

Industrial Internet of Things using ESP32

– Yogesh Mene



Overview



Master IoT Fundamentals & ESP32 Development: Gain a solid understanding of Industrial Internet of Things (IIoT) concepts, architecture, and protocols. Learn to program and configure the ESP32 microcontroller using Arduino IDE and Wokwi simulator for real-world industrial applications.



Build Practical IIoT Projects: Apply hands-on skills to design and implement IIoT solutions including GPIO control, sensor integration, LED and traffic light projects, and edge-to-cloud communication — bridging theory with Industry 4.0 practice.



DAY 1 AGENDA

IoT Fundamentals & History

Definition of IoT, evolution from M2M to IIoT, key milestones, and real-world industrial applications overview.

Architecture & Industry 4.0

IoT layered architecture, edge/fog/cloud computing, Industry 4.0 principles, and smart manufacturing integration.

Communication Protocols

MQTT, CoAP, HTTP, and RFID protocols — how devices communicate in industrial IoT ecosystems.

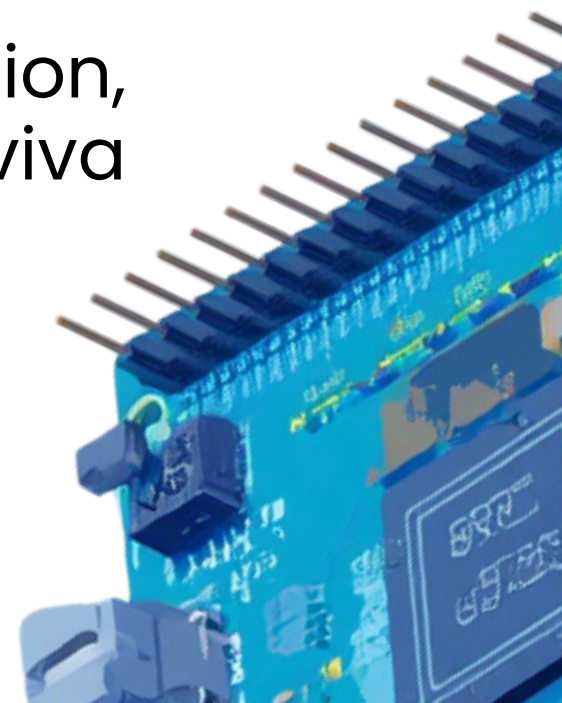


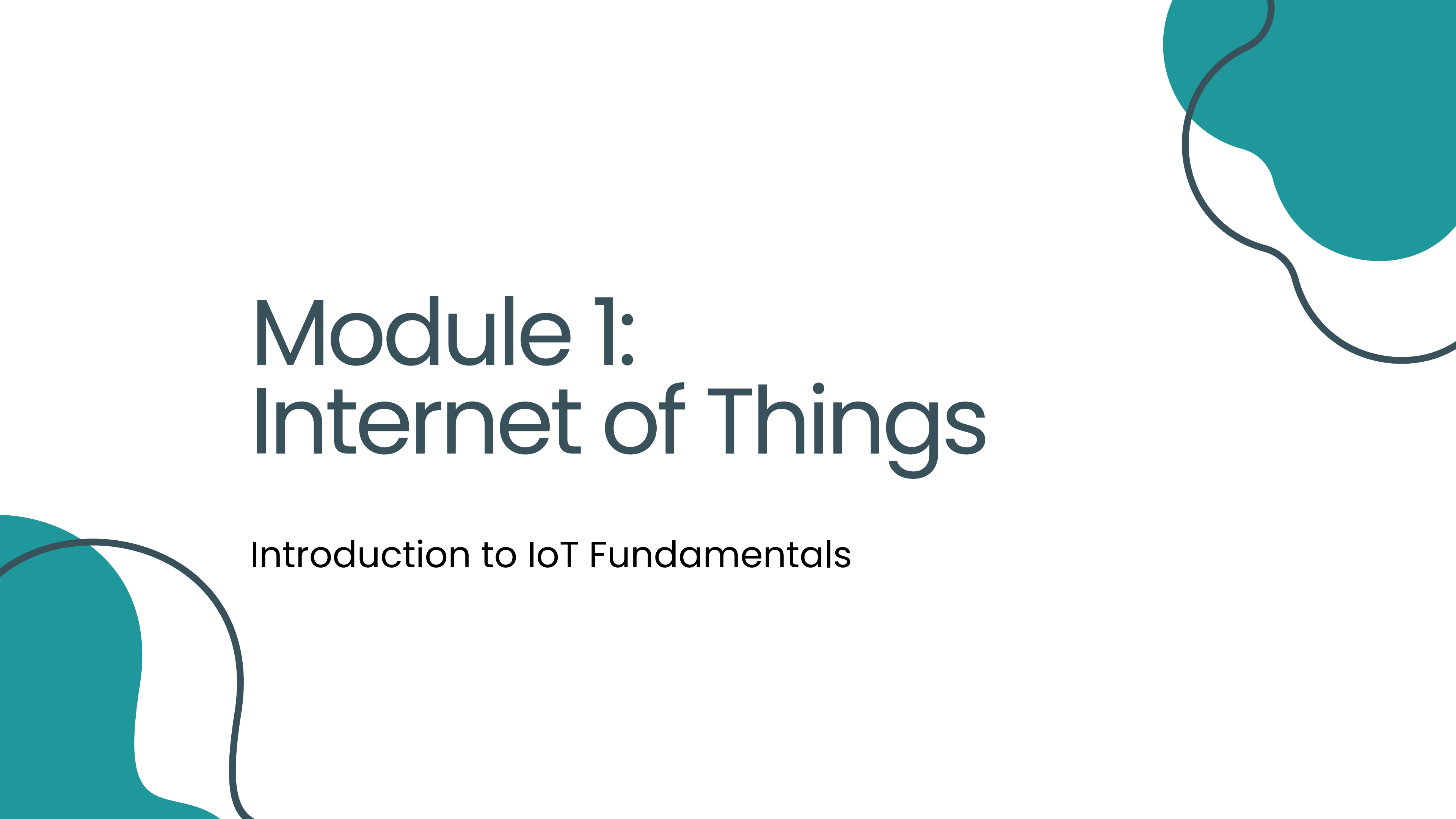
DAY 2 AGENDA

ESP32 Hardware & Setup — ESP32 architecture, pinout diagram, Arduino IDE installation, Wokwi simulator introduction, and board configuration for IIoT development.

Programming & GPIO — GPIO digital input/output, serial monitor usage, LED blinking exercises, PWM control, and hands-on coding with ESP32 in Arduino IDE.

Practical Projects — Traffic light simulation, sensor integration, industrial case studies, troubleshooting techniques, quizzes, viva questions, and assignments.





Module 1: Internet of Things

