

IoT Applications (315341)

Model Question Paper — Set B · Model Answers

Q.1

a) Characteristics of IoT.

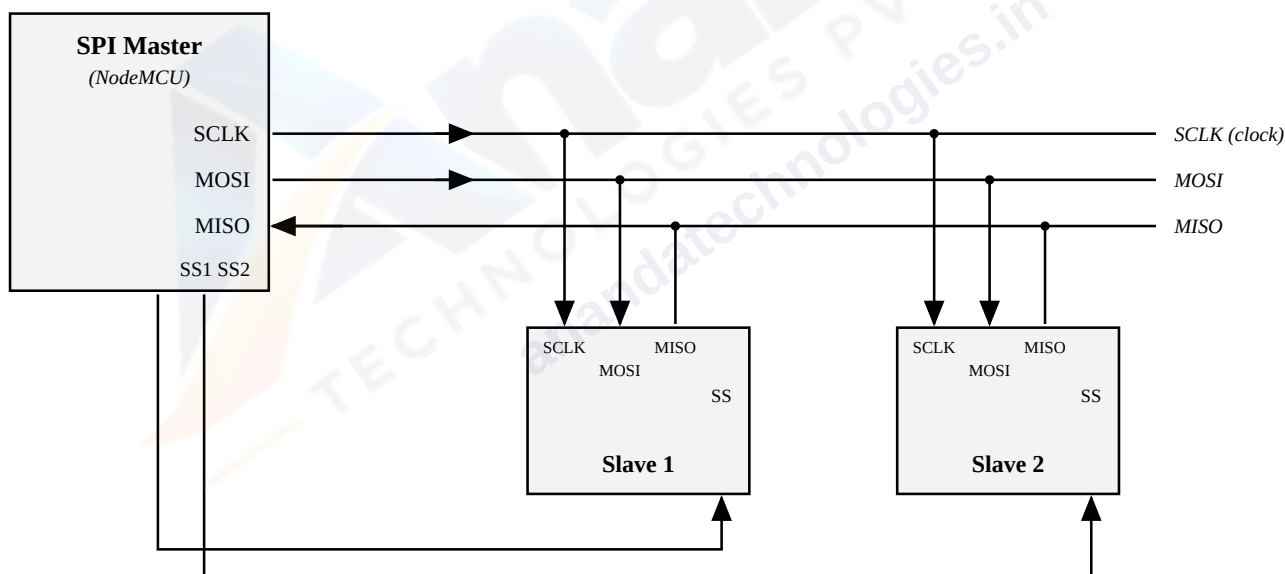
2 M

1. **Dynamic and self-adapting** — devices adapt to changing context (e.g. camera switches to night mode).
2. **Self-configuring** — devices configure themselves, fetch updates and join the network with little manual setup.
3. **Interoperable communication protocols** — devices talk to each other and to the cloud using standard protocols.
4. **Unique identity** — every device has a unique ID / IP address and can be queried and controlled remotely.
5. **Integrated into information network** — data from devices is combined with other systems for analysis; also scalability, connectivity, intelligence, security.

(Any four, ½ M each)

b) Block diagram of SPI interface.

2 M



SCLK, MOSI and MISO are shared; each slave has its own SS (slave select, active LOW)

Fig.: SPI interface – one master with two slaves

SCLK – clock from master, MOSI – master out slave in, MISO – master in slave out, SS – slave select (active LOW). On NodeMCU (HSPI): SCLK = D5, MISO = D6, MOSI = D7, SS = D8.

(Diagram 2 M)

c) Arduino functions.

2 M

Function	Use
<code>pinMode(pin, mode)</code>	Configures a pin as INPUT, OUTPUT or INPUT_PULLUP, e.g. <code>pinMode(D1, OUTPUT);</code>
<code>digitalWrite(pin, value)</code>	Writes HIGH (3.3 V) or LOW (0 V) on a digital output pin, e.g. <code>digitalWrite(D1, HIGH);</code>
<code>analogRead(pin)</code>	Reads the analog input (A0) and returns 0–1023 (10-bit ADC).
<code>delay(ms)</code>	Pauses the program for the given milliseconds, e.g. <code>delay(1000);</code> = 1 s.

(½ M each)

d) Applications of PIR sensor.

2 M

- Automatic lighting — lights in corridors, staircases, washrooms turn ON when a person enters (energy saving).
- Intruder / burglar alarm and security systems.
- Automatic door opening, hand-dryers; occupancy-based fan/AC control in smart homes and offices.
- Smart street lights (bright only when people move) and IoT surveillance (start camera on motion).

(Any two, 1 M each)

e) Pins of IR obstacle sensor module.

2 M

Pin	Function
VCC	Supply, 3.3 V – 5 V
GND	Ground
OUT	Digital output — LOW when an obstacle is detected, HIGH when no obstacle; connect to a NodeMCU digital pin

The module also has a potentiometer to adjust the detection distance (about 2–30 cm), a power LED and an obstacle-indicator LED.

(Pins 1½ M, note ½ M)

f) MQTT broker and topic.

2 M

MQTT broker: the central server in MQTT that receives every message published by clients and forwards it to all clients that have subscribed to that topic. It also handles client connections, authentication, QoS and retained messages. Examples: Mosquitto, HiveMQ, EMQX.

MQTT topic: a UTF-8 string used by the broker to route messages. It has levels separated by “/”, e.g. `farm/zone1/soil`. Clients publish to and subscribe to topics.

(1 M each)

g) Block diagram of EV battery management using IoT.

