USB,





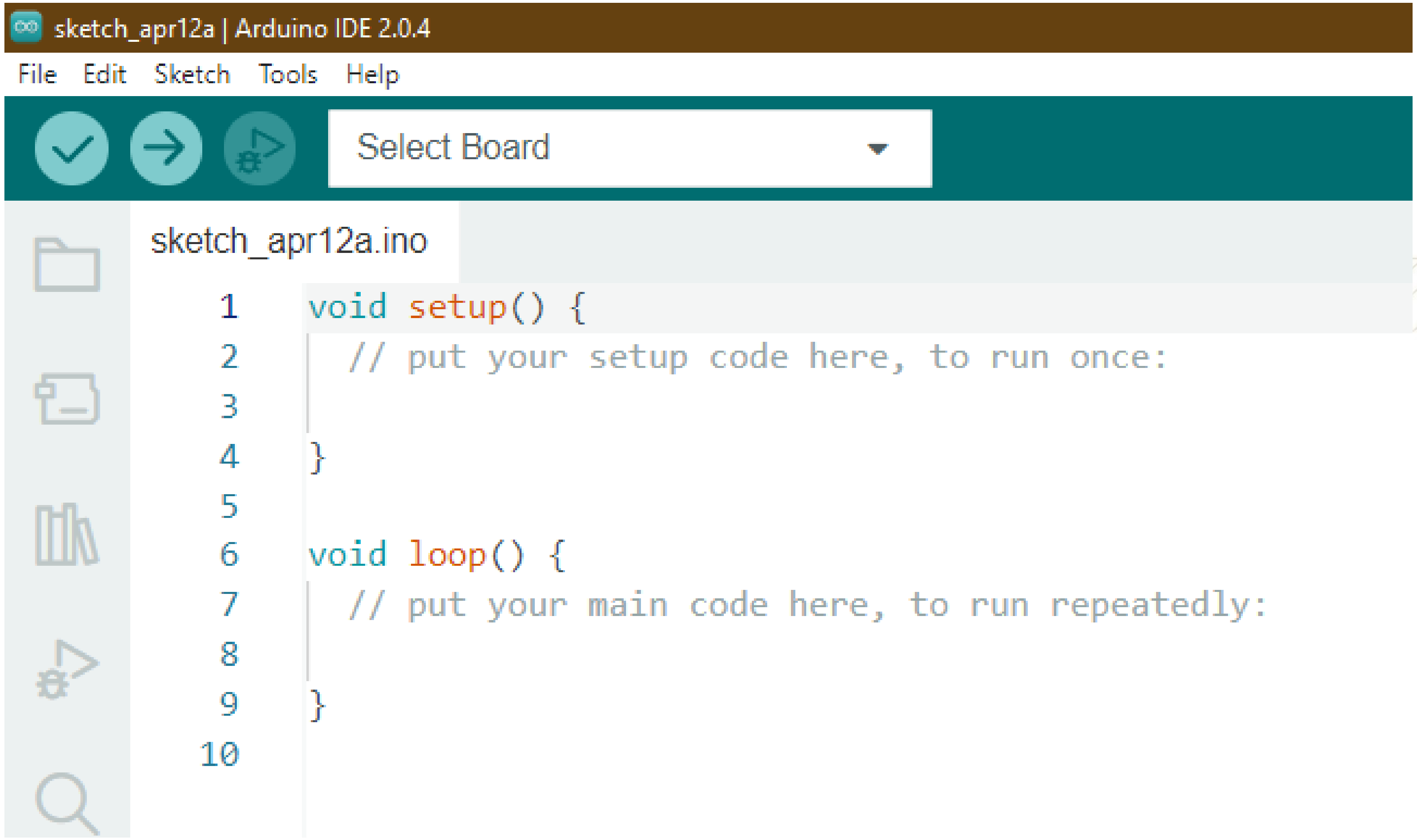


SCHOOL OF ENGINEERING AND TECHNOLOGY

Structure of SKETCH

Sketch can be divided into two parts:

1. Setup ()
2. Loop()


sketch_apr12a | Arduino IDE 2.0.4
File Edit Sketch Tools Help
Select Board
sketch_apr12a.ino

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Structure of SKETCH

A sketch can be divided into two parts:

1. **Setup ()**

- Here, code starts, like main() function in C and C++
- I/O Variables, pin modes are initialized in Setup() function

2. **Loop()**

- Loop() function, iterates specified task in program

DATA TYPES

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

Arduino Function Libraries

1. **Input/Output Functions:**
   - Arduino pins can be configured to act as input or output pins using pinMode() function
2. **Example**

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

   - **Pin-** pin number on Arduino board
   - **Mode-** INPUT/OUTPUT
3. **digitalWrite():** Writes a HIGH or LOW value to a digital pin
4. **analogRead():** Reads from analog input pin i.e., voltage applied across pin
5. **Character functions:** isdigit(), isalpha(), isalnum(), isxdigit(), islower(), isupper(), isspace() return 1(true) or 0(false)
6. **Delay() function:** Most common time manipulation function used to provide a delay of specified time. It accepts integer value (time in miliseconds)

EXAMPLE BLINKING LED

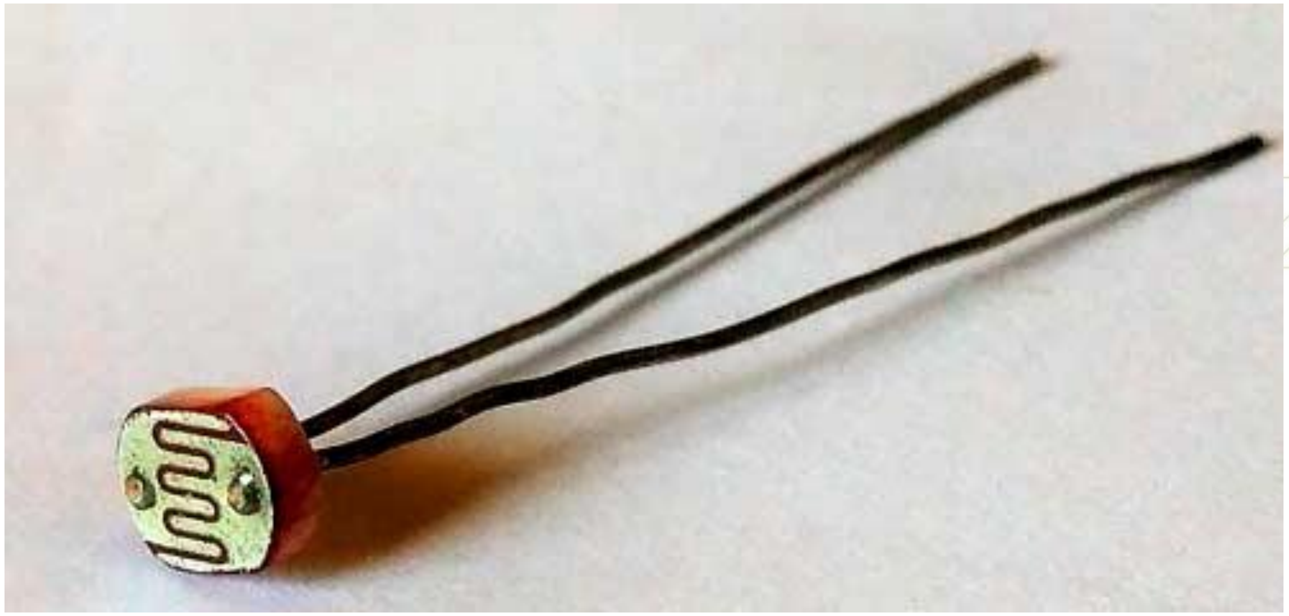
1. **Requirement:**
   - Arduino controller board,
   - USB connector,
   - Bread board,
   - LED,
   - 1.4Kohm resistor,
   - connecting wires,
2. **Arduino IDE**
   - Connect LED to Arduino using Bread board and connecting wires
   - Connect Arduino board to PC using USB connector
   - Select board type and port
   - Write sketch in editor, verify and upload
   - Connect positive terminal of LED to digital pin …… and negative terminal to ground pin (GND) of Arduino Board

EXAMPLE BLINKING LED

```
void setup()
{
pinMode(3, OUTPUT); // set the pin mode
}
void loop()
{
digitalWrite(3, HIGH); // Turn on the LED
delay(1000);
digitalWrite(3, LOW); //Turn of the LED
delay(1000);
}
```

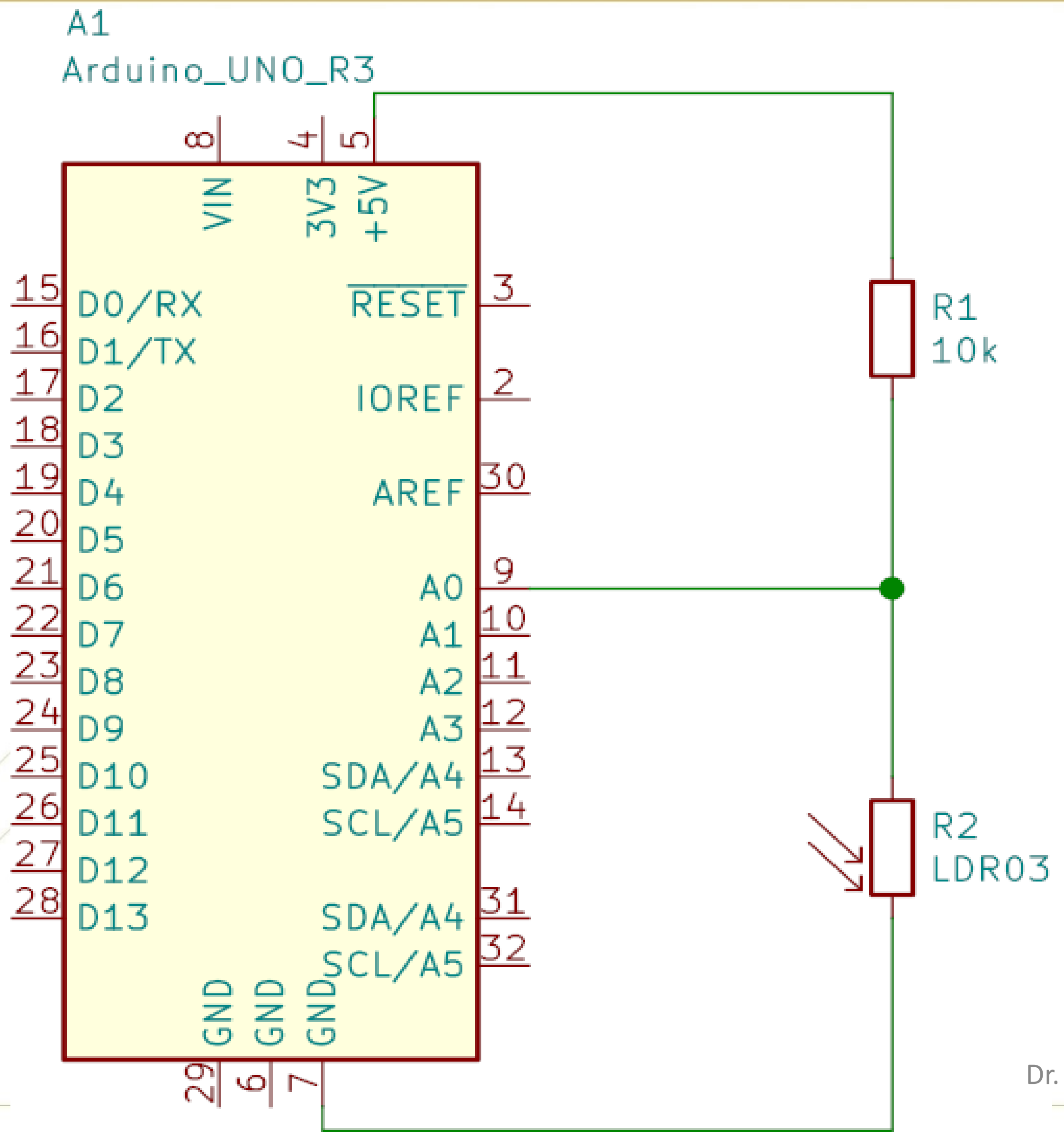
Light Dependent Resistor - LDR

- LDR is a type of resistor that changes resistance as light on its surface changes.
- i.e., less light or more darkness on LDR surface causes its resistance to increase.
- LDR connected in series with a 10k resistor.
- To detect light levels and switch an LED on or off.



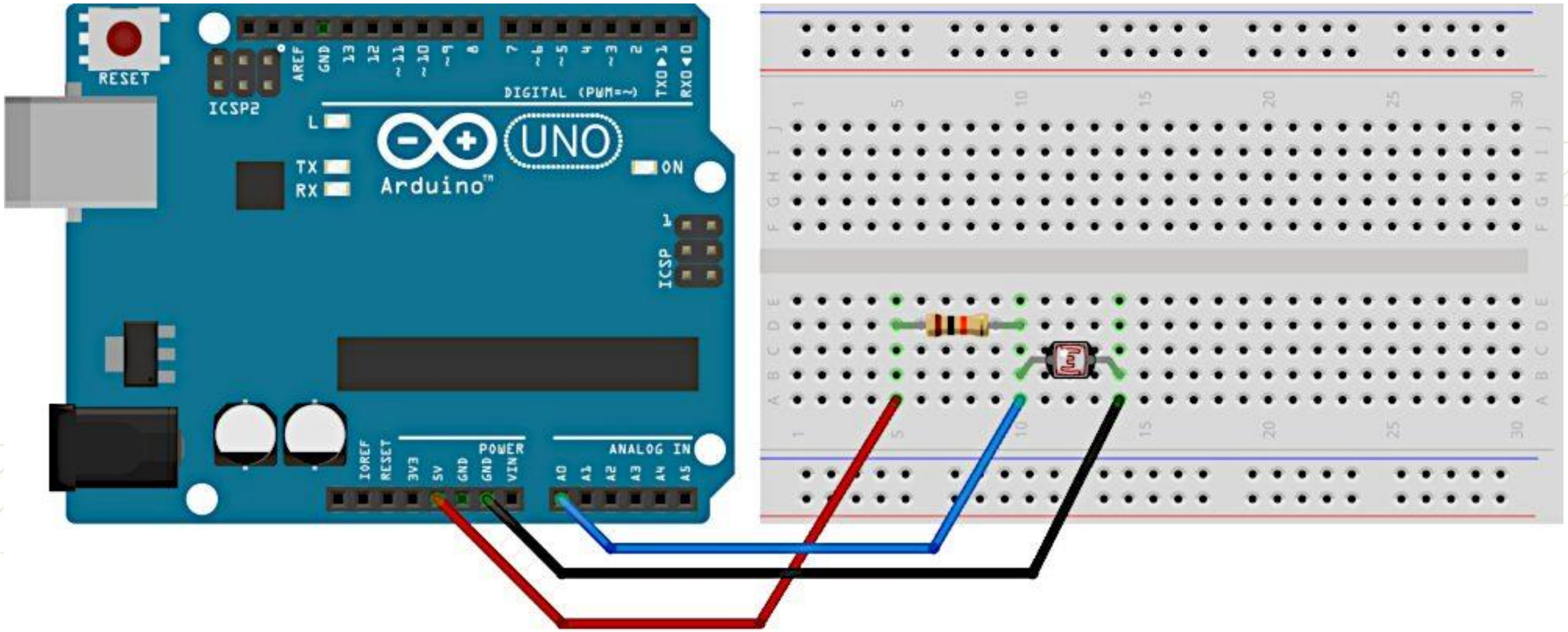
Connection Diagram

SCHOOL OF ENGINEERING AND TECHNOLOGY



- Here, 10k resistor R1 and LDR R2 form a voltage divider.
- i.e, voltage at junction of R1 and R2 is divided voltage from 5V that is across them.
- As light varies on LDR surface, so does its resistance which causes voltage between GND and A0 to vary as well.