Introduction to IoT Fundamentals

What is IoT?

The Internet of Things (IoT) is a network of physical devices — machines, vehicles, appliances, and more — embedded with sensors, software, and connectivity that enables them to collect and exchange data over the internet.

- Physical devices connected to the internet
- Embedded with sensors & actuators
- Data collection, transmission & analysis
- Enables smart automation & monitoring
- Bridges the physical and digital worlds



History of IoT



- 1999: Kevin Ashton coined the term "Internet of Things (IoT)" while working at Procter & Gamble (P&G).
- 2000: LG introduced the world's first internet-connected refrigerator.
- 2004: IoT gained worldwide recognition through research papers and industry publications.
- 2008: The number of connected devices exceeded the global human population.
- 2011: IPv6 was launched, enabling trillions of unique IP addresses for IoT devices.
- 2013: Google Glass and other smart wearable devices accelerated IoT adoption.
- 2016: Global IoT devices reached approximately 6.4 billion.
- 2020: The number of connected IoT devices surpassed 30 billion worldwide.
- Today: Industrial IoT (IIoT) is transforming manufacturing, energy, logistics, healthcare, agriculture, and smart cities through automation, real-time monitoring, and predictive maintenance.

IoT by Numbers



75+ Billion Connected Devices by 2025

The explosive growth of IoT is transforming industries worldwide. From smart factories to connected vehicles, the number of IoT devices is projected to surpass 75 billion globally — reshaping how machines, systems, and people interact.