2 M

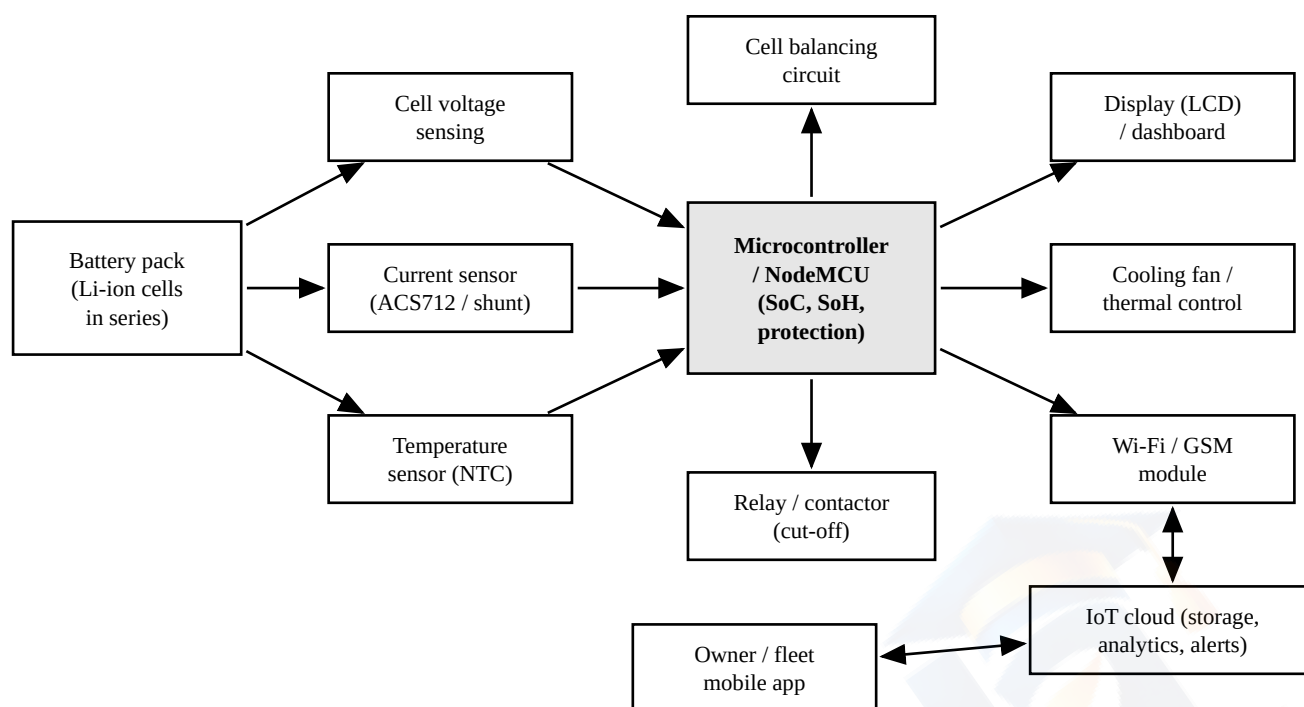


Fig.: Block diagram of EV battery management system (BMS) using IoT

(Diagram 2 M)

Q.2

a) Four-stage IoT architecture.

4 M

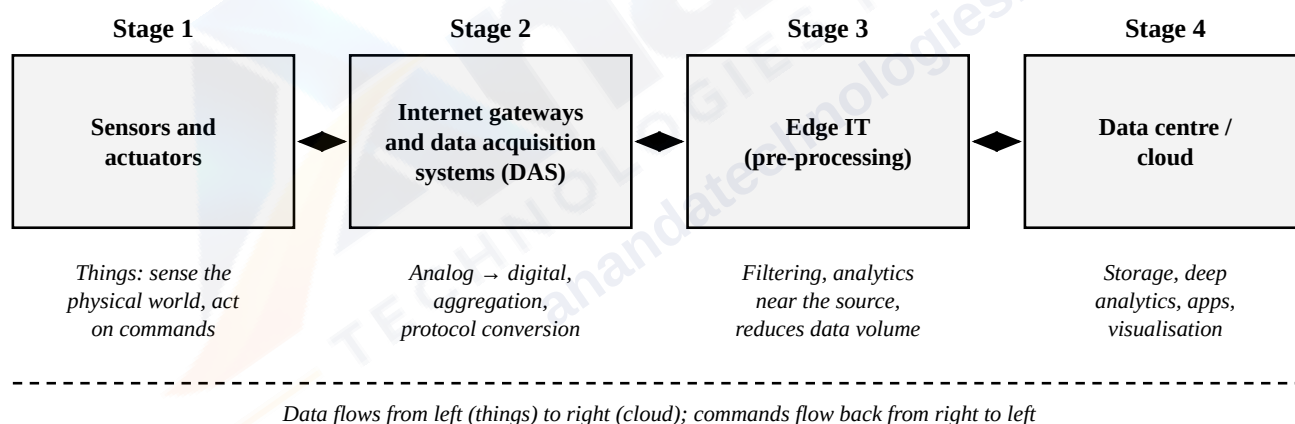


Fig.: Four-stage IoT architecture

- Stage 1 – Sensors and actuators:** the “things” that sense the physical world (temperature, pressure, motion) and act on commands (valves, motors). They produce analog/digital data.
- Stage 2 – Internet gateways and data acquisition systems (DAS):** collect data from many sensors, convert analog to digital, aggregate it and convert protocols; send it over Wi-Fi/LAN/cellular to the next stage.
- Stage 3 – Edge IT (pre-processing):** edge/fog computers close to the source filter, compress and analyse data in real time, so that only useful data is sent to the cloud; reduces bandwidth and delay.

4. **Stage 4 – Data centre / cloud:** long-term storage, deep analytics, machine learning, visualisation and business applications; results and commands flow back to the devices.

(Diagram 2 M, explanation 2 M)

b) Cloud computing and big data analytics in IoT.

4 M

Cloud computing: delivery of computing services — servers, storage, databases, software and analytics — over the Internet on demand and pay-per-use. Service models: **IaaS** (virtual machines, storage), **PaaS** (platform to build apps, e.g. AWS IoT Core, Azure IoT Hub), **SaaS** (ready applications).

- Stores the huge data produced by IoT devices and gives scalable processing power that small devices do not have.
- Provides device management, dashboards and remote access from anywhere; e.g. ThingSpeak, AWS IoT.

Big data analytics: collecting, storing and analysing very large and varied data sets to extract useful information. IoT data has the features of big data — **Volume** (huge), **Velocity** (high rate, streaming), **Variety** (numbers, images, text) and **Veracity**.

- Analytics finds patterns and predictions: e.g. predicting machine failure from vibration data, traffic forecasting, crop yield prediction.
- Tools: Hadoop, Spark, NoSQL databases, machine learning.

(Cloud computing 2 M, big data analytics 2 M)

c) GPIO pins of NodeMCU.

4 M

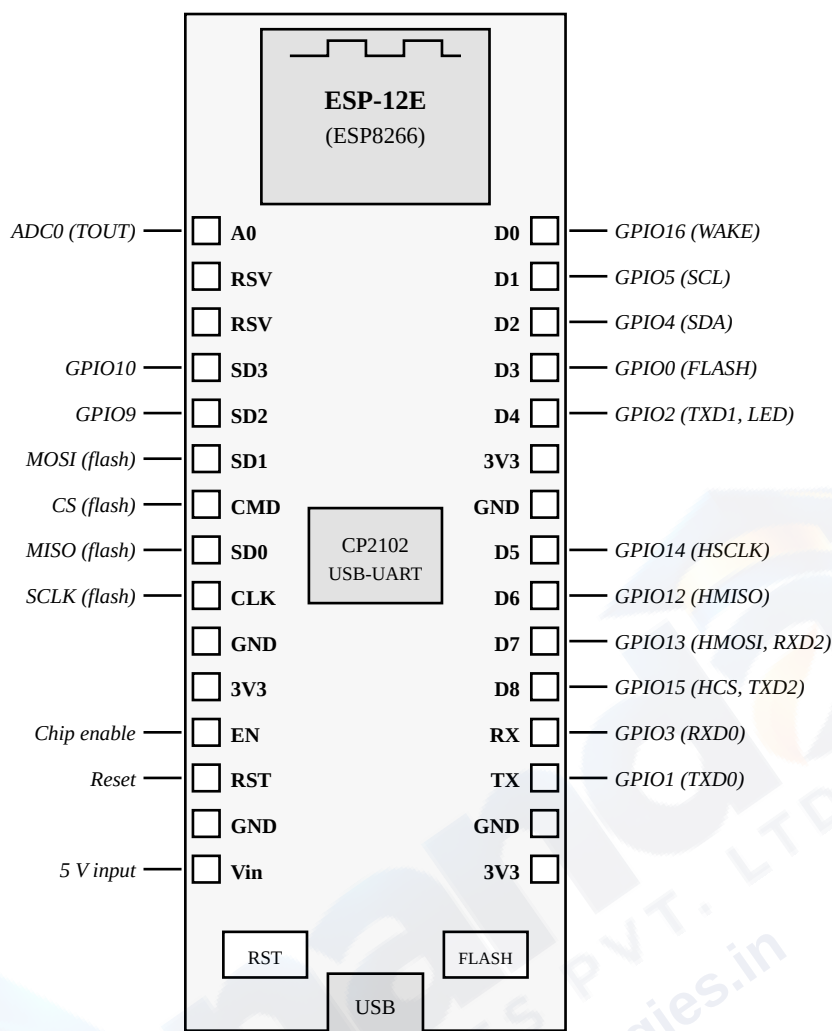


Fig.: Pin diagram of NodeMCU ESP8266 (ESP-12E, v1.0)