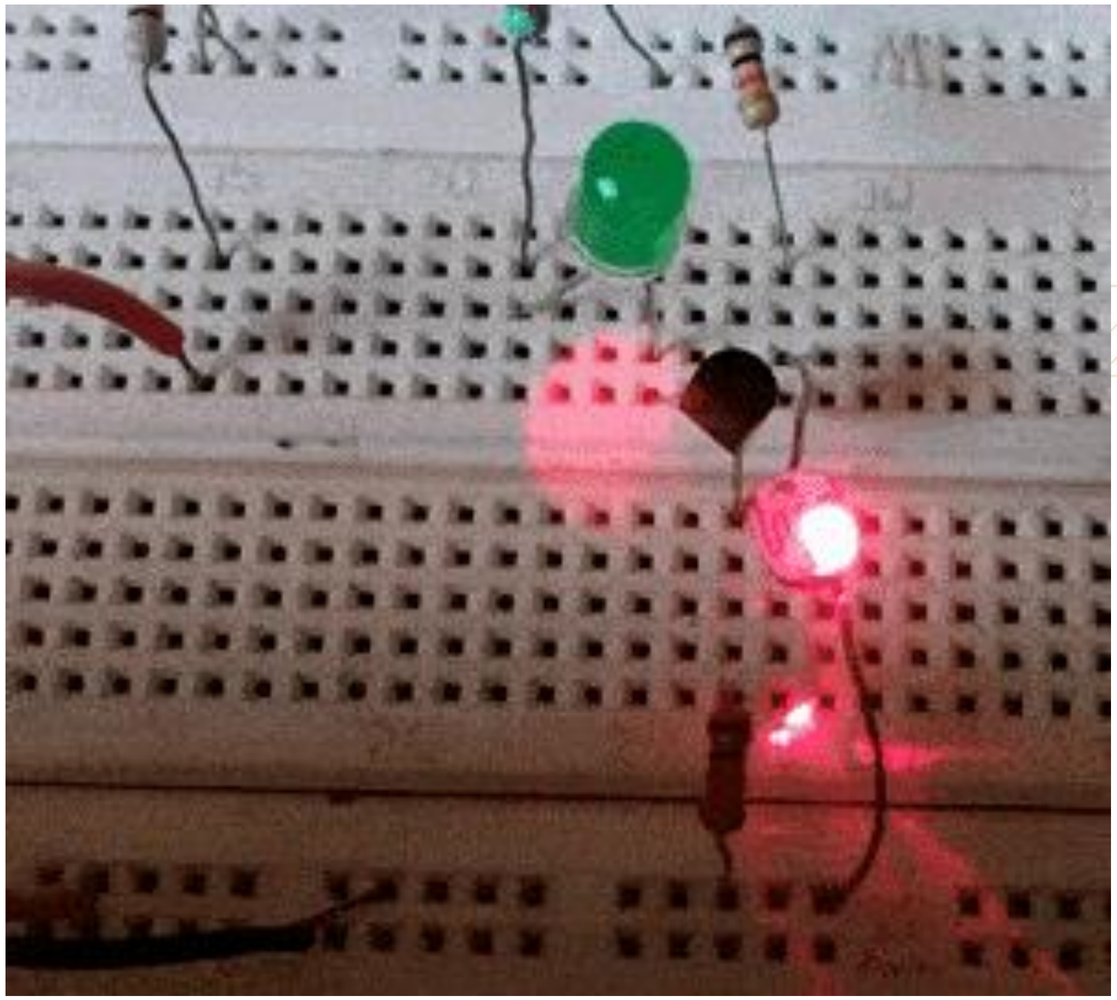
Arduino Uno LDR Breadboard Circuit



Code Switch on an LED when Light Levels are Low

VIPS Technical Campus SCHOOL OF ENGINEERING AND TECHNOLOGY

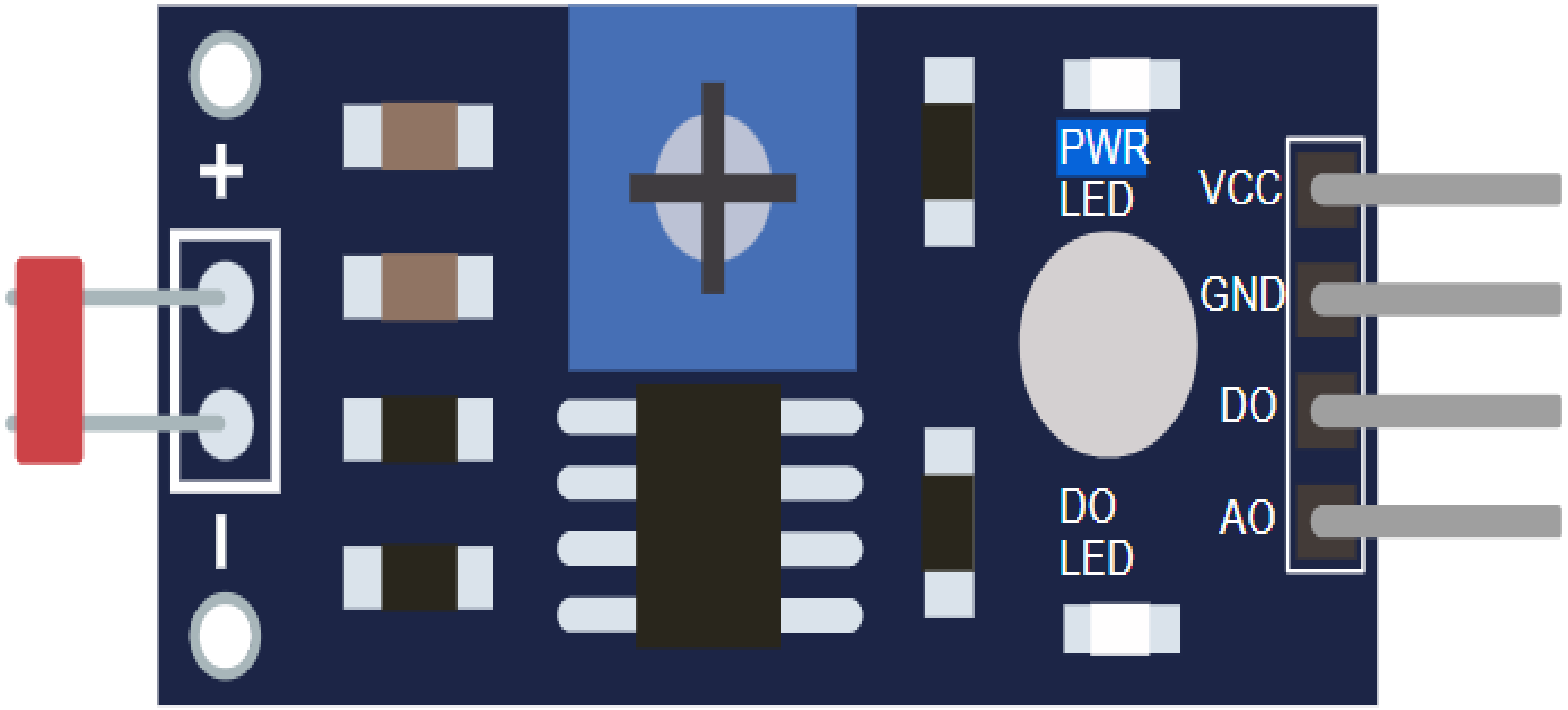
```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  int sensorValue = analogRead(A0);
  if (sensorValue > 700)
  {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else
  {
    digitalWrite(LED_BUILTIN, LOW);
  }
  delay(10);
}
```



LDR Module Pin Description

Pins on sensor,

- **VCC -** Positive power supply,
- **GND -** Ground
- **D_O -** Digital output
- **A_O -** Analog output



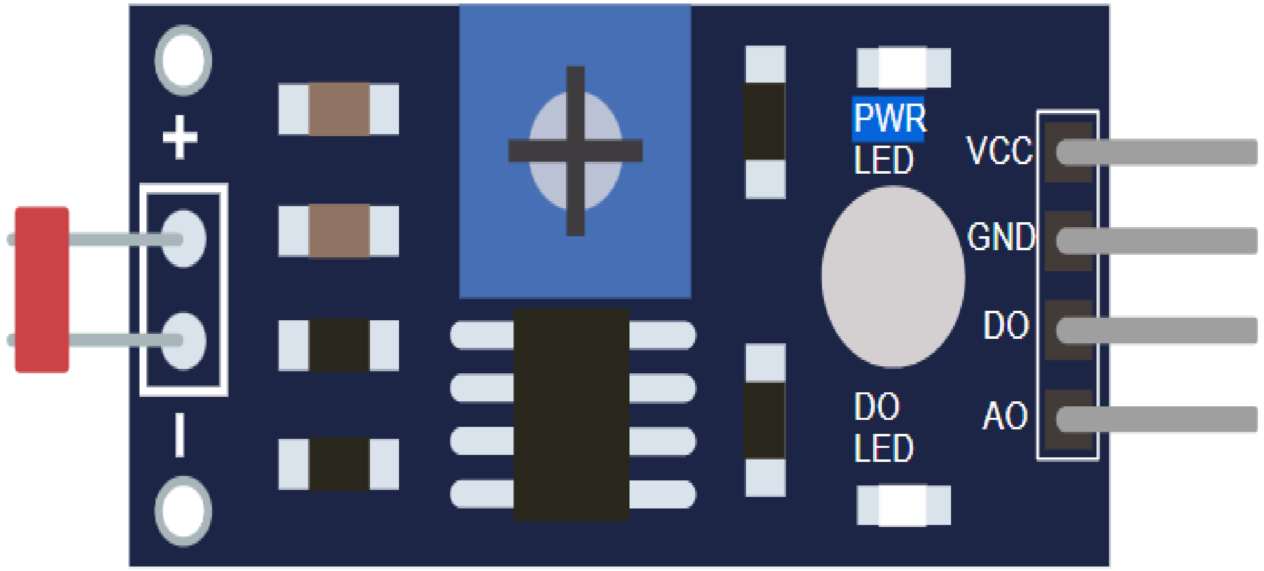
LDR - Operation

- Photoresistor sensor module includes LDR in series with 10K resistor.
- A_O pin is connected between LDR and 10K resistor.
- Voltage on A_O pin depends on illumination.
- Read this voltage by connecting A_O pin of photoresistor sensor to an analog input pin and then using analogRead() function.
- Parameters that control sensitivity of LDR: rl10 and gamma.
- **rl10 –** LDR Resistance at illumination level of 10 lux.
- **Gamma –** determines slope of log(R) / log(lux) graph.

Connect a LDR to an Arduino Uno

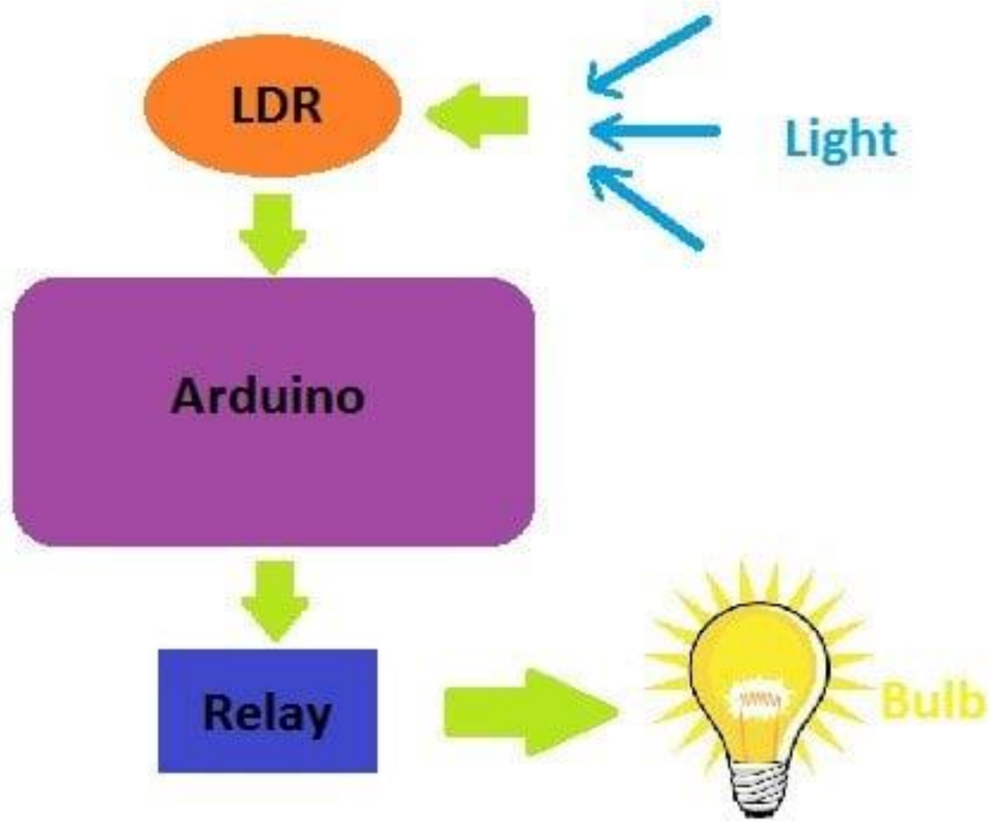
Step 1: Read analog value from LDR circuit. Watch analog value change as light on sensor increases or decreases.

Step 2: Load sketch code that turns on an LED when LDR is darkened. Also switch off LED when LDR is exposed to enough light.



Working of LDR with Arduino Uno

- LDR gives out analog voltage when connected to VCC (5V), which varies with input light intensity.
- i.e., greater the intensity of light, greater will be corresponding voltage from LDR.
- LDR gives out an analog voltage, as it is connected to analog input pin on Arduino.
- Arduino, with its built-in ADC (analog-to-digital converter), converts analog voltage (from 0-5V) into a digital value in range of (0-1023).
- When there is sufficient light in its environment or on its surface, converted digital values read from LDR through Arduino will be in range of 800-1023.



Simple IF statement code

```
void setup() {
   pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
   int sensorValue = analogRead(A0);
   if (sensorValue > 700) {
     digitalWrite(LED_BUILTIN, HIGH);

  }
    delay(10);
}
```

IF-else statement code

```
void setup() {
   pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
   int sensorValue = analogRead(A0);
   if (sensorValue > 700) {
     digitalWrite(LED_BUILTIN, HIGH);

  }
  else {
     digitalWrite(LED_BUILTIN, LOW);

  }
  delay(10);
}
```

While Statement to blink two LEDs for two different conditions

```
int sensorPin = A0;
int light_1 = 13;
int light_2 = 12;
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(light_1, OUTPUT);
  pinMode(light_2, OUTPUT);
}
void loop() {
while (analogRead (sensorPin >100))
  {
    digitalWrite(light_1, HIGH);
    delay(500);
    digitalWrite(light_1, HIGH);
    delay(500);
 }
while (analogRead (sensorPin < 100))
  {
    digitalWrite(light_1, LOW);
    delay(500);
    digitalWrite(light_1, LOW);
    delay(500);
}
}
```

Do while statement used to blink an LED

```
int sensorPin = A1;
int light_1 = 2;
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(light_1, OUTPUT);
}
void loop() {
do
  {
    digitalWrite(light_1, HIGH);
    delay(500);
    digitalWrite(light_1, HIGH);
    delay(500);
 }
while (analogRead (sensorPin > 100));
 }
```

Temperature Sensor

- Simple temperature sensor is a device, to measure temperature through an electrical signal it requires a thermocouple or RTD (Resistance Temperature Detectors).
- **Thermocouple** is prepared by two dissimilar metals which generate electrical voltage indirectly proportional to change temperature.
- **RTD** is a variable resistor, it will change electrical resistance indirectly proportional to changes in temperature in a precise, and nearly linear manner.