\$1.5 Trillion Global IoT Market Value

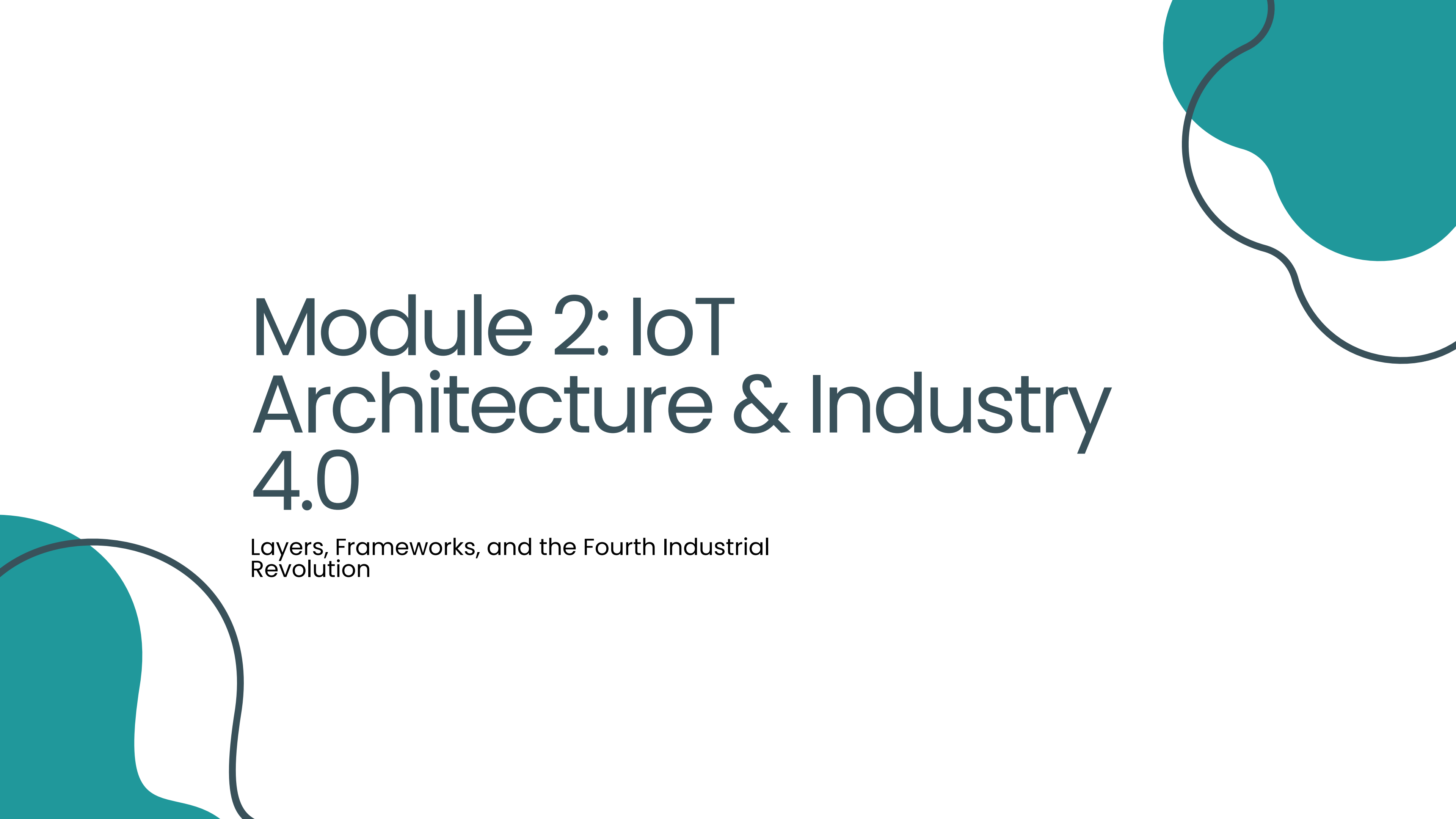
The IoT economy is booming. With massive investments in industrial automation, smart infrastructure, and edge intelligence, the global IoT market is forecast to reach \$1.5 trillion — driving unprecedented innovation and opportunity.