Purpose: GPIO (General Purpose Input/Output) pins can be programmed as digital **input** (read switches and digital sensors such as PIR, IR) or digital **output** (drive LEDs, relays, buzzers). NodeMCU brings out D0–D8 (plus RX, TX, SD2, SD3) at 3.3 V logic, max ≈ 12 mA per pin. They also support **PWM** (analogWrite), interrupts, and act as I2C/SPI/UART lines.

Pin	GPIO	Special function
D0	GPIO16	Wake-up from deep sleep (connect to RST); no PWM / interrupt
D1, D2	GPIO5, GPIO4	I2C SCL and SDA (default)
D3	GPIO0	FLASH button; must be HIGH at boot (LOW = programming mode)
D4	GPIO2	On-board LED (active LOW), UART1 TX
D5–D8	GPIO14, 12, 13, 15	HSPI: SCLK, MISO, MOSI, CS
RX, TX	GPIO3, GPIO1	UART0 – programming and Serial Monitor

(Purpose with diagram 2 M, four special functions 2 M)

d) Stepper motor – working, phase energization and calculation.

4 M

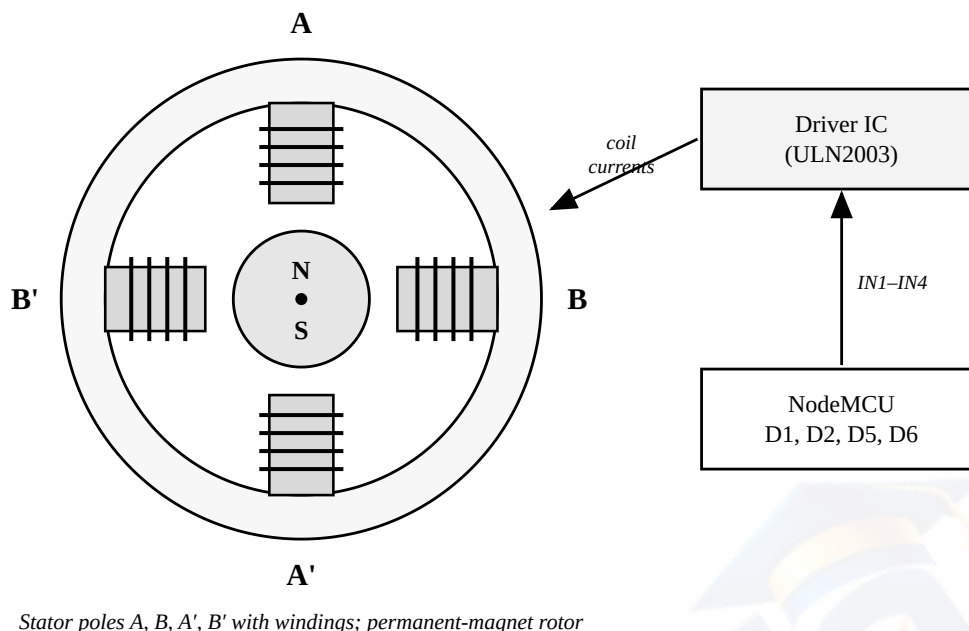


Fig.: Stepper motor (4-phase unipolar) – construction and driver

A **stepper motor** rotates in fixed angular steps. The stator windings (phases A, B, A', B') are energised one after another in a sequence by a driver (ULN2003); the permanent-magnet rotor aligns itself with the energised pole, so each change of the sequence moves the rotor by one **step angle**. Reversing the sequence reverses the direction; the step rate sets the speed.

(a) Wave drive (one phase ON) (b) Full step (two phases ON, more torque)

Step	A	B	A'	B'
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Step	A	B	A'	B'
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Calculation: Steps per revolution = $360^\circ / \text{step angle} = 360 / 1.8 = \mathbf{200 \text{ steps}}$. Steps for $90^\circ = 90 / 1.8 = \mathbf{50 \text{ steps}}$. (At 100 steps/s the speed would be $100/200 \times 60 = 30 \text{ rpm}$.)

(Working 1 M, sequence 2 M, calculation 1 M)

Q.3

a) MQTT vs CoAP.

4 M

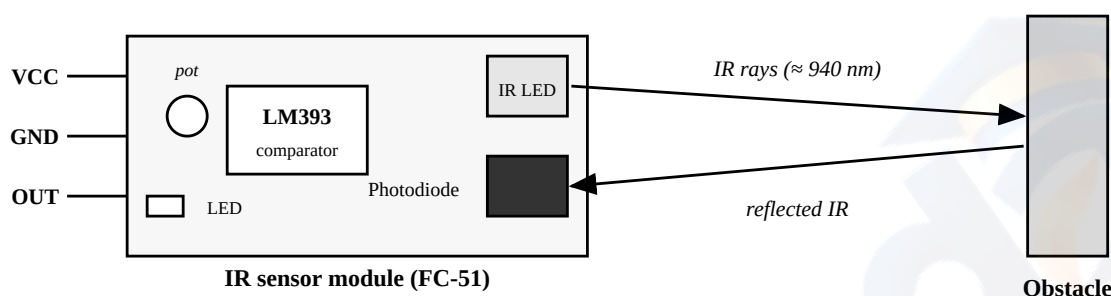
Point	MQTT	CoAP
i) Transport protocol	TCP (port 1883, 8883 with TLS)	UDP (port 5683, 5684 with DTLS)
ii) Communication model	Publish–subscribe through a central broker (many-to-many)	Request–response (client–server, REST: GET, POST, PUT, DELETE); also “observe”

iii) Message / header size	Fixed header of only 2 bytes; small packets	Fixed header of 4 bytes; compact binary format, designed for very constrained nodes (6LoWPAN)
iv) Reliability	TCP + three QoS levels (0, 1, 2); retained messages, last-will	Confirmable (CON) messages with ACK and retransmission; NON for unreliable data
Typical use	Telemetry to cloud, mobile apps, dashboards	Low-power sensor networks, M2M, direct device control

(Four points, 1 M each)

b) IR sensor – construction and working.

4 M



Obstacle present → reflected IR reaches photodiode → its resistance falls → comparator output LOW (indicator LED ON)

Fig.: IR obstacle sensor – construction and working (reflection type)

Construction: an IR obstacle sensor module (FC-51) has (1) an **IR LED** transmitter (≈ 940 nm), (2) an **IR photodiode** receiver placed beside it and shielded from direct light, (3) an **LM393 comparator** with a **potentiometer** to set the reference (distance), (4) indicator LEDs and (5) pins VCC, GND, OUT.