Digital Humidity and Temperature Sensor (DHT)

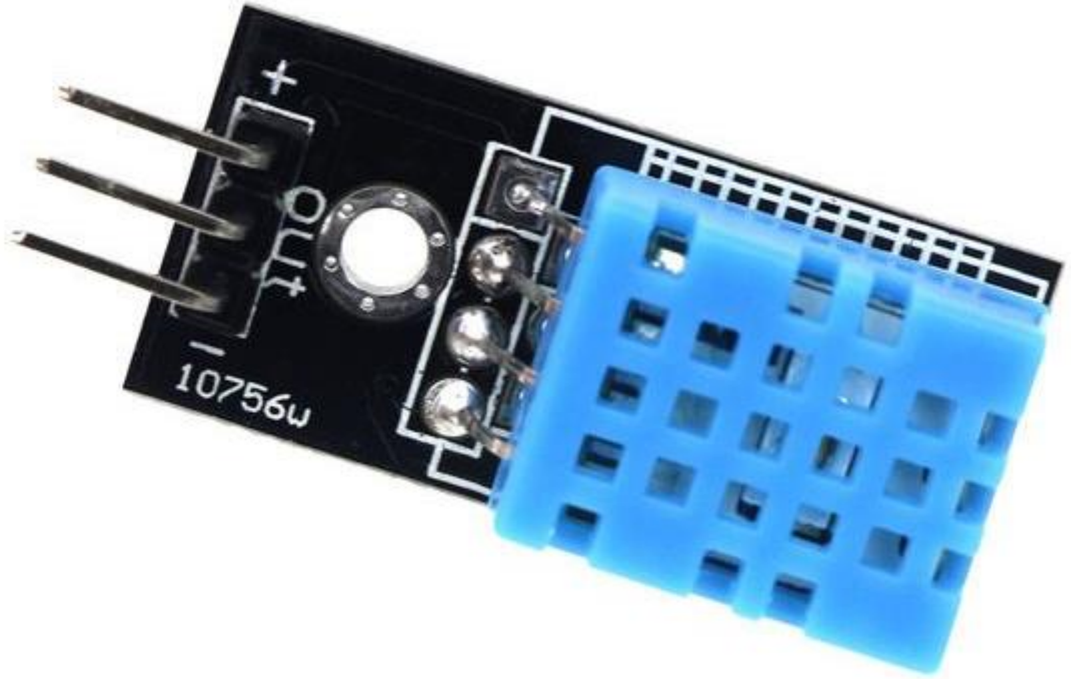
- DHT11 -Low-cost digital sensor for sensing temperature and humidity.
- Easily interface with any microcontroller such as Arduino, Raspberry Pi, etc... to measure humidity and temperature instantaneously.
- Available as a sensor and as a module.
- **Difference:** Pull-up resistor and a power-on LED.
- Has resistive humidity sensing and negative temperature coefficient (NTC).
- 8 bit MCU is connected in it which is responsible for its fast response.

Working of DHT sensor

- Working simple.
- DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature.
- Humidity sensing capacitor has two electrodes with a moisture-holding substrate as a dielectric between them.
- Change in capacitance value occurs with change in humidity levels.
- IC measure, process this changed resistance values and change them into digital form.
- For measuring temperature this sensor uses Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with an increase in temperature.
- To get a larger resistance value even for smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

Specification : DHT22

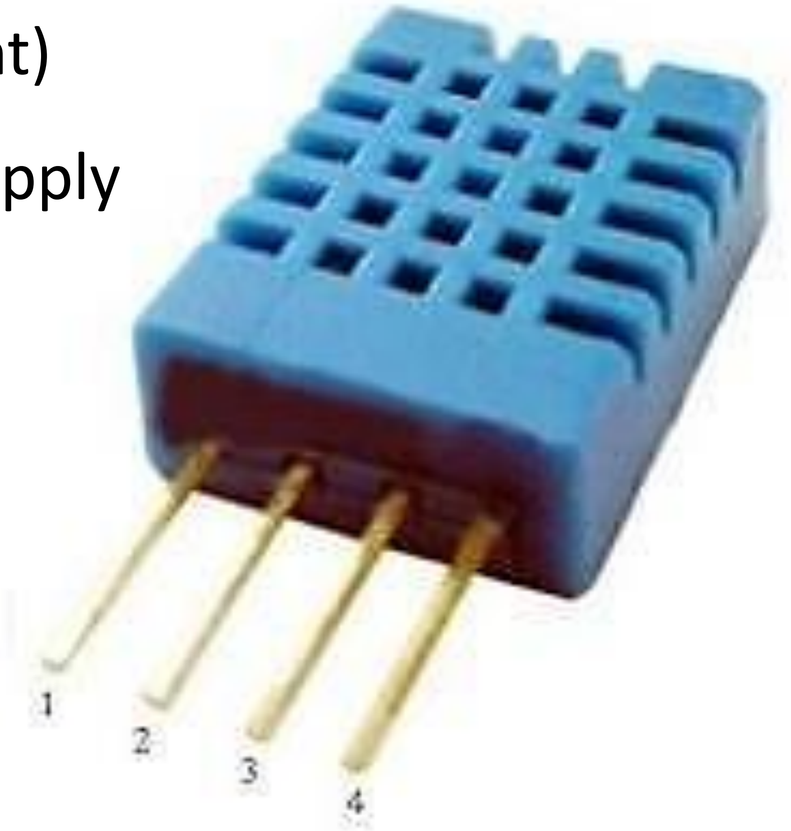
- Humidity range from 20 to 90% RH
- Temperature range from 0 – 50 C
- Signal transmission range of 20 m
- Inexpensive
- Fast response and durable

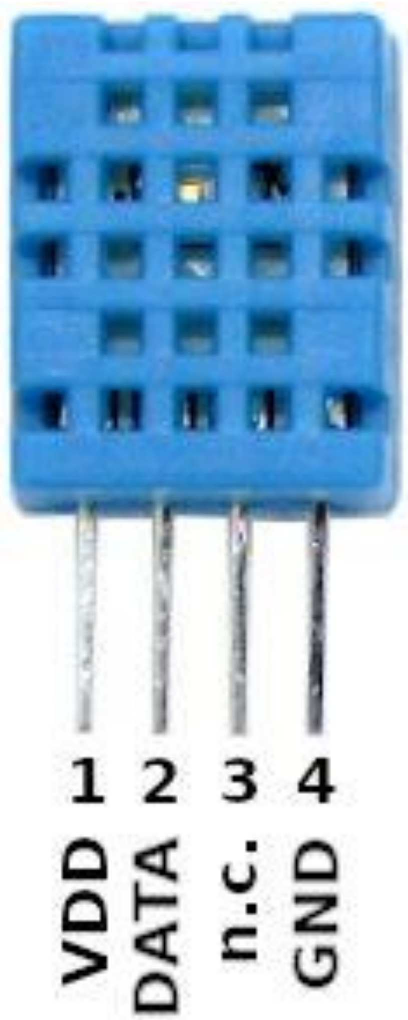


DHT22 Pin out

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

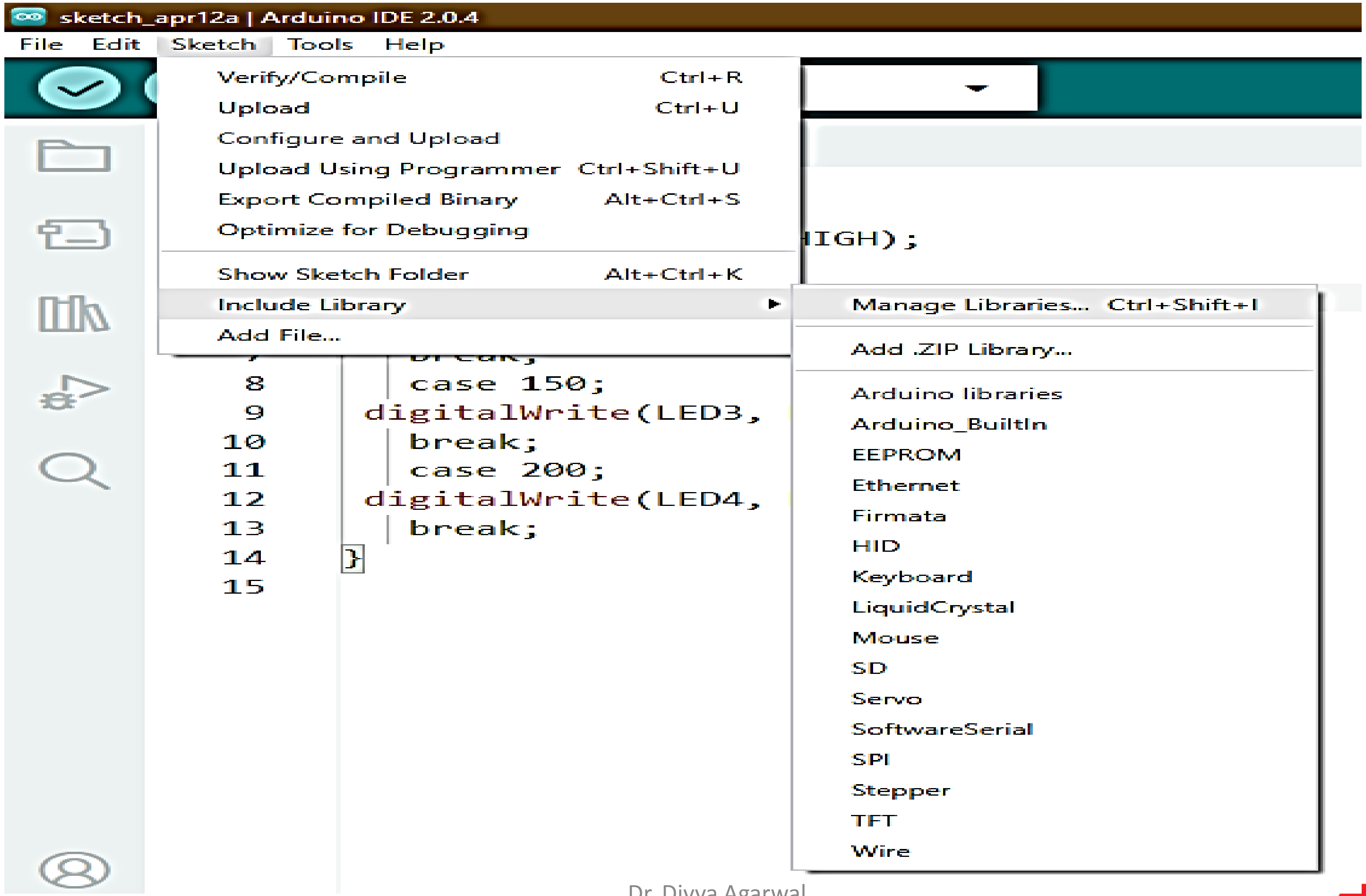




DHT Sensor Library

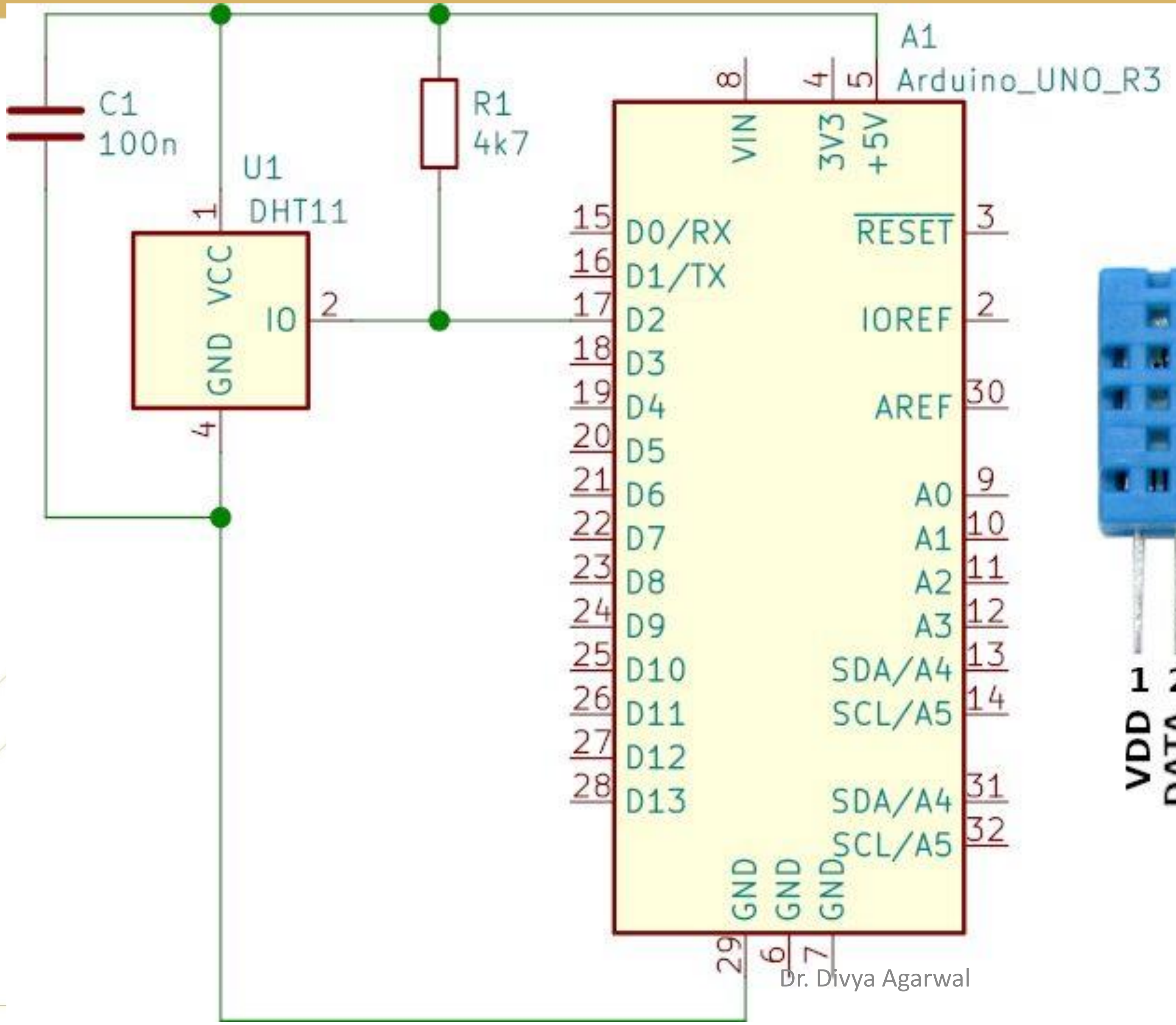
- Arduino supports a special library for DHT11 and DHT22 sensors
- Provides function to read temperature and humidity values from data pin
  1. dht.readHumidity()
  2. dht.readTemperature()
- Install DHT Sensor Library
- Go to **sketch -> Include Library -> Manage Library**