Working: the IR LED continuously emits invisible IR rays. With no obstacle the rays do not return; the photodiode receives little IR, its resistance is high and the comparator output is HIGH. When an object comes in front (2–30 cm), the IR rays are **reflected** to the photodiode, its resistance falls, the voltage at the comparator input crosses the reference and **OUT becomes LOW**; the obstacle LED glows. NodeMCU reads OUT with digitalRead(). Dark (black) surfaces reflect less IR, so the range is shorter.

(Construction with sketch 2 M, working 2 M)

c) Physical design of IoT.

4 M

The physical design of an IoT system consists of the **things (IoT devices)** and the **IoT protocols**.

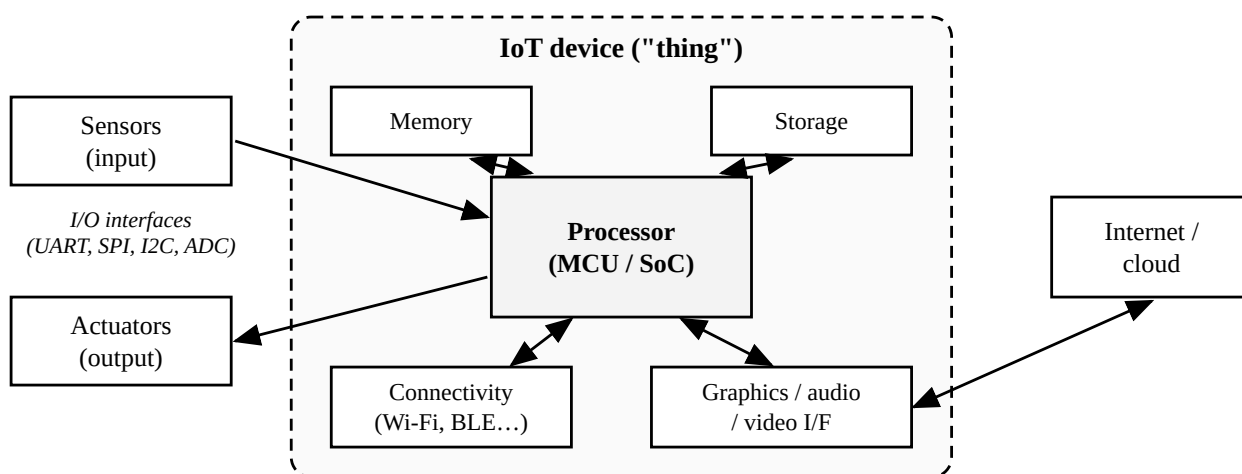


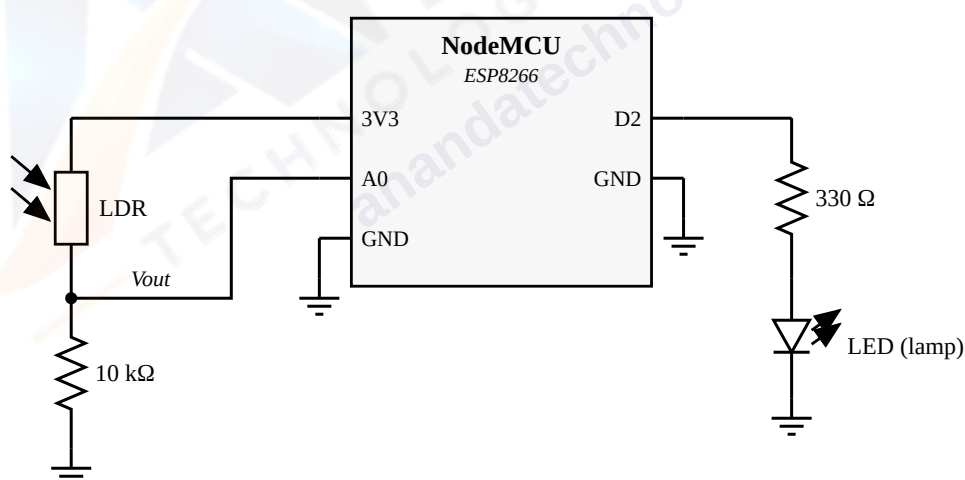
Fig.: Generic block diagram of an IoT device (physical design)

- A **thing** has a unique identity and can sense, actuate and monitor remotely. It contains a **processor** (MCU/SoC), **memory and storage**, **I/O interfaces** for sensors and actuators (UART, SPI, I2C, ADC, CAN), **connectivity** interfaces (Wi-Fi, Ethernet, Bluetooth, ZigBee) and optional audio/video/graphics interfaces.
- **IoT protocols** — link layer (802.3 Ethernet, 802.11 Wi-Fi, 802.15.4, 2G/3G/4G), network layer (IPv4, IPv6, 6LoWPAN), transport layer (TCP, UDP) and application layer (HTTP, CoAP, MQTT, WebSocket, XMPP, AMQP, DDS).
- Example of a thing: NodeMCU with a DHT11 sensor and relay — senses temperature, switches a fan and sends data over Wi-Fi.

(Diagram 2 M, explanation 2 M)

d) Smart light-intensity control (PWM) using NodeMCU and LDR.

4 M



For a 230 V street lamp, D2 drives a relay module instead of the LED

Fig.: LDR divider on A0; LED (or LED driver) on PWM pin

The LDR divider gives a high ADC value in bright light and a low value in darkness. The program maps the reading *inversely* to a PWM duty cycle (analogWrite) — the darker the room, the brighter the LED — keeping the room illumination nearly constant and saving energy.

```
// Smart light intensity control: darker room -> brighter LED (PWM on D2)
const int LED = D2;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  analogWriteRange(1023);          // PWM duty 0 - 1023
}

void loop() {
  int light = analogRead(A0);      // 0 (dark) - 1023 (bright)
  int duty = map(light, 0, 1023, 1023, 0); // inverse relation
  analogWrite(LED, duty);
  Serial.print("Light = "); Serial.print(light);
  Serial.print(" PWM duty = "); Serial.print(duty);
  Serial.print(" ("); Serial.print(duty * 100 / 1023);
  Serial.println("% brightness)");
  delay(500);
}
```

Output — Serial Monitor output (program logic run on a PC with test input values; on the board the values depend on the real sensor):

```
Light = 1000 PWM duty = 23 (2% brightness)
Light = 700 PWM duty = 323 (31% brightness)
Light = 400 PWM duty = 623 (60% brightness)
Light = 150 PWM duty = 873 (85% brightness)
Light = 20 PWM duty = 1003 (98% brightness)
```

(Diagram 1 M, explanation 1 M, program 2 M)

Q.4

a) MQ-4 module pins and interfacing.

4 M

Pin	Function
VCC	+5 V supply for the heater and circuit (heater draws ≈ 150 mA)
GND	Ground
DO (digital out)	From LM393 comparator: goes LOW when gas concentration is above the threshold set by the pot; to a digital pin (D1)
AO (analog out)	Analog voltage that rises with methane concentration; to A0 through a divider (AO can reach 5 V, A0 max 3.3 V)

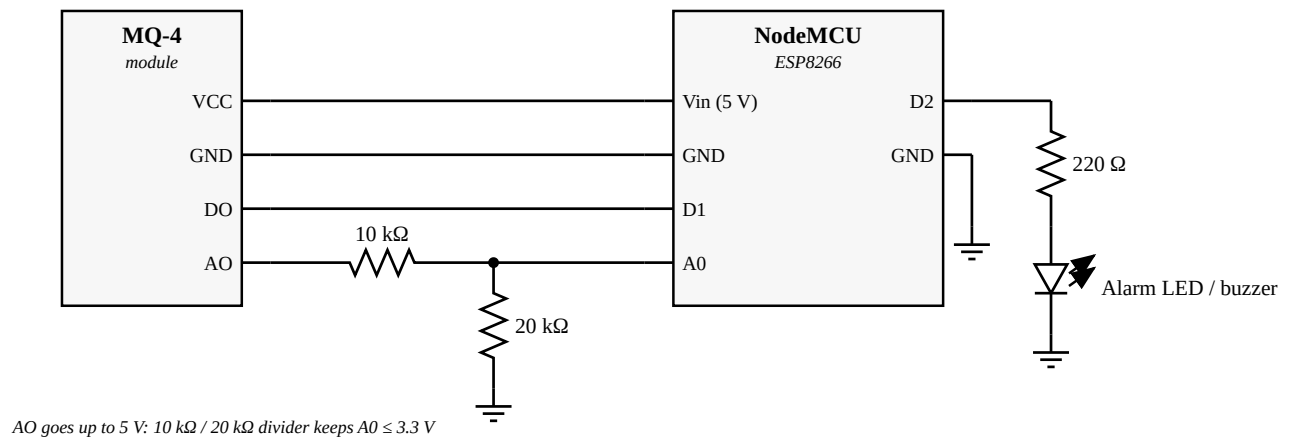


Fig.: Interfacing MQ-4 gas sensor module with NodeMCU

(Pins 2 M, interfacing diagram 2 M)

b) Wi-Fi features and NodeMCU station-mode program.

4 M

- Standard IEEE **802.11** (b/g/n/ac/ax) wireless LAN; NodeMCU supports 802.11 b/g/n.
- Works in the **2.4 GHz** (and 5 GHz) ISM bands; range about 30–50 m indoors, ~100 m outdoors.
- High data rate — 11 Mbps (b), 54 Mbps (g), up to 600 Mbps (n) — and direct **IP / Internet** connectivity.
- Security: WPA2 / WPA3 encryption; access by SSID and password; CSMA/CA medium access; infrastructure (AP) and ad-hoc modes.



Fig.: Wi-Fi modes of NodeMCU – (a) station (STA), (b) access point (AP)

```
// Connect NodeMCU to a Wi-Fi router in station (STA) mode
#include <ESP8266WiFi.h>

const char* ssid = "MyHomeWiFi";
const char* password = "12345678";

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);           // station mode
  WiFi.begin(ssid, password);
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP());
}
```

```
Serial.print("Signal strength (RSSI): ");
Serial.println(WiFi.RSSI());
}

void loop() { }
```

Output — Typical Serial Monitor output (depends on your network; not captured from hardware):

```
Connecting.....
Connected, IP address: 192.168.1.105
Signal strength (RSSI): -58
```

(Four features 2 M, program 2 M)

c) Smart energy meter using IoT.

4 M

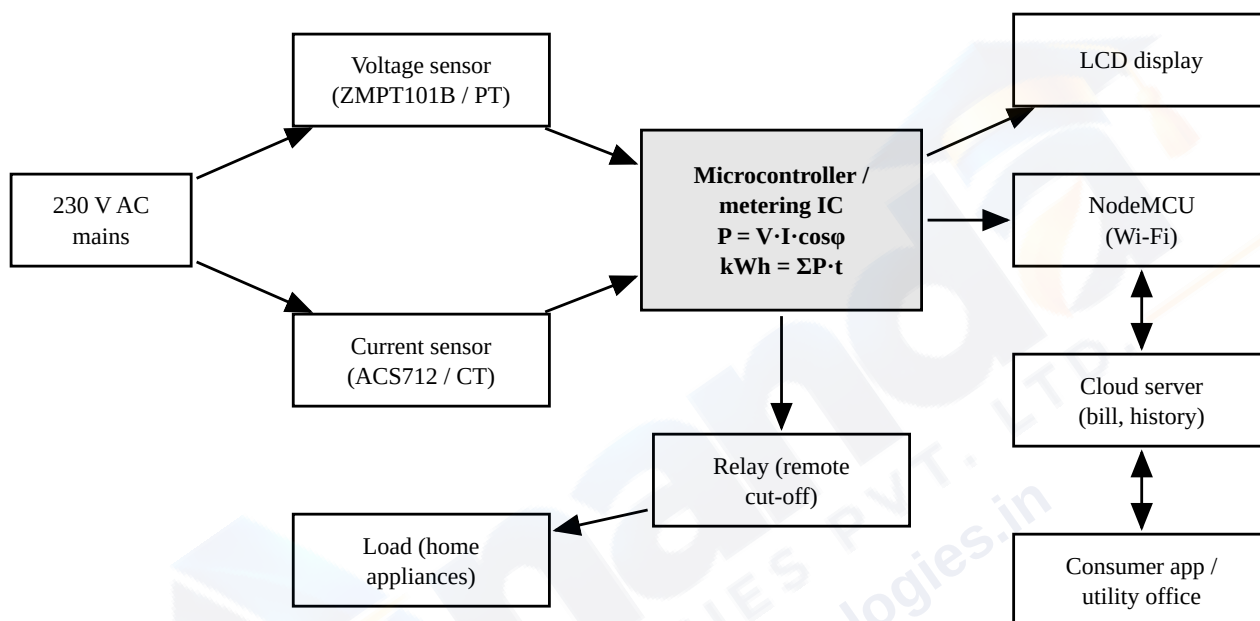


Fig.: Block diagram of IoT based smart energy meter

1. A **voltage sensor** (ZMPT101B / potential transformer) and a **current sensor** (ACS712 / current transformer) measure the mains voltage and the load current.
2. The **microcontroller / metering IC** samples them and calculates power $P = V \cdot I \cdot \cos \phi$ and energy $(kWh) = \sum P \times t$.
3. Readings are shown on an **LCD** and sent by **NodeMCU (Wi-Fi)** to a **cloud server** at regular intervals.
4. The consumer sees real-time usage and bill on a **mobile app**; the utility gets automatic meter reading — no manual visit. A **relay** allows remote disconnection, load limiting and prepaid metering; alerts are sent for over-use or tampering.

(Block diagram 2 M, working 2 M)

d) UART frame for 'K'.