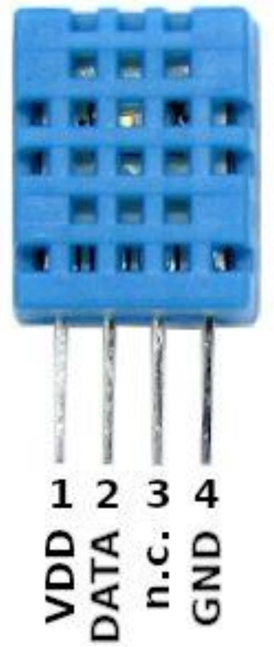
DHT Sensor Library



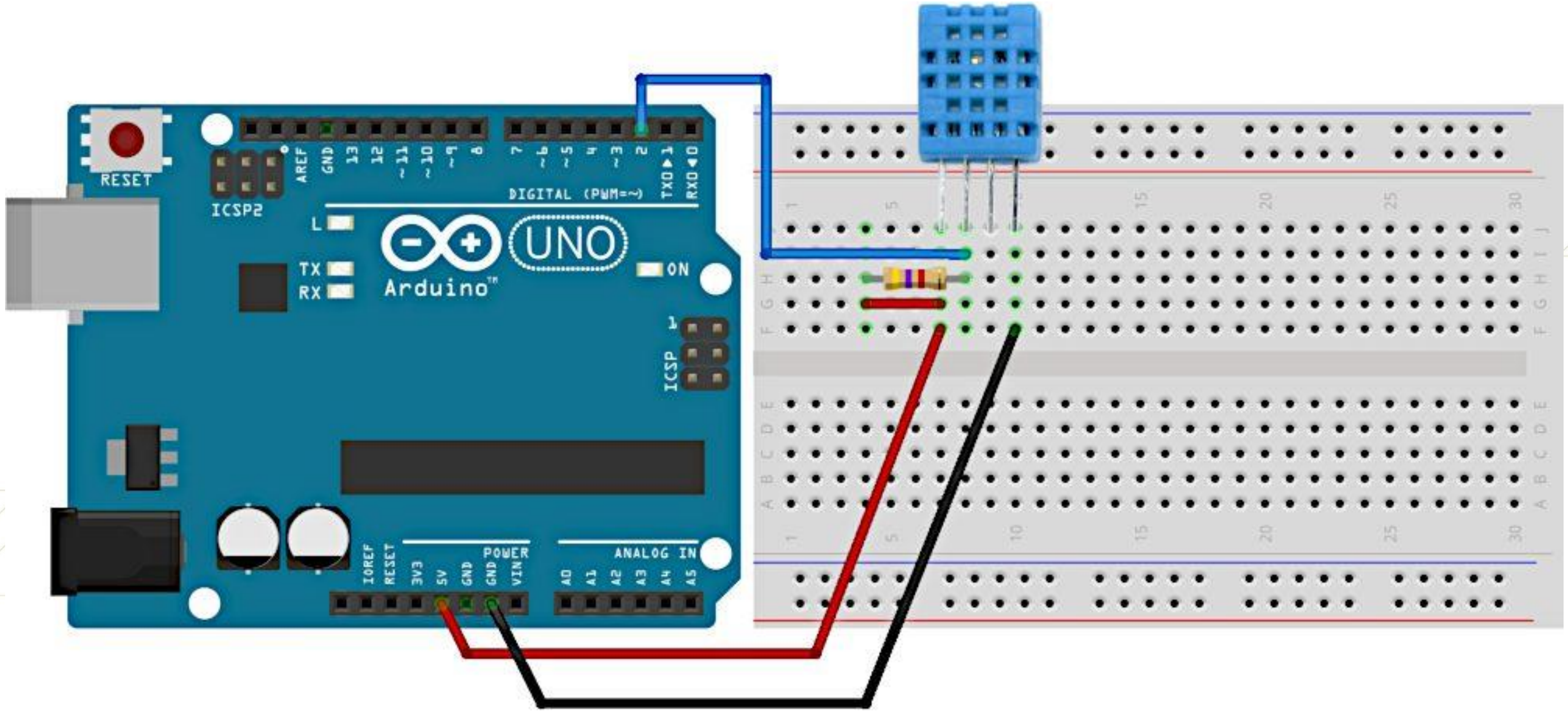
```
IGH);
   case 150;
digitalWrite(LED3,
   break;
   case 200;
digitalWrite(LED4,
   break;
}
```

Arduino DHT22Sensor Circuit Diagram





Arduino DHT22 Sensor Circuit Diagram



DHT 22Code

```
#include <DHT.h>
DHT dht(2, DHT22);
void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print("%  Temperature: ");
  Serial.print(temperature);
  Serial.println("°C ");
  delay(2000);
}
```

DHT 11Code

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT11
// DHT dht(2,DHT11)
DHT dht(DHTPIN, DHTTYPE);
void setup() {
Serial.begin(9600);
Serial.println("DHT11 test!");
dht.begin();
}
void loop() {
delay(2000);
float humidity = dht.readHumidity();
float temperature = dht.readTemperature();
if (isnan(humidity) || isnan(temperature) )
{
Serial.println("Failed to read from DHT sensor!");
return;
}
Serial.print("Humidity: ");
Serial.print(humidity);
Serial.print("% Temperature: ");
Serial.print(temperature);
Serial.print("°C ");
}
```

Difference between DHT 11 and DHT 22

- DHT11 has a humidity range that falls between 5 and 95%
- DHT11 has a temperature range that falls between -20 and 60°C.
- DHT22 features a temperature sensor with high precision, as well as a humidity sensor.
- DHT 22 is highly reliable and has great long-term stability.
- DHT22 has a humidity range that falls between 0 and 100%
- DHT22 has a temperature range that falls between -40 and 80°C.

Blink a RGB LED

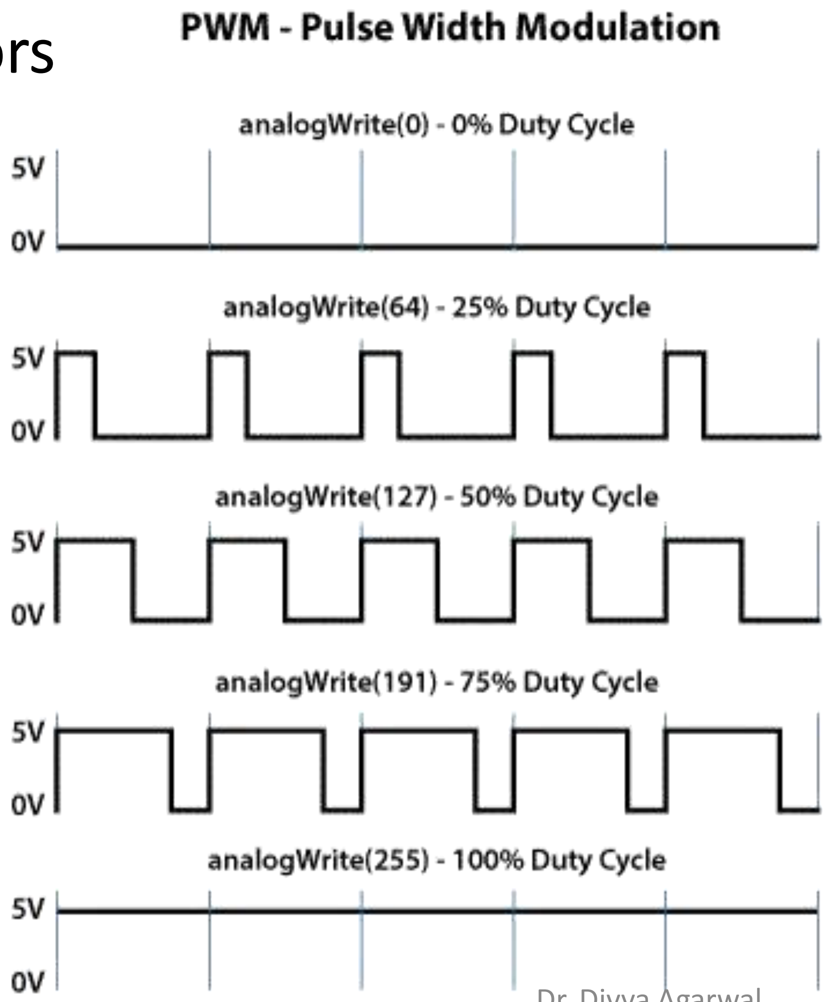
- RGB - Stands for "Red Green Blue."
- RGB refers to three hues of light that can be mixed together to create different colors.
- Combining red, green, and blue light is standard method of producing color images on screens, such as TVs, computer monitors, and smartphone screens.
- **Types of RGB led's: Common cathode** and **Common anode**.
  - ❑ In common cathode RGB led, cathode of all led's is common and PWM signals is given to anode of led's
  - ❑ In common anode RGB led, anode of all led's is common and PWM signals is given to cathode of led's.

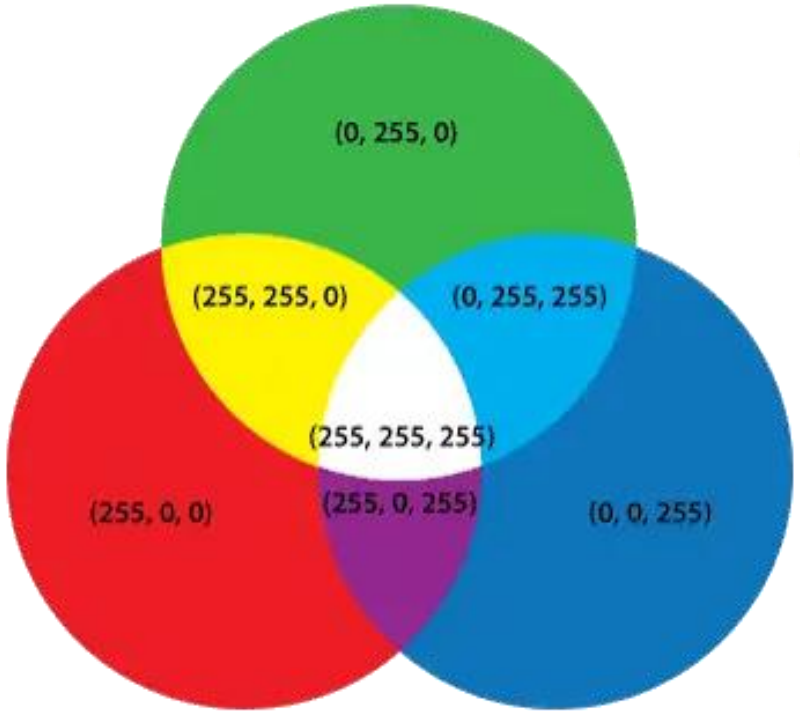
How does an RGB LED work?

- RGB LED can emit different colors by mixing 3 basic colors red, green and blue.
- Consists of 3 separate LEDs red, green and blue packed in a single case.
- Comprises of 4 leads, one lead for each of 3 colors and one common cathode or anode depending on RGB LED type.
- Cathode will be connected to ground and 3 anodes will be connected through 220 Ohms resistors to 3 digital pins on Arduino Board that can provide PWM signal.
- Using PWM for simulating analog output will provide different voltage levels to LEDs to get desired colors

How does an RGB LED work?

- Using PWM for simulating analog output will provide different voltage levels to LEDs to get desired colors

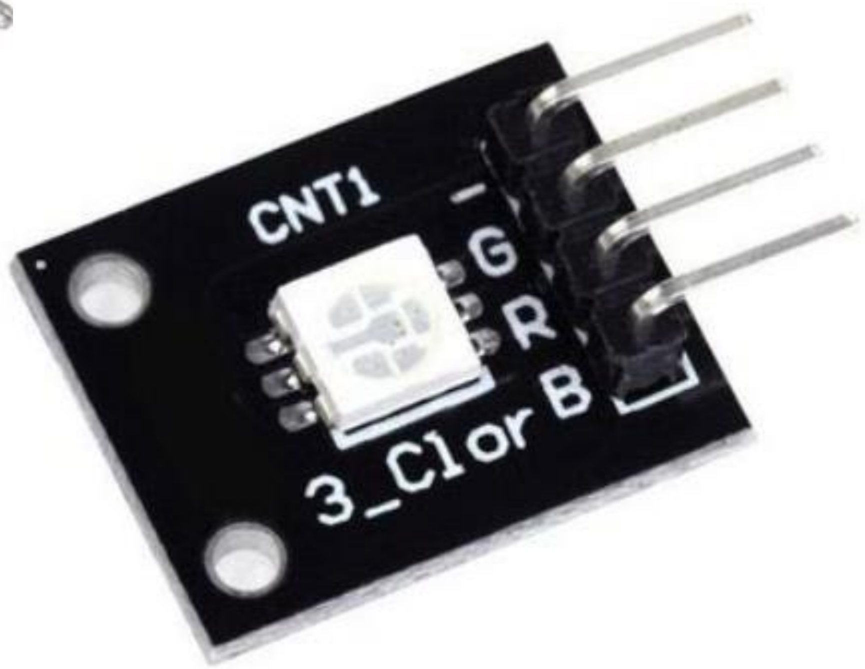

PWM - Pulse Width Modulation
analogWrite(0) - 0% Duty Cycle
analogWrite(64) - 25% Duty Cycle
analogWrite(127) - 50% Duty Cycle
analogWrite(191) - 75% Duty Cycle
analogWrite(255) - 100% Duty Cycle
5V
0V



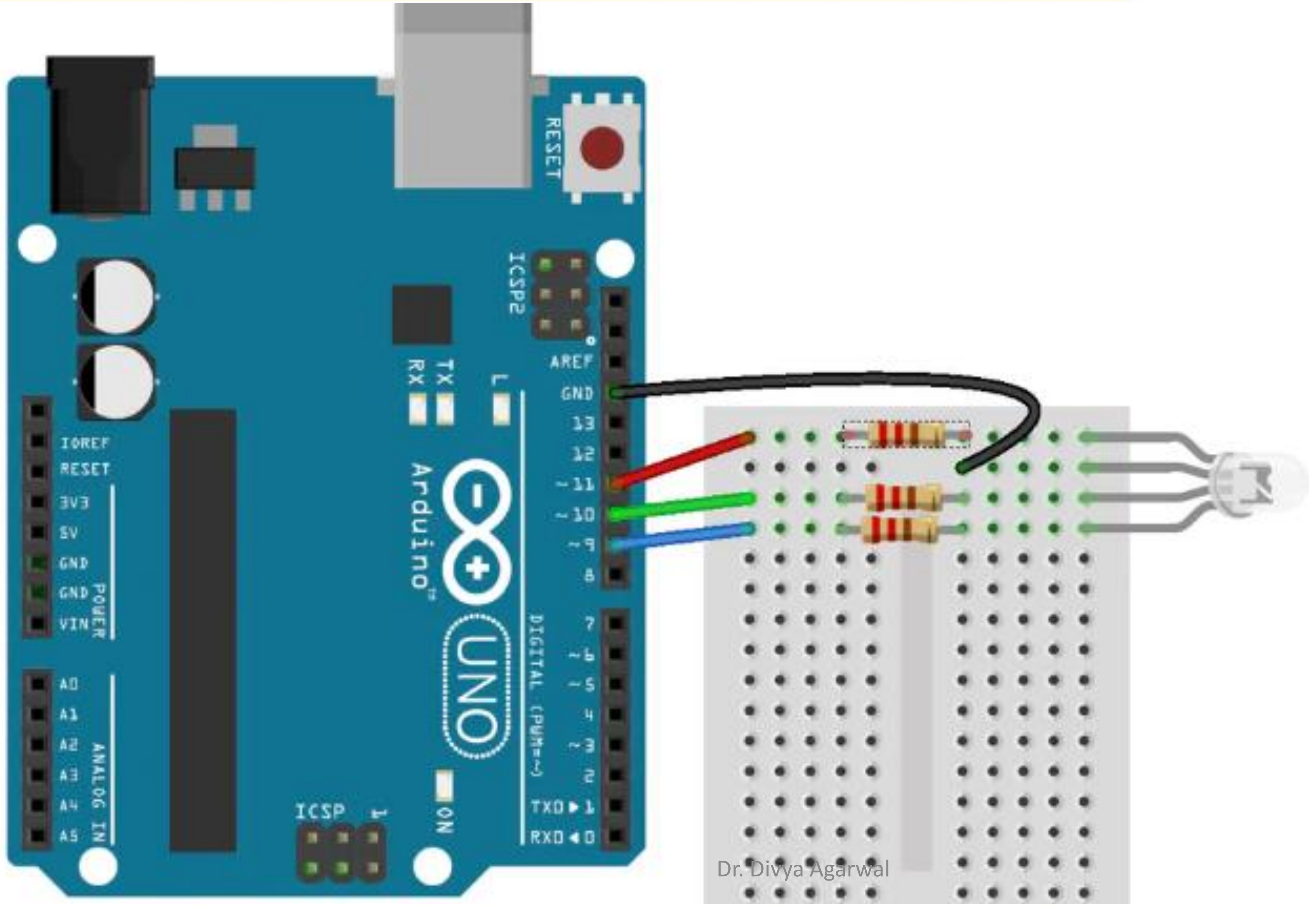
RGB LED Pin out

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

- **Pin D5 - R**
- **Pin D4 - G**
- **Pin D3 - B**



Arduino RGB LED wiring diagram



Connection steps

- Use wiring diagram to connect RGB LED to your Arduino
- Plug Arduino board into your PC
- Check COM ports in Windows device list
- Open Arduino application on your computer
- Upload code to Arduino
- Verify and run

RGB LED Code

SCHOOL OF ENGINEERING AND TECHNOLOGY