4 M

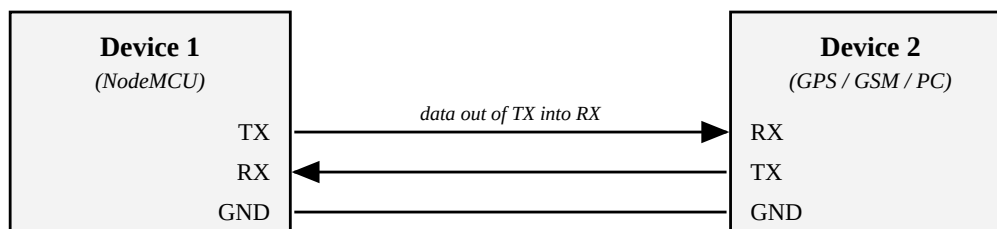
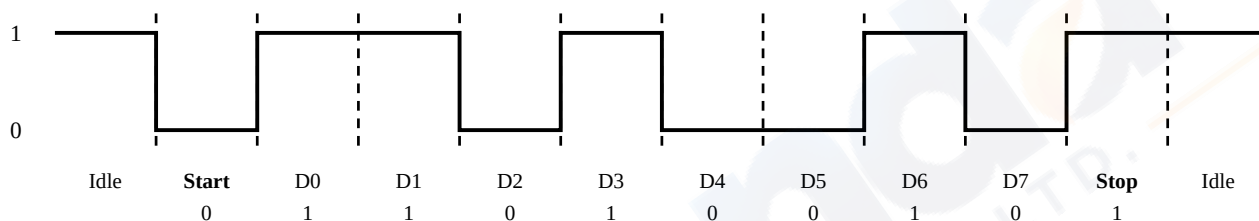


Fig.: UART connection between two devices (TX → RX cross-connection)

UART (Universal Asynchronous Receiver Transmitter) is an **asynchronous**, point-to-point, full-duplex serial communication using two lines — TX and RX (plus common GND). No clock is sent; both sides must use the same **baud rate** (9600, 115200 bps...). Frame: idle line HIGH → 1 **start bit** (LOW) → 5–9 **data bits** (LSB first) → optional **parity bit** → 1 or 2 **stop bits** (HIGH).
Character 'K': ASCII = 75 = 0x4B = 01001011₂. Sent LSB first: 1 1 0 1 0 0 1 0.



Character 'K' – data bits sent LSB first (8N1: 1 start, 8 data, no parity, 1 stop)

Fig.: UART frame of 'K' (8N1)

Time: bit time = $1 / 9600 = 104.17 \mu\text{s}$; one frame = 10 bits → $10 \times 104.17 \mu\text{s} = \mathbf{1.042 \text{ ms}}$ (max 960 characters/s).

(Explanation 1 M, frame diagram 2 M, time 1 M)

e) LoRa vs NB-IoT.

4 M

Point	LoRa	NB-IoT
Spectrum	Unlicensed ISM sub-GHz band (865–867 MHz in India)	Licensed LTE cellular spectrum (operator)
Standard / owner	LoRa (Semtech) + LoRaWAN (LoRa Alliance); private networks possible	3GPP Release 13; run by telecom operators (needs SIM)
Modulation / bandwidth	Chirp spread spectrum, 125 kHz channels	OFDMA / SC-FDMA, 180 kHz carrier
Data rate	0.3 – 50 kbps	Higher — up to about 250 kbps (tens of kbps typical)
Range / coverage	2–5 km urban, 15 km+ rural; own gateways needed	Uses existing towers; very good deep-indoor coverage
Cost	No subscription; low module cost	Subscription fee; module costlier

(Any four points, 1 M each)

Q.5

a) Arduino character functions.

6 M

These functions (from the Arduino core, based on C `ctype.h`) test a single character and return **true (1)** or **false (0)**:

Function	Returns true when the character is...	Example
<code>isAlpha(c)</code>	a letter A–Z or a–z (C: <code>isalpha</code>)	<code>isAlpha('B') → true</code>
<code>isDigit(c)</code>	a decimal digit 0–9 (C: <code>isdigit</code>)	<code>isDigit('7') → true</code>
<code>isAlphaNumeric(c)</code>	a letter or a digit (C: <code>isalnum</code>)	<code>isAlphaNumeric('?') → false</code>
<code>isHexadecimalDigit(c)</code>	0–9, A–F or a–f (C: <code>isxdigit</code>)	<code>isHexadecimalDigit('f') → true</code>
<code>isUpperCase(c)</code>	an upper-case letter A–Z (C: <code>isupper</code>)	<code>isUpperCase('a') → false</code>
<code>isSpace(c)</code>	white space: space, <code>\t</code> , <code>\n</code> , <code>\v</code> , <code>\f</code> , <code>\r</code> (C: <code>isspace</code>)	<code>isSpace(' ') → true</code>

```
// Character functions of the Arduino language (tested on NodeMCU)
const char text[] = "aB7 f?";

void setup() {
  Serial.begin(9600);
  Serial.println("ch | Alpha Digit AlNum Hex Upper Lower Space Punct");
  for (int i = 0; text[i] != '\0'; i++) {
    char c = text[i];
    Serial.print(""); Serial.print(c); Serial.print(" | ");
    Serial.print(isAlpha(c)); Serial.print(" ");
    Serial.print(isDigit(c)); Serial.print(" ");
    Serial.print(isAlphaNumeric(c)); Serial.print(" ");
    Serial.print(isHexadecimalDigit(c)); Serial.print(" ");
    Serial.print(isUpperCase(c)); Serial.print(" ");
    Serial.print(isLowerCase(c)); Serial.print(" ");
    Serial.print(isSpace(c)); Serial.print(" ");
    Serial.println(isPunct(c));
  }
  Serial.print("toUpperCase('f') = ");
  Serial.println((char)toUpperCase('f'));
}

void loop() { }
```

Output — Serial Monitor output (no sensor input, so the PC run gives exactly the board output):

```
ch | Alpha Digit AlNum Hex Upper Lower Space Punct
'a' | 1 0 1 1 0 1 0 0
'B' | 1 0 1 1 1 0 0 0
'7' | 0 1 1 1 0 0 0 0
' ' | 0 0 0 0 0 0 1 0
'f' | 1 0 1 1 0 1 0 0
'?' | 0 0 0 0 0 0 0 1
toUpperCase('f') = F
```

Also available: `isLowerCase()`, `isPunct()`, `isPrintable()`, `isControl()`, `toUpperCase()`, `toLowerCase()`.

(Six functions 3 M, program with output 3 M)

b) DHT11 and LM35 pin details with interfacing.

6 M

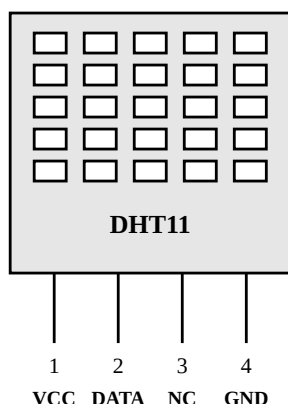


Fig.: DHT11 sensor pin diagram

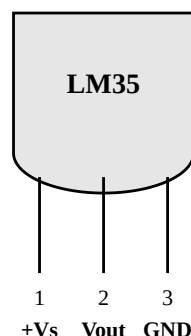


Fig.: LM35 (TO-92) pin diagram – flat face towards viewer

DHT11 pin	Function
1 VCC	3.3 – 5.5 V supply
2 DATA	Single-wire serial data (40-bit frame); 10 kΩ pull-up to VCC
3 NC	Not connected
4 GND	Ground

DHT11: humidity 20–90 % RH (± 5 %), temperature 0–50 °C (± 2 °C), 1 reading per second. (The 3-pin module has VCC, DATA, GND with the pull-up resistor on board.)

LM35 pin	Function
1 +Vs	Supply 4 – 30 V (use $V_{in} = 5$ V on NodeMCU)
2 Vout	Analog output, 10 mV/°C (0 V at 0 °C); to A0
3 GND	Ground

LM35: precision centigrade sensor, –55 to +150 °C, ± 0.5 °C accuracy at 25 °C, linear, no calibration needed.