```
#define LED1RED 5
#define LED1BLUE 3
#define LED1GREEN 4
void setup() {
pinMode(LED1RED, OUTPUT);
pinMode(LED1BLUE, OUTPUT);
pinMode(LED1GREEN, OUTPUT);
Serial.begin(9600);
}
void loop() {
digitalWrite(LED1RED, HIGH);
Serial.println("LED1RED ");
delay(1000);
digitalWrite(LED1RED, LOW);
Serial.println("LED1RED");
delay(1000);
digitalWrite(LED1BLUE, HIGH);
Serial.println("LED1BLUE");
delay(1000);
digitalWrite(LED1BLUE, LOW);
Serial.println("LED1BLUE");
delay(1000);
digitalWrite(LED1GREEN, HIGH);
Serial.println("LED1GREEN");
delay(1000);
digitalWrite(LED1GREEN, LOW);
Serial.println("LED1GREEN");
delay(1000);
}
```

RGB LED Code

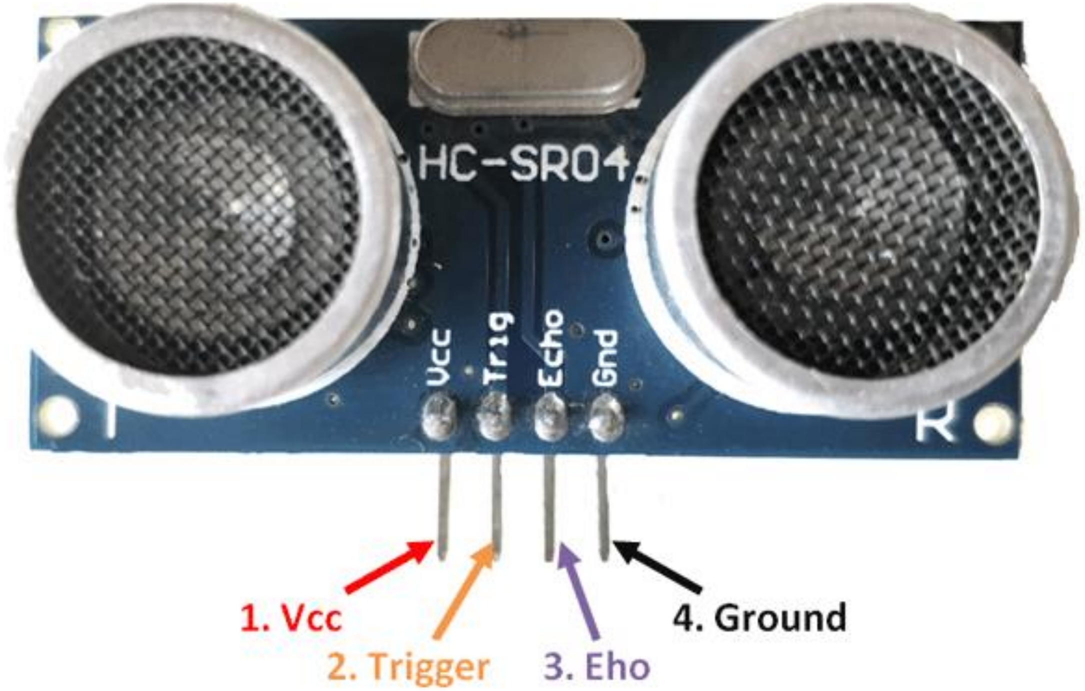
```
int redPin= 7;
int greenPin = 6;
int bluePin = 5;
void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}
void loop() {
  setColor(255, 0, 0); // Red Color
  delay(1000);
  setColor(0, 255, 0); // Green Color
  delay(1000);
  setColor(0, 0, 255); // Blue Color
  delay(1000);
  setColor(255, 255, 255); // White Color
  delay(1000);
 setColor(170, 0, 255); // Purple Color
  delay(1000);
}

void setColor(int redValue, int
greenValue, int blueValue) {
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

Use RGB as Traffic Signal Generator

```
int ledDelay = 10000;
int red = 11;
int yellow = 13;
int green = 12;

void setup() {
pinMode(red,OUTPUT);
pinMode(yellow,OUTPUT);
pinMode(green,OUTPUT);
}

void loop() {
digitalWrite(red,HIGH);
delay(ledDelay);
digitalWrite(yellow,HIGH);
delay(2000);
digitalWrite(green,HIGH);
digitalWrite(red,LOW);
digitalWrite(yellow,LOW);
delay(ledDelay);
digitalWrite(yellow,HIGH);
digitalWrite(green,LOW);
delay(2000);
digitalWrite(yellow,LOW);
}
```

Ultrasonic Sensor

- Ultrasonic sensors measure distance by using ultrasonic waves.
- Sensor head emits an ultrasonic wave and receives wave reflected back from target.
- Ultrasonic Sensors measure distance to target by measuring time between emission and reception.



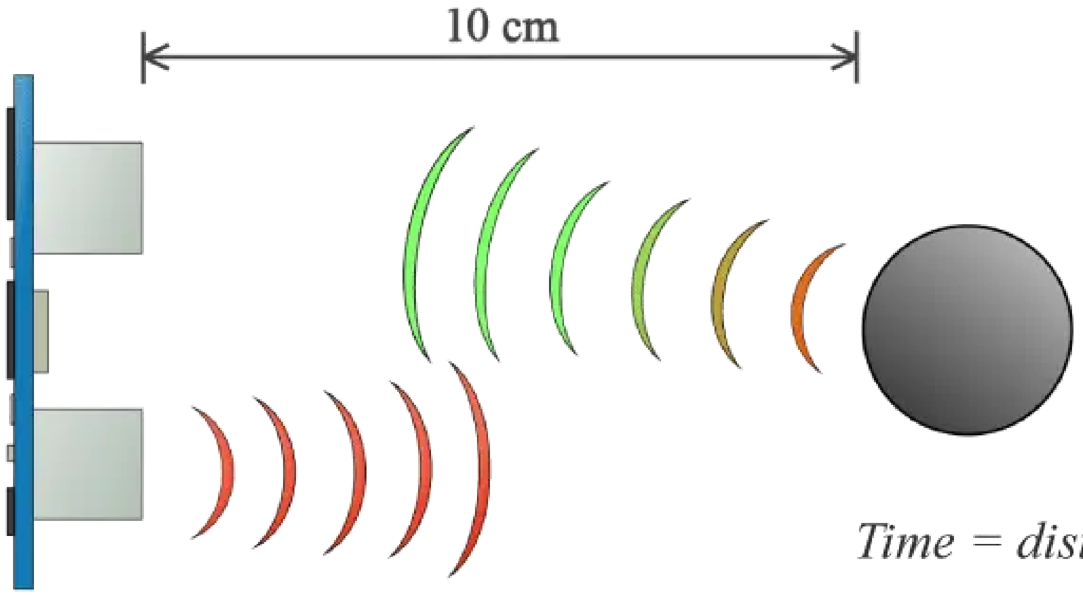
- Optical sensor has a Tx and Rx, whereas an ultrasonic sensor uses a single ultrasonic element for both emission and reception.
- In a reflective model ultrasonic sensor, a single oscillator emits and receives ultrasonic waves alternately. This enables miniaturization of the sensor head.

Ultrasonic Sensor

- HC-SR04 Ultrasonic Module has 4 pins, Ground, VCC, Trig and Echo.
- Ground and VCC pins of module needs to be connected to Ground and 5 volts pins on Arduino Board respectively and trig and echo pins to any Digital I/O pin on Arduino Board.
- To generate ultrasound, set Trig on a High State for 10 us to send out an 8 cycle sonic burst which will travel at speed sound and will be received in Echo pin.
- Echo pin will output time in microseconds sound wave traveled.

Ultrasonic Sensor

- For example, if object is 10 cm away from sensor, and speed of sound is 340 m/s or 0.034 cm/us sound wave will travel about 294 microseconds.
- Echo pin output will be twice because sound wave needs to travel forward and bounce backward.



speed of sound:

$$v = 340\ m/s$$

$$v = 0{,}034\ cm/\mu s$$

Time = distance / speed:

$$t = s / v = 10 / 0{,}034 = 294\ \mu s$$

Distance:

$$s = t \cdot 0{,}034 / 2$$

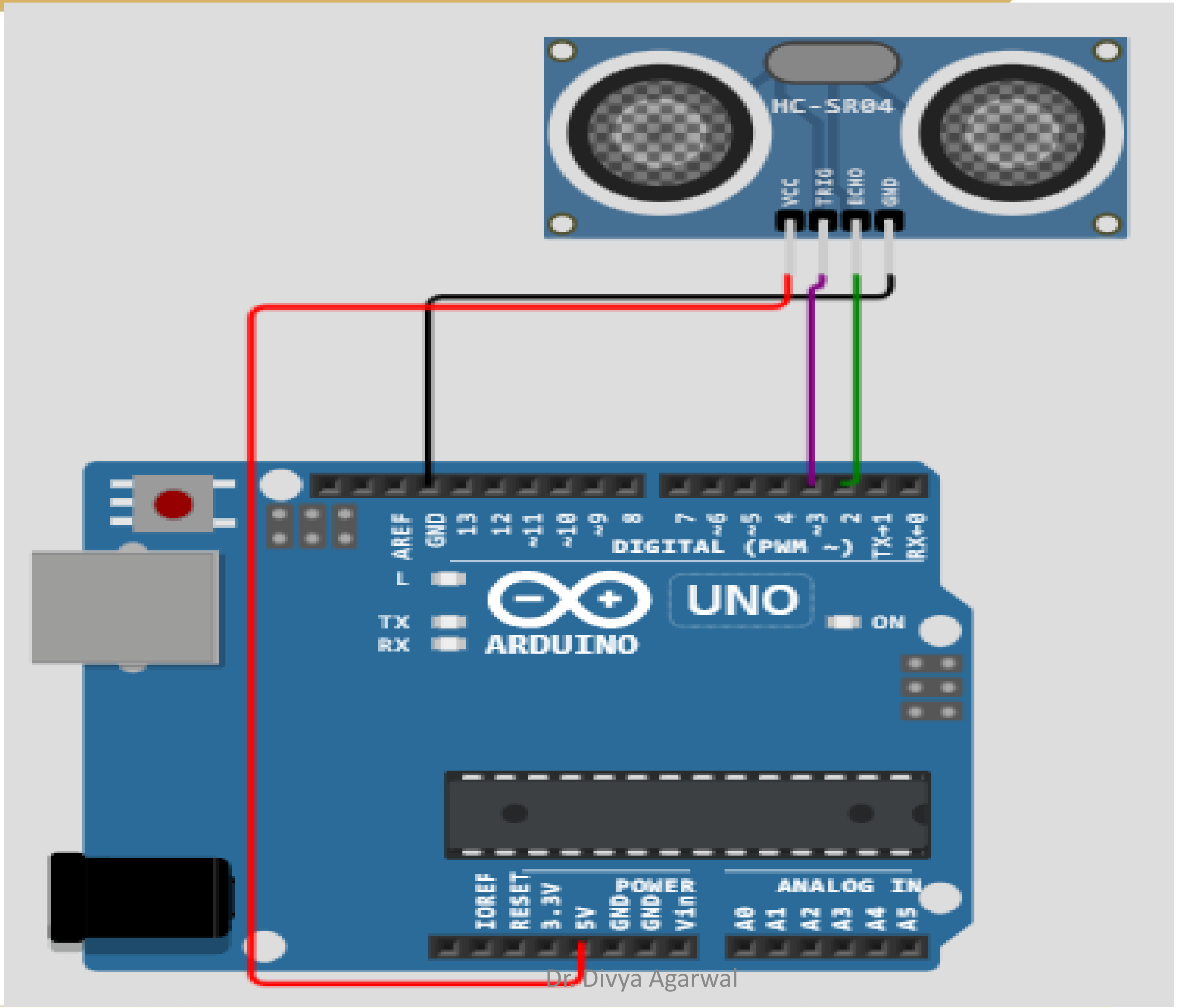
Steps to write Source code for Ultrasonic Sensor

- Define Trig (pin 9) and Echo pins (pin 10) as **trigPin** and **echoPin**.
- Initialize long variable, named "duration" for travel time from sensor and an integer variable for distance.
- In setup() define trigPin as output and echoPin as Input and also start serial communication for showing results on serial monitor.
- In loop() set trigPin to LOW State for just 2 us; to generate Ultra sound wave set trigPin to HIGH State for 10 us.
- Use pulseIn() function to read travel time and put this value into variable "duration".
- This function has 2 parameters, echo pin and HIGH or LOW.

Steps to write Source code for Ultrasonic Sensor

- This function has 2 parameters, echo pin and HIGH or LOW.
- HIGH means pulsIn() function will wait for pin to go HIGH caused by bounced sound wave and it will start timing, then it will wait for pin to go LOW when sound wave will end which will stop timing.
- At the end function will return length of the pulse in microseconds.
- For getting distance, multiply duration by 0.034 and divide it by 2.
- At end, print the value of the distance on the Serial Monitor.

Wiring Connection



Ultrasonic Sensor Code

SCHOOL OF ENGINEERING AND TECHNOLOGY

```
#define trigPin 11
#define echoPin 12
#define ledPin 13

void setup() {
Serial.begin(9600);
pinMode(trigPin,OUTPUT);
pinMode(echoPin,INPUT);
pinMode(ledPin,OUTPUT);
}

void loop() {
long duration , distance;
digitalWrite(trigPin,LOW);
delayMicroseconds(2);
digitalWrite(trigPin,HIGH);
delayMicroseconds(10);
digitalWrite(trigPin,LOW);
duration = pulseIn(echoPin,HIGH);
distance = (duration/2) / 29.1;
if(distance<10)
{
digitalWrite(ledPin,HIGH);
}else{
digitalWrite(ledPin,LOW);
}
Serial.print(distance);
Serial.println("cm");
delay(1500);
}
```

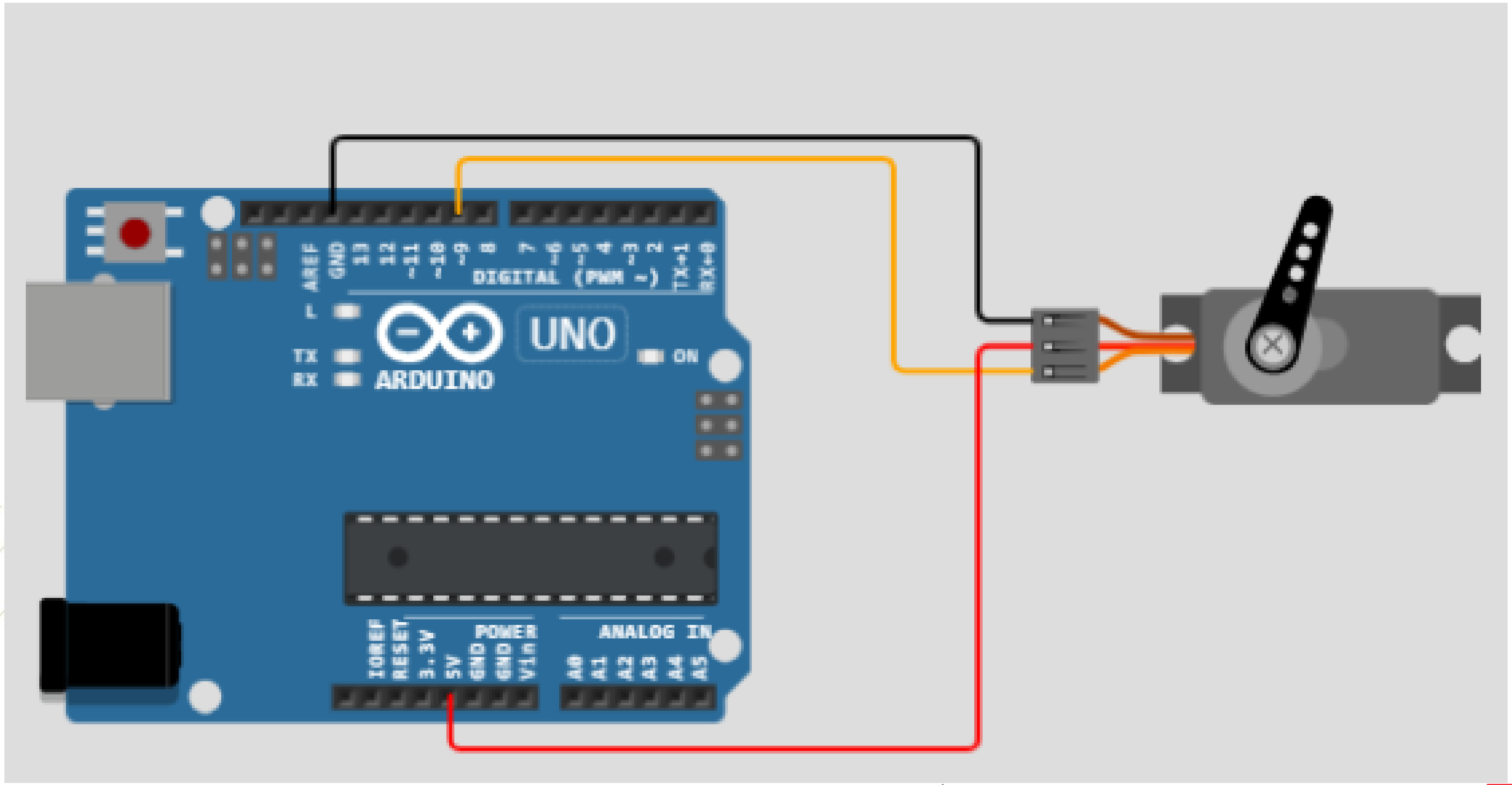
Servo Motor

- A servo motor is a self-contained electrical device, that rotate parts of a machine with high efficiency and with great precision.
- Output shaft of this motor can be moved to a particular angle, position and velocity that a regular motor does not have.
- Servo Motor utilizes a regular motor and couples it with a sensor for positional feedback.
- Servo motor is a closed-loop mechanism that incorporates positional feedback in order to control the rotational or linear speed and position
- Motor is controlled with an electric signal, either analog or digital, which determines the amount of movement which represents the final command position for the shaft.