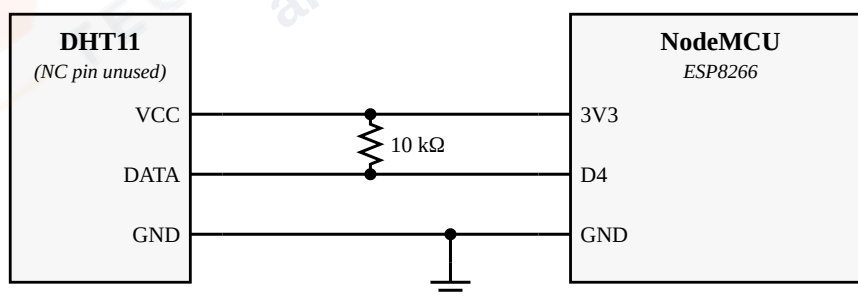


Fig.: Interfacing DHT11 with NodeMCU (DATA on D4 with 10 kΩ pull-up)

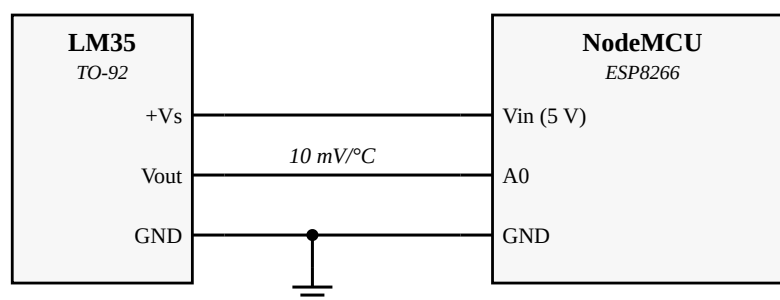


Fig.: Interfacing LM35 temperature sensor with NodeMCU

(DHT11 pins 2 M, LM35 pins 2 M, interfacing 2 M)

c) Industrial IoT applications and predictive maintenance.

6 M

IoT applications in industrial automation (any four):

- Predictive maintenance of machines
- Remote monitoring and control of plant (SCADA with IoT)
- Asset and inventory tracking (RFID, GPS)
- Energy monitoring of machines and smart grids
- Quality control with sensors / machine vision
- Worker safety — gas leakage and fire detection, wearables

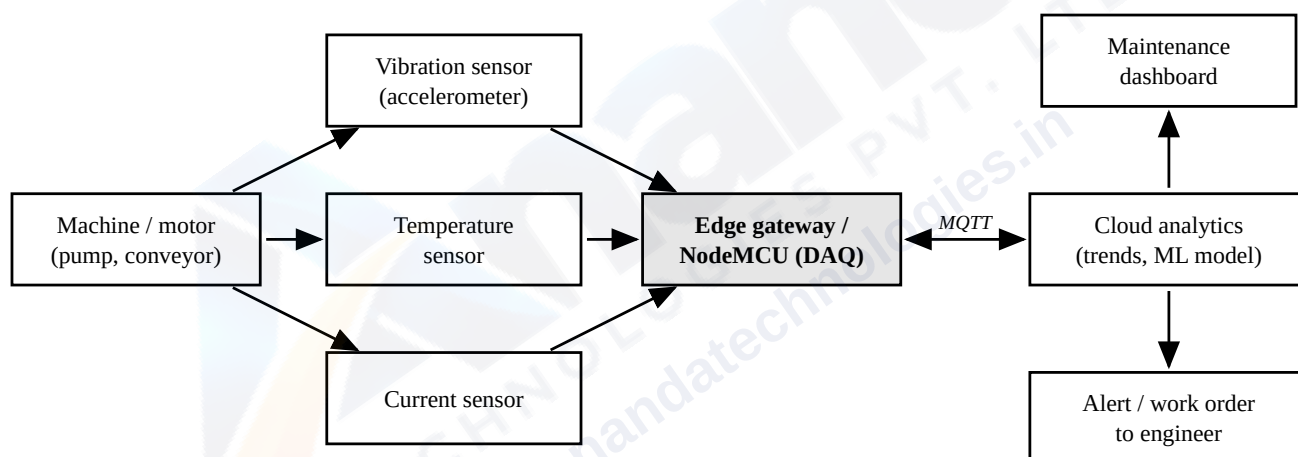


Fig.: Predictive maintenance of a machine using industrial IoT

Predictive maintenance: instead of repairing a machine after breakdown (reactive) or at fixed intervals (preventive), IoT predicts failure from the actual condition of the machine.

1. **Sensors** on the motor/pump continuously measure vibration (accelerometer), temperature and current.
2. An **edge gateway / NodeMCU** acquires the data, filters it and sends it to the cloud using MQTT.
3. **Cloud analytics** compares trends with normal behaviour and uses machine-learning models to detect abnormal vibration, overheating or overload.
4. When a fault is predicted, an **alert / work order** is sent to the maintenance engineer and shown on the dashboard, so the part is replaced before failure.

Benefits: less unplanned downtime, lower maintenance cost, longer machine life, better safety.

Q.6

a) Web-server based appliance control.