Servo Motor: Circuit

- Servo motors have three wires: power, ground, and signal.
- Power wire is typically red, and connected to 5V pin on Arduino board.
- Ground wire is typically black or brown and connected to ground pin on board.
- Signal pin is typically yellow or orange and should be connected to PWM pin on board. In

Circuit Diagram



Servo Motor Code

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
int pos = 0; // variable to store the servo position
void setup() {
myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
for (pos = 0; pos <= 180; pos += 1)
{
myservo.write(pos); // tell servo to go to position in variable 'pos'
delay(15); // waits 15ms for the servo to reach the position
}
for (pos = 180; pos >= 0; pos -= 1)
{
myservo.write(pos); // tell servo to go to position in variable 'pos'
delay(15); // waits 15ms for the servo to reach the position
}
}
```

Micro Servo Code

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
void setup() {
Serial.begin(9600);
myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
int val;
while (Serial.available()>0)
{
val = Serial.parseInt();
if(val!=0)
{
Serial.println(val);
myservo.write(val);
}
delay(15);
}
```

LCD 16*2 Pin Connections

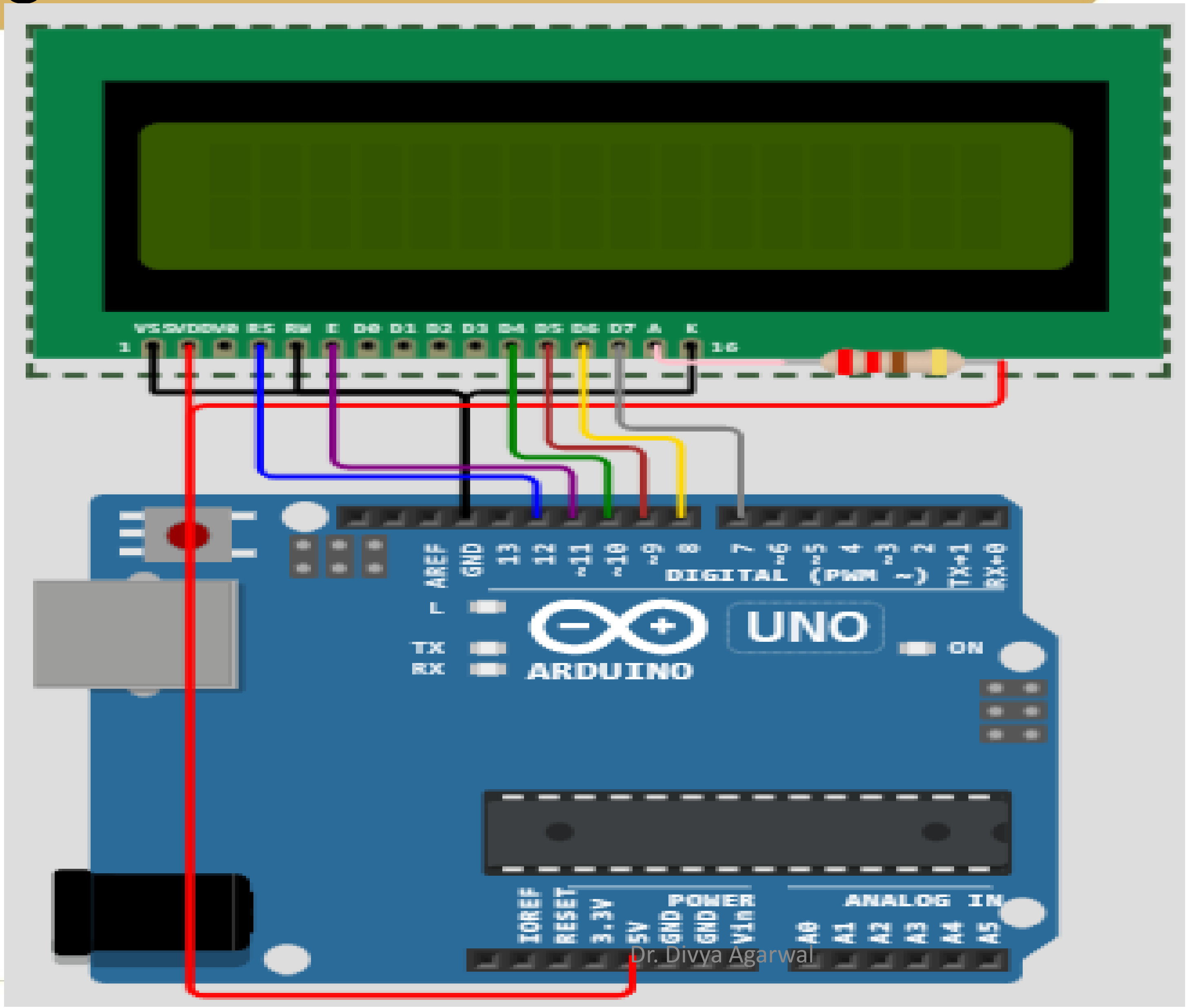
Name	Description	Arduino Pin*
VSS	Ground	GND.1
VDD	Supply voltage	5V
V0	Contrast adjustment (not simulated)	
RS	Command/Data select	12
RW	Read/Write. Connect to Ground.	GND.1
E	Enable	11
D0 – D3	Parallel data 0 - 3 (optional) †	
D4	Parallel data 4	10
D5	Parallel data 5	9
D6	Parallel data 6	8
D7	Parallel data 7	7
A	Backlight anode	5V / 6‡
K	Backlight cathode	GND.1

LCD 16*2 Pin Connections

Name	Description	Arduino Pin*
VSS	Ground	GND.1
VDD	Supply voltage	5V
V0	Contrast adjustment (not simulated)	
RS	Command/Data select	12
RW	Read/Write. Connect to Ground.	GND.1
E	Enable	11
D0 – D3	Parallel data 0 - 3 (optional) †	
D4	Parallel data 4	10
D5	Parallel data 5	9
D6	Parallel data 6	8
D7	Parallel data 7	7
A	Backlight anode	5V / 6‡
K	Backlight cathode	GND.1

SCHOOL OF ENGINEERING AND TECHNOLOGY

LCD wiring Connections


VSS VDD V0 RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K
1
16
AREF GND 13 12 ~11 ~10 ~9 8 7 ~6 ~5 4 ~3 2 TX→1 RX←0
DIGITAL (PWM ~)
L
TX
RX
UNO
ARDUINO
ON
IOREF RESET 3.3V 5V GND GND Vin
POWER
ANALOG IN
A0 A1 A2 A3 A4 A5

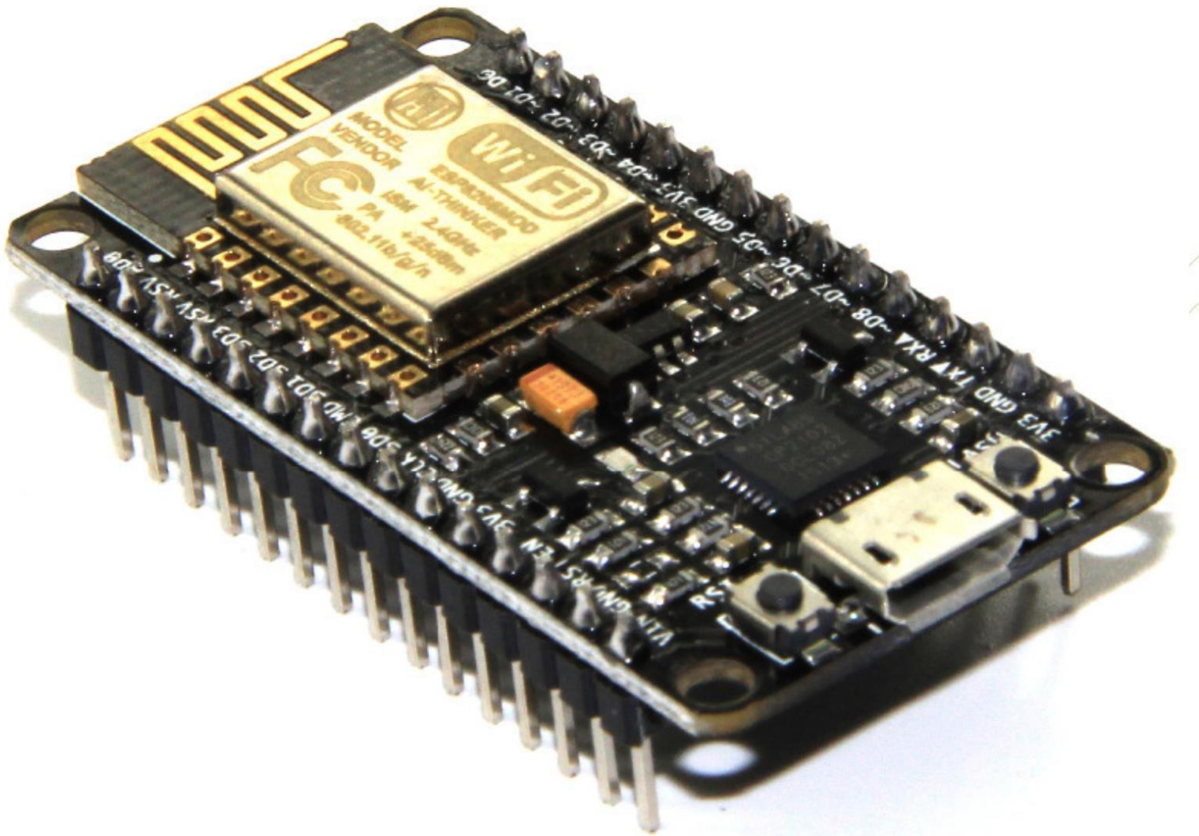
LCD16*02 to Arduino Uno Code

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 10, 9, 8, 7);

void setup() {
  lcd.begin(16, 2);
  // you can now interact with the LCD, e.g.:
  lcd.print("Hello World!");
}

void loop() {
  // ...
}
```

SCHOOL OF ENGINEERING AND TECHNOLOGY

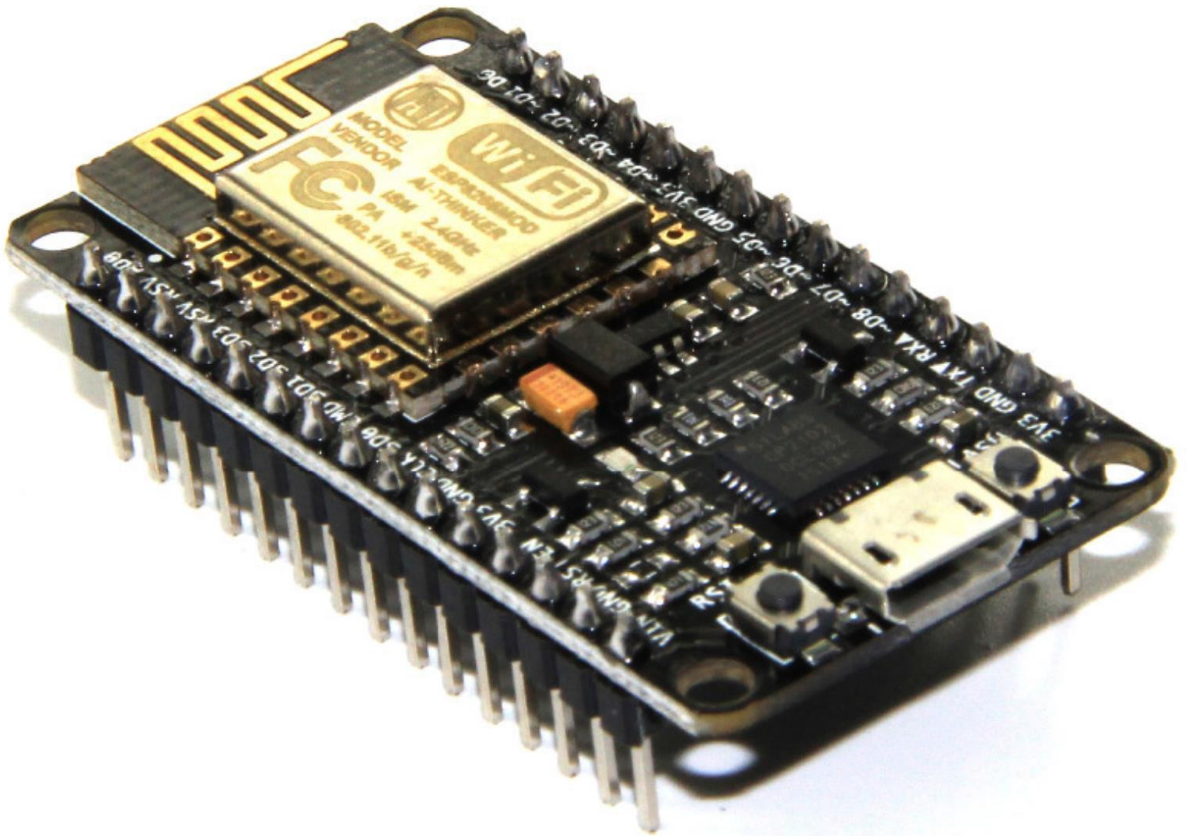
ESP8266 NodeMCU

- NodeMCU is an open-source firmware and development kit that helps to prototype or build IoT products.
- Includes firmware that runs on ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on ESP-12 module.
- Firmware uses Lua scripting language.
- Based on eLua project and built on Espressif Non-OS SDK for ESP8266.



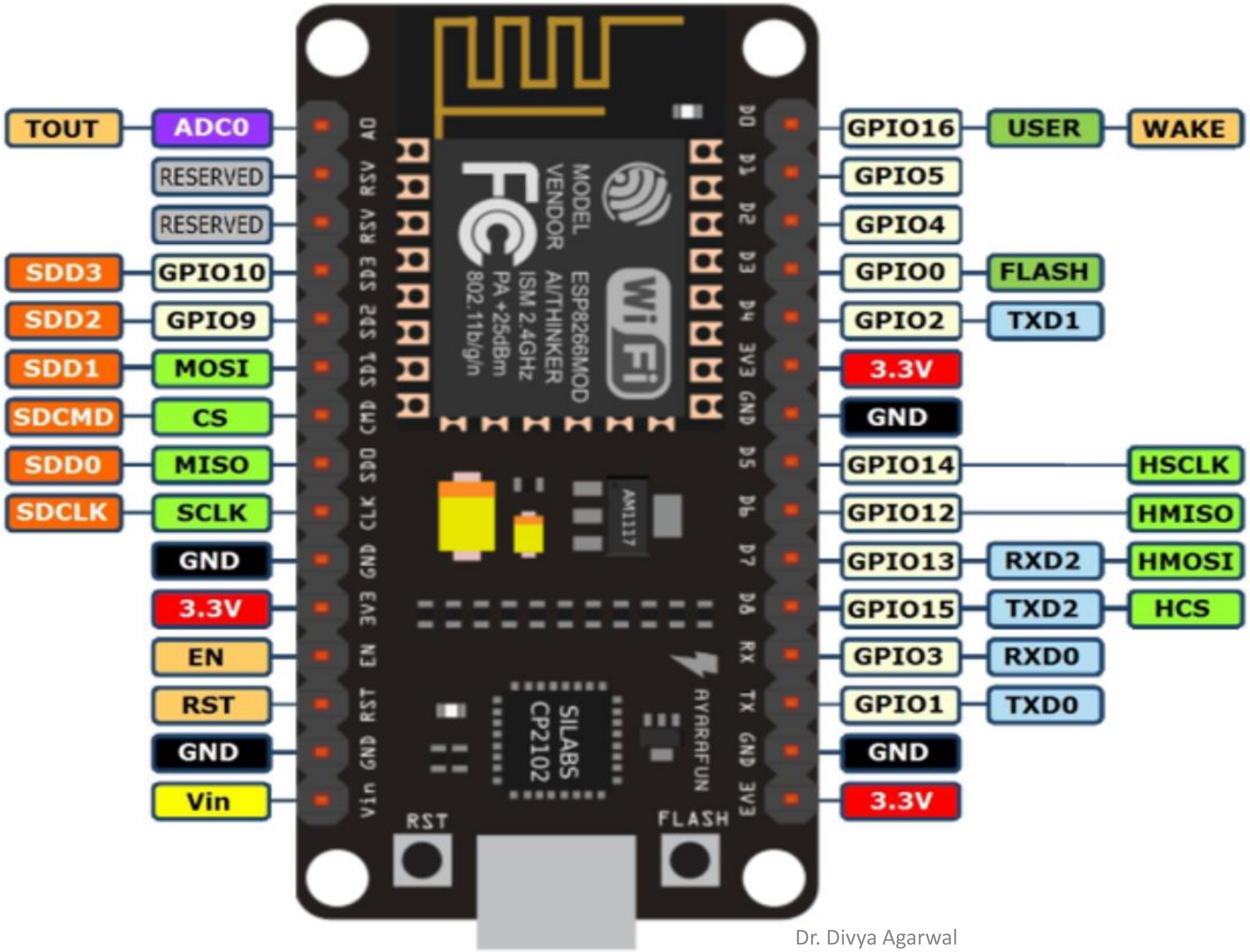
ESP8266 NodeMCU

- MCU stands for MicroController Unit - which means it is a computer on a single chip.
- Microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals.
- They are used to automate automobile engine control, implantable medical devices, remote controls, office machines, appliances, power tools, toys etc.



VIPS Technical Campus योग: कर्मसु कौशलम् IN PURSUIT OF PERFECTION SCHOOL OF ENGINEERING AND TECHNOLOGY

ESP8266 NodeMCU Pin Description



Dr. Divya Agarwal

ESP8266 NodeMCU Pin Description and Specifications

- NodeMCU_ESP8266 has 30 pins in total out of which there are 17 GPIO pins.
- GPIO stands for General Purpose Input Output.
- 9 digital pins ranging from D0-D8
- One analog pin A0, which is a 10 bit ADC.
- D0 pin can only be used to read or write data and can't perform other options.
- ESP8266 chip is enabled when EN pin is pulled HIGH and is disabled when EN pin is pulled LOW.
- Board has a 2.4 GHz antenna for a long-range of network
- CP2102 is USB to TTL converter.
- ESP-12E module containing ESP8266 chip having Tensilica Xtensa® 32-bit LX106 RISC microprocessor which operates at 80 to 160 MHz adjustable clock frequency and supports RTOS.

ESP8266 NodeMCU Pin Description and Specifications

- 128 KB RAM and 4MB of Flash memory
- ESP8266 Integrates 802.11b/g/n HT40 Wi-Fi transceiver, so it can connect to WiFi network for interacting with Internet and can also set up a network of its own, allowing other devices to connect directly to it which makes it even more versatile.
- ESP8266 has 2 onboard buttons along with an on-board LED which connects with D0 PIN.
- Two buttons are FLASH and RST.
  - **FLASH pin–** It is to download new programs to the board
  - **RST pin –** It is to reset the ESP8266 chip
- ESP8266 can be used are:
  - Making a web server using ESP8266; Controlling DHT11 using the NodeMCU; ESP8266 weather station-using BMP280; ESP8266 NTP server for fetching time

Lua Programming

- Lua is a powerful, efficient, lightweight, embeddable scripting language.
- Supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.
- Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics.
- Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.
- Lua is a proven, robust language which is powerful and available for free.
- Lua is fast, portable, embeddable

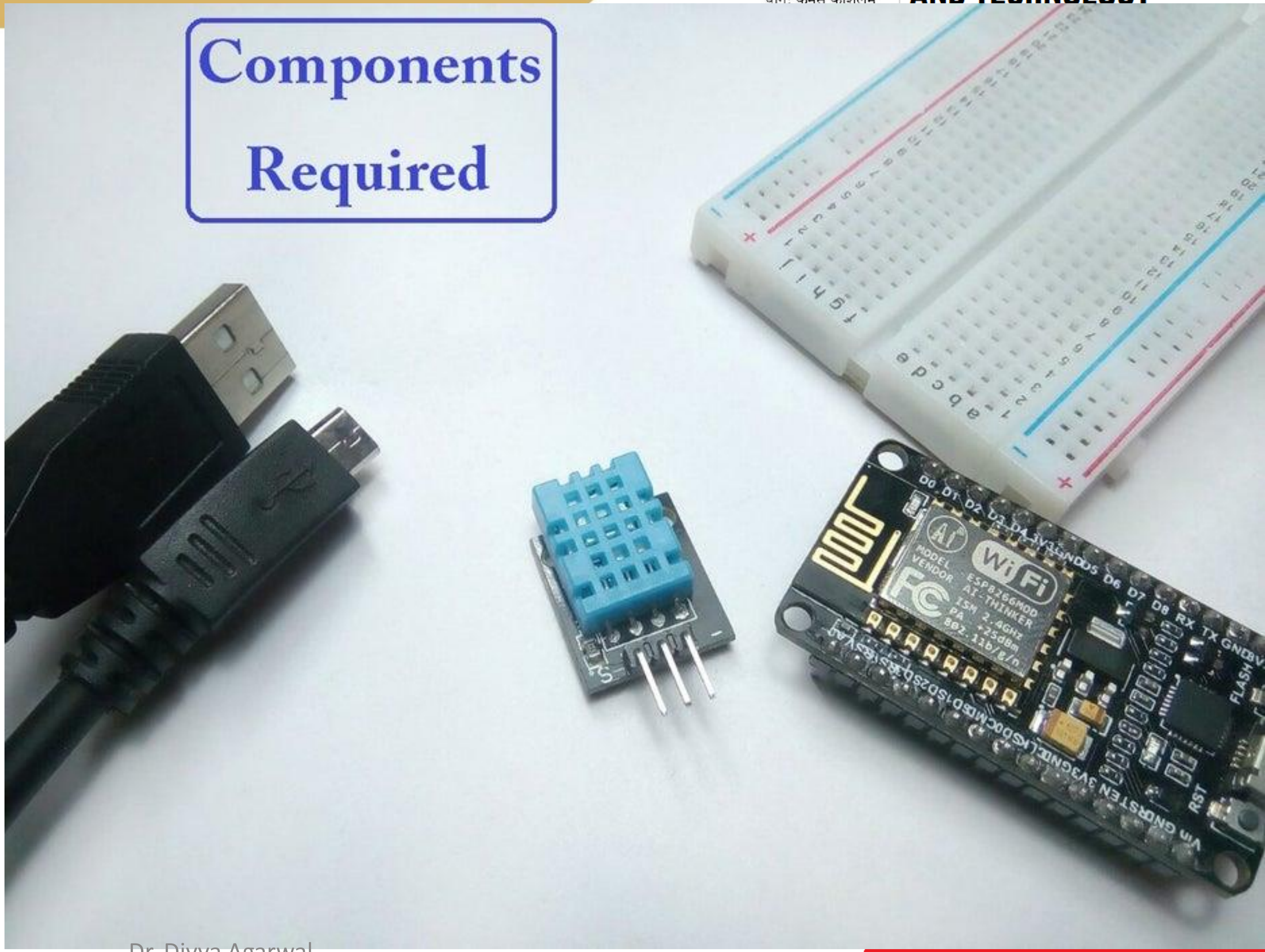
Programming Model

- NodeMCU programming model is similar to that of Node.js, only in Lua.
- It is asynchronous and event-driven.
- Includes callback functions.

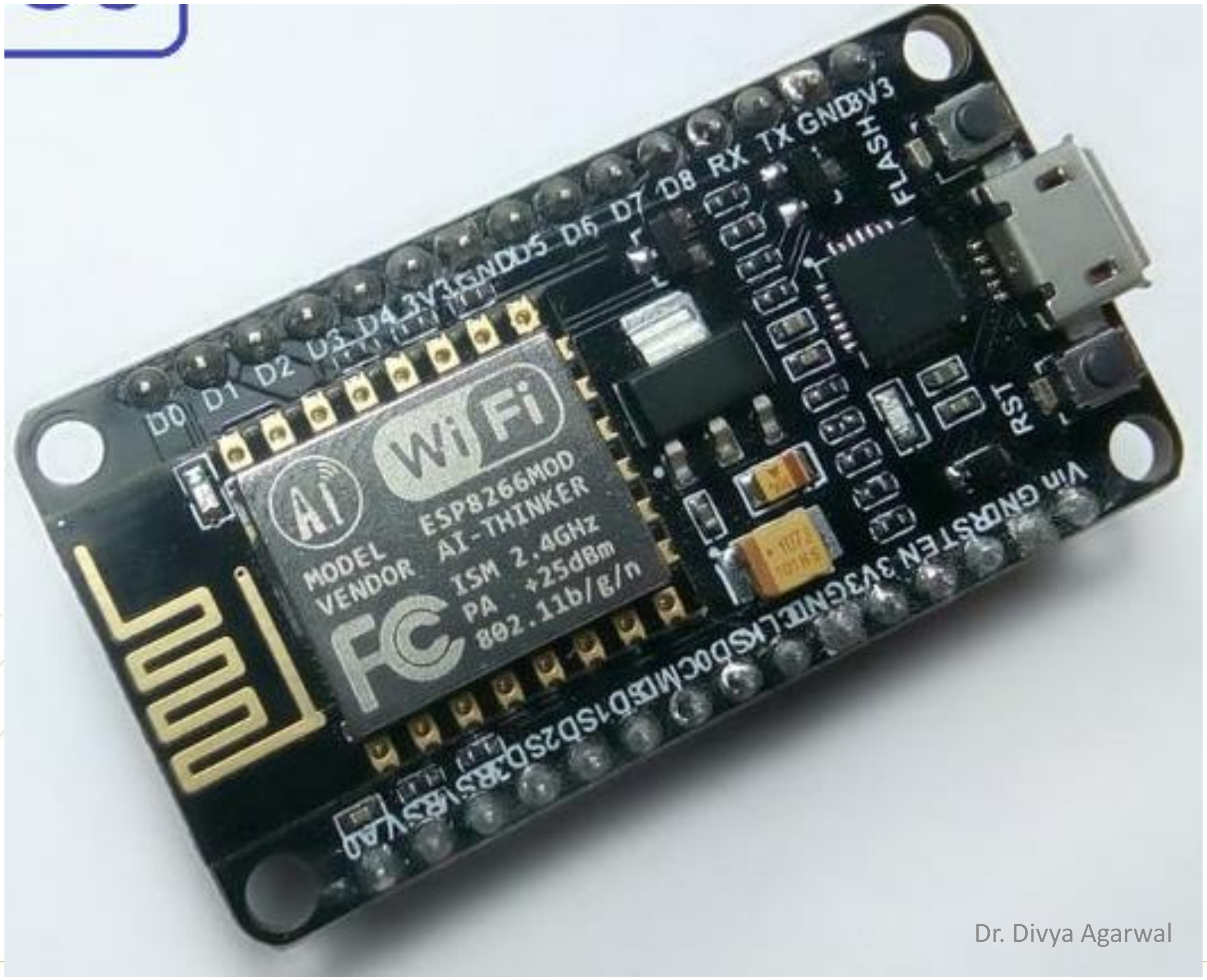
Interface DHT11 (Humidity Sensor) Using NodeMCU

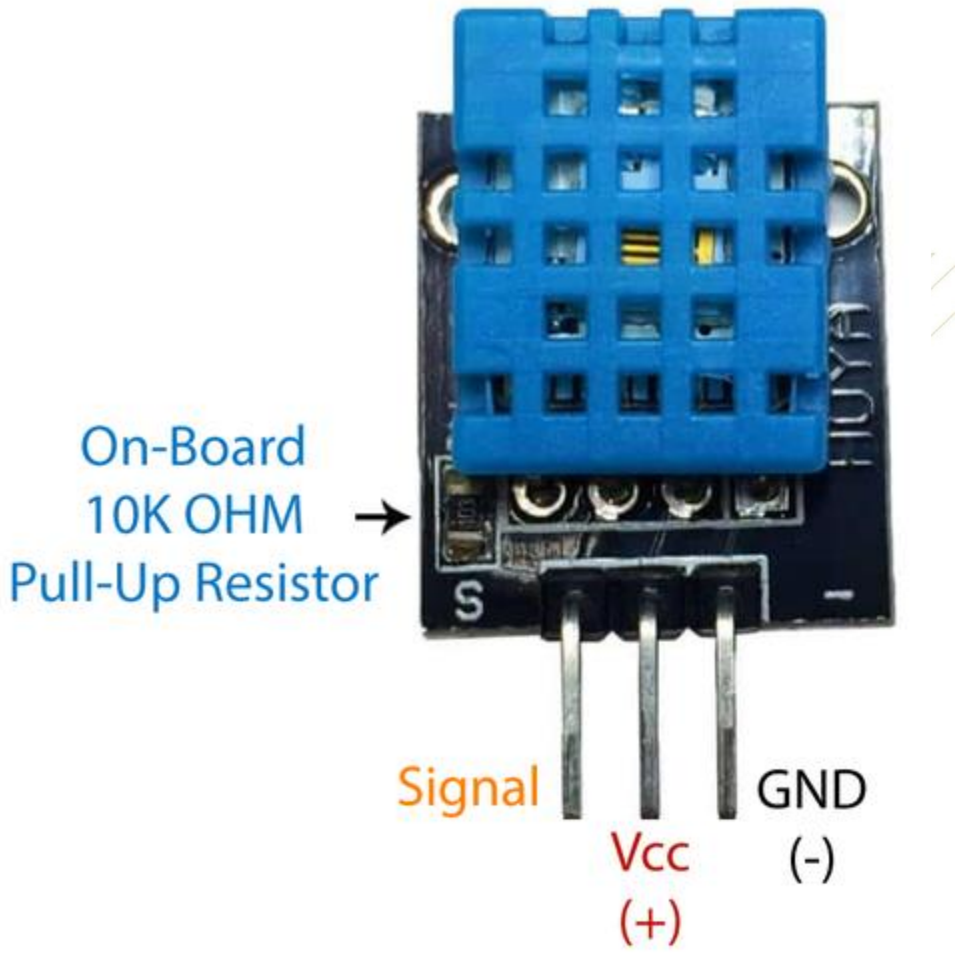
Components Required

- Breadboard
- Micro USB Cable
- ESP8266 NodeMCU
- DHT11 Humidity and Temperature
- Jumper Wires
- Arduino IDE



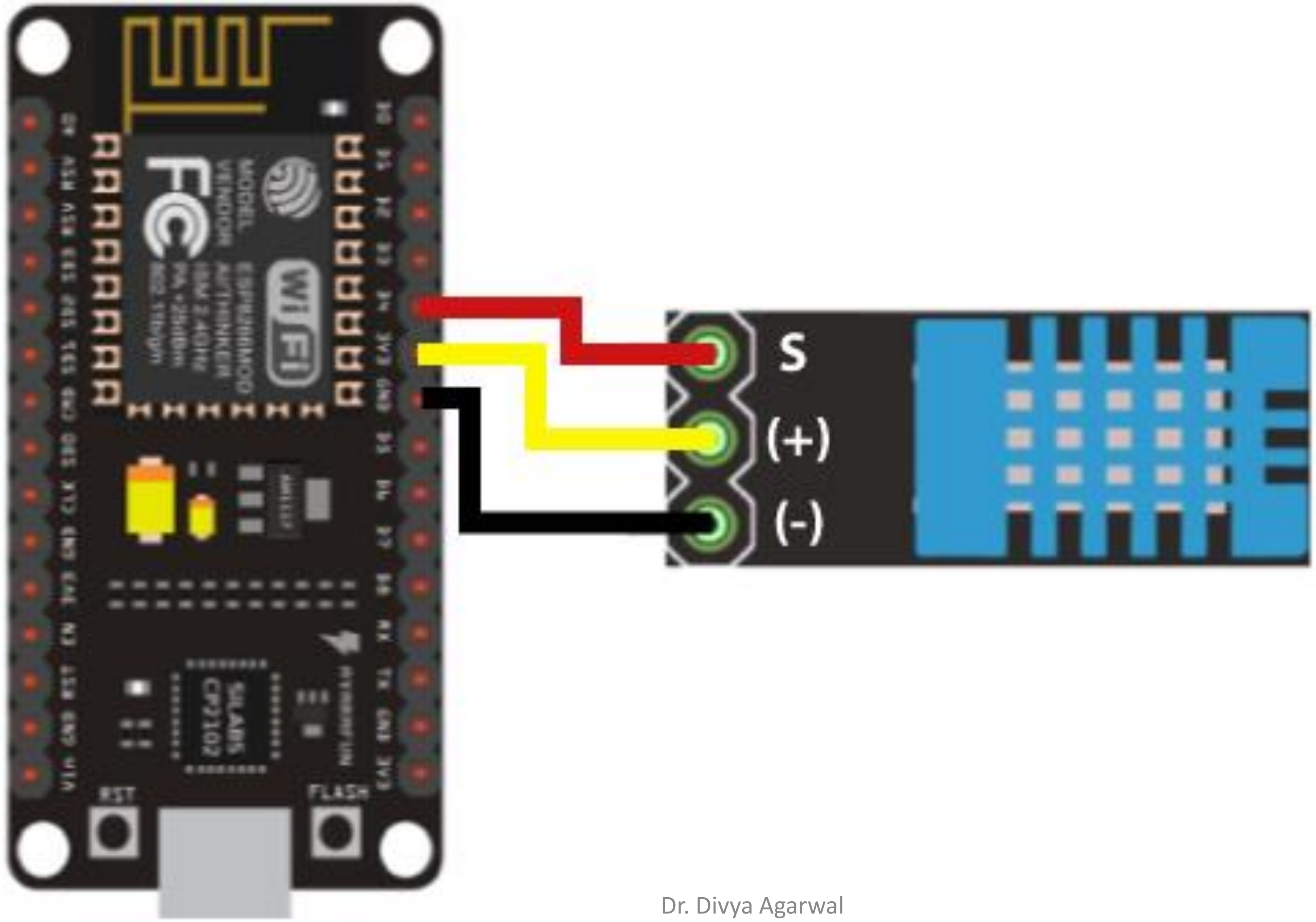
Interface DHT11 (Humidity Sensor) Using NodeMCU





Dr. Divya Agarwal

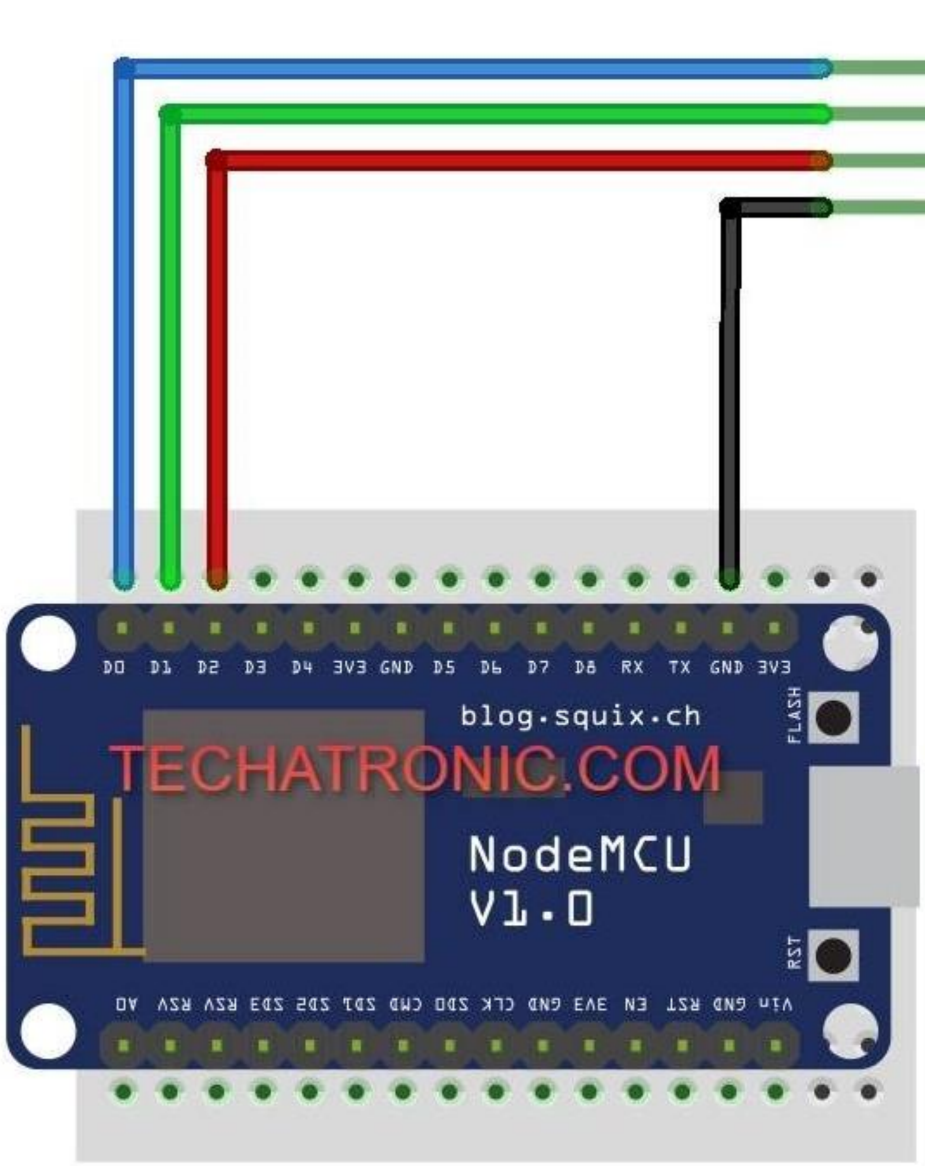
Connection between Node MCU and DHT11



Node MCU and DHT sensor Code