6 M

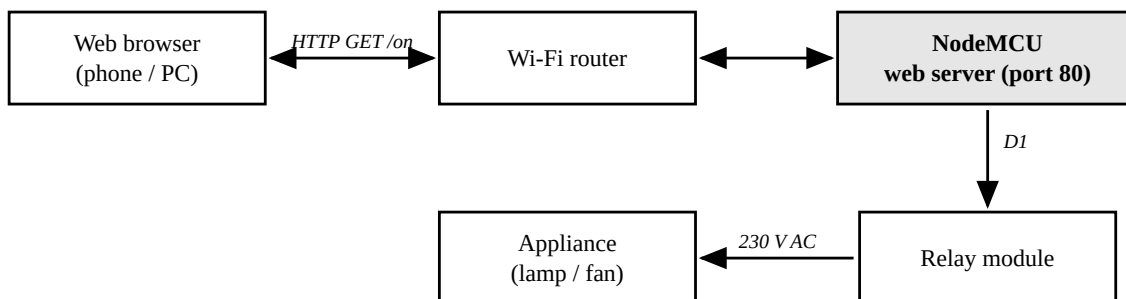


Fig.: NodeMCU as a web server controlling a relay

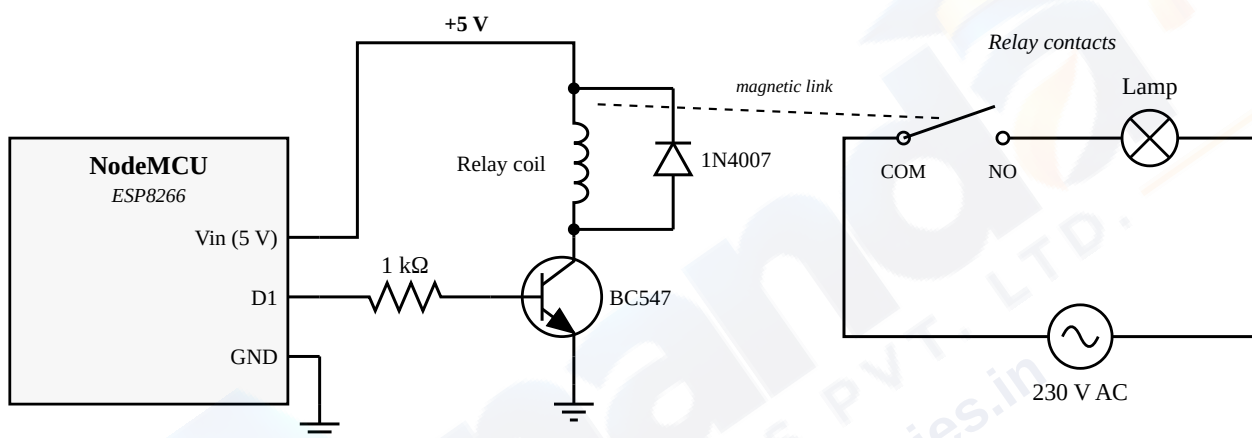


Fig.: Relay driver circuit – NodeMCU D1 switching a 230 V AC appliance

NodeMCU joins the home Wi-Fi and runs a web server on port 80. A browser on the same network opens the IP address; clicking the links sends GET /on or GET /off, and the handler functions switch the relay on D1.

```
// Home appliance ON/OFF from a web browser: NodeMCU web server, relay on D1
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

const char* ssid = "MyHomeWiFi";
const char* password = "12345678";
const int RELAY = D1;
ESP8266WebServer server(80);

void sendPage() {
  String s = "<h2>Home Appliance</h2><p>Status: ";
  s += digitalRead(RELAY) ? "ON" : "OFF";
  s += "</p><a href='/on'>TURN ON</a> | <a href='/off'>TURN OFF</a>";
  server.send(200, "text/html", s);
}

void handleOn() { digitalWrite(RELAY, HIGH); sendPage(); }
void handleOff() { digitalWrite(RELAY, LOW); sendPage(); }
```

```
void setup() {
  Serial.begin(115200);
  pinMode(RELAY, OUTPUT);
  digitalWrite(RELAY, LOW);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print("."); }
  Serial.print("\nOpen http://");
  Serial.println(WiFi.localIP());
  server.on("/", sendPage);
  server.on("/on", handleOn);
  server.on("/off", handleOff);
  server.begin();
}

void loop() {
  server.handleClient();           // serve browser requests
}
```

Output — Typical Serial Monitor output (depends on your network; not captured from hardware):

```
.....
Open http://192.168.1.107
(browser shows: Home Appliance  Status: OFF  TURN ON | TURN OFF)
```

(Block diagram 1 M, program 4 M, explanation 1 M)

b) IoT based greenhouse / smart agriculture system.

6 M

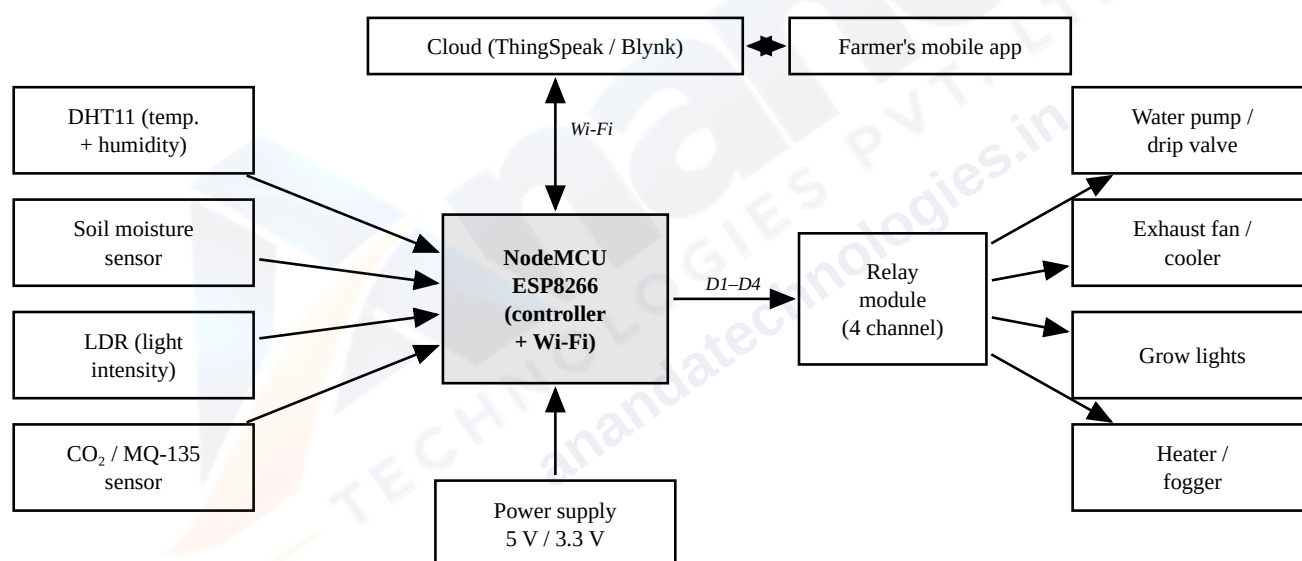


Fig.: Block diagram of IoT based greenhouse control system

NodeMCU reads all sensors every few seconds, compares them with set points, drives the actuators through a relay module and uploads the data to a cloud dashboard (ThingSpeak / Blynk), where the farmer can watch and override from a mobile app.

Sensor	Condition	Control action
Soil moisture sensor	Soil dry (moisture below set value, e.g. 40 %)	Water pump / drip valve ON until moisture is normal
DHT11 – temperature	Above set value (e.g. 32 °C)	Exhaust fan / cooler ON

DHT11 – temperature	Below set value (e.g. 15 °C)	Heater ON
DHT11 – humidity	Too low	Fogger / mister ON
LDR	Low light during daytime	Grow lights ON
CO ₂ / gas sensor	Poor air quality	Ventilation fan ON, alert to farmer

Advantages: saves water and energy, higher yield, less labour, remote monitoring and alerts.

(Block diagram 3 M, explanation with control actions 3 M)

c) HTTP-REST and HTTP vs MQTT.

6 M

REST (Representational State Transfer) is an architectural style for web services that uses HTTP. Each sensor or actuator is a **resource** with a URL, and standard HTTP methods act on it: GET (read), POST (create), PUT (update), DELETE (remove). Data is usually exchanged in JSON.

Features:

- Client–server architecture over TCP/IP (port 80 / 443).
- **Stateless** — every request contains all required information.
- Uniform interface using URLs and standard methods; human-readable text headers and status codes (200, 404...).
- Responses can be **cached**; layered system (proxies, gateways).

Advantages: simple and universal — works with every browser, web server and cloud; easy to debug; firewall friendly; secure with HTTPS; many libraries (ESP8266HTTPClient).

Point	HTTP	MQTT
Model	Request–response (client–server)	Publish–subscribe via broker
Header / overhead	Large text headers (hundreds of bytes)	Very small, 2-byte fixed header
Direction	Client must poll for new data	Broker pushes data to subscribers instantly
Connection	Usually new connection per request	Persistent TCP connection, keep-alive
Power use	Higher	Lower — suited to battery devices
QoS	None (relies on TCP)	QoS 0, 1, 2

(REST features and advantages 3 M, comparison 3 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in