```
#include "DHT.h"
 DHT dht2(2,DHT11);
 void setup()
 {
  Serial.begin(9600);
 }
 void loop()
 {
   Serial.println("Temperature in C:");
   Serial.println((dht2.readTemperature( )));
   Serial.println("Humidity in C:");
   Serial.println((dht2.readHumidity()));
   delay(1000);
 }
```

Connection between Node MCU and RGB Led



- GND pin of module – GND pin of nodemcu.
- R pin (red light) of module – digital-2 pin of nodemcu.
- G pin (green light) of module – digital-1 pin of nodemcu.
- B pin (blue color) of module – digital-0 pin of nodemcu.

Connection between Node MCU and RGB Led

- GND pin of module – GND pin of nodemcu.
- R pin (red light) of module – digital-2 pin of nodemcu.
- G pin (green light) of module – digital-1 pin of nodemcu.
- B pin (blue color) of module – digital-0 pin of nodemcu.

Node MCU and RGB LED Module Code

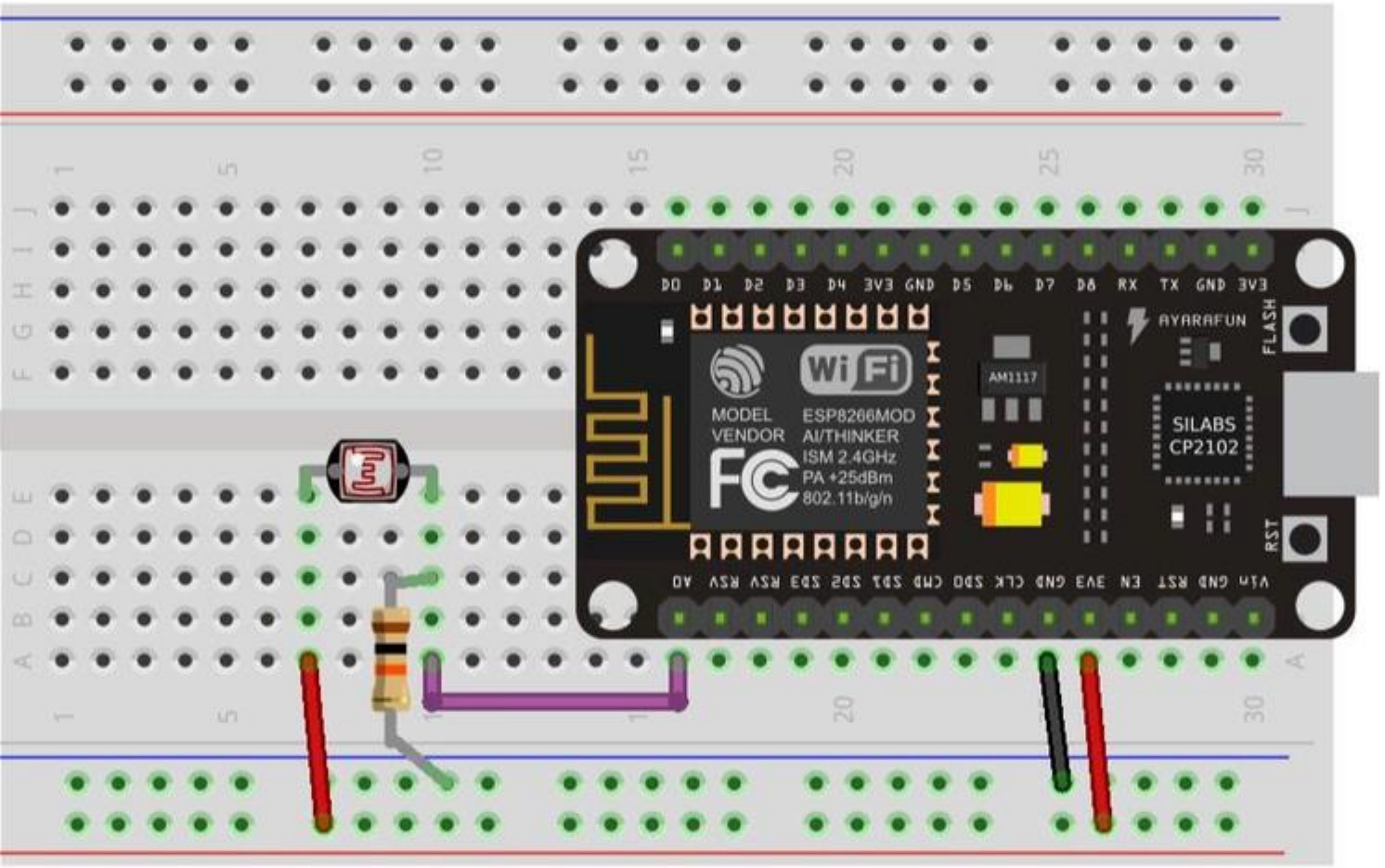
```
void setup()
 {
  pinMode(16,HIGH);  // Blue - D0 Pin
  pinMode(5,HIGH);  // Green - D1 Pin
  pinMode(4,HIGH);  // Red - D2 Pin
 }
 void loop()
 {
    // BLUE LED ON
   digitalWrite(16,HIGH);
   digitalWrite(5,LOW);
   digitalWrite(4,LOW);
   delay(1000);
 // GREEN LED ON
   digitalWrite(16,LOW);
   digitalWrite(5,HIGH);
   digitalWrite(4,LOW);
   delay(1000);
   // RED LED ON
   digitalWrite(16,LOW);
   digitalWrite(5,LOW);
   digitalWrite(4,HIGH);
   delay(1000);
 }
```

Node MCU and RGB LED Module Code

```
const int RED = 5;
const int GREEN = 4;
const int BLUE = 0;

void setup() {
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
}
```

```
void loop() {
  analogWrite(RED, 50);
  analogWrite(GREEN, 50);
  analogWrite(BLUE, 200);
  delay (1000);
  analogWrite(RED, 100);
  analogWrite(GREEN, 100);
  analogWrite(BLUE, 100);
  delay (1000);
}
```

Connection between Node MCU and LDR Module


D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
AYARAFUN
FLASH
AM1117
MODEL ESP8266MOD
VENDOR AI/THINKER
ISM 2.4GHz
PA +25dBm
802.11b/g/n
SILABS CP2102
RST
A0 RSV RSV SD3 SD2 SD1 CMD SDO CLK GND 3V3 EN RST GND Vin

GND pin of module – GND pin of nodemcu.

R pin (red light) of module – digital-2 pin of nodemcu.

G pin (green light) of module – digital-1 pin of nodemcu.

B pin (blue color) of module – digital-0 pin of nodemcu.

Node MCU and LDR Module Code

```
void setup()
{
        Serial.begin(9600);   // initialize serial communication at 9600 BPS
}

void loop()
{

        int sensorValue = analogRead(A0);   // read the input on analog pin 0

        float voltage = sensorValue * (5.0 / 1023.0);   // Convert the analog reading (which
goes from 0 - 1023) to a voltage (0 - 5V)

        Serial.println(voltage);   // print out the value you read
}
```

Additional Questions

- Evaluate the efficiency and performance of different communication protocols in terms of data transfer speed and compatibility with sensors/actuators.
- Compare and evaluate the benefits of using software libraries versus writing custom code for interacting with complex hardware components in Arduino sketches.
- Evaluate the effectiveness of an IoT application programming solution in addressing real-life problems, considering factors like usability, scalability, and reliability.

Syllabus Contents

- **UNIT- II**

❑**Communication protocols and Arduino Programming:** Understand various network protocols used in IoT, Understand various communication protocols (SPI, I2C, UART). Design and develop Arduino code needed to communicate the microcontroller with sensors and actuators, build circuits using IoT supported Hardware platforms such as Arduino, ESP8266 etc., Use of software libraries with an Arduino sketch that allows a programmer to use complicated hardware without dealing with complexity, Learning IoT application programming and build solutions for real life problems and test them in Arduino and Node MCU environments. Understand various wireless Technologies for IoT and its range, frequency and applications.

Unit 2 – Completed

THANKS

By: Dr. Divya Agarwal