IoT Applications



Smart Homes & Healthcare: Automated lighting, HVAC control, and energy management transform living spaces. Wearable health monitors, remote patient tracking, and smart medical devices enable real-time diagnostics and improved patient outcomes.

Agriculture, Transport & Industry: Precision farming uses soil sensors and automated irrigation. Connected vehicles enable fleet tracking and traffic optimization. Industrial automation leverages IIoT for predictive maintenance, quality control, and smart manufacturing.



Module 2: IoT Architecture & Industry 4.0

Layers, Frameworks, and the Fourth Industrial
Revolution

IoT Architecture Layers

Perception Layer — The physical layer consisting of sensors, actuators, RFID tags, and embedded devices (e.g., ESP32) that collect real-world data such as temperature, humidity, motion, and pressure from the environment.

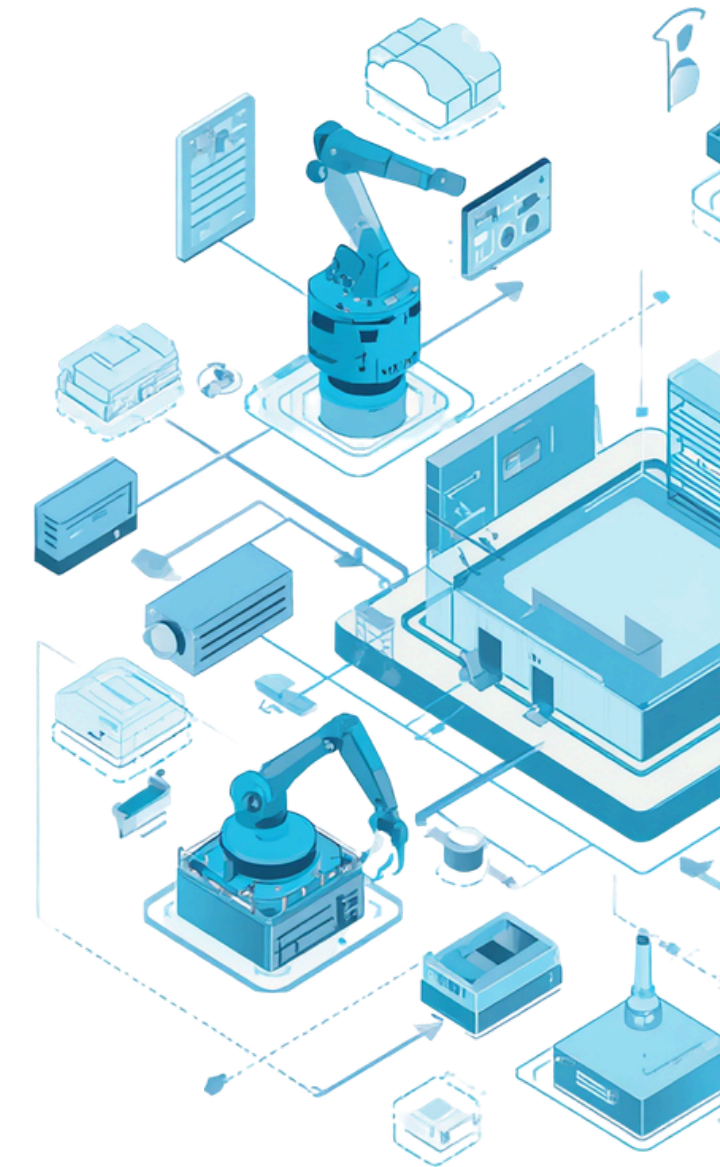
Network Layer — The communication backbone that transmits data from perception devices to processing systems using protocols such as Wi-Fi, Bluetooth, Zigbee, LoRa, MQTT, and CoAP across gateways and the internet.

Application Layer — The top layer delivering processed data as intelligent services and dashboards for industries including smart manufacturing, predictive maintenance, remote monitoring, and IIoT automation platforms.

Industry 4.0

The Fourth Industrial Revolution integrates cyber-physical systems, IoT, and cloud computing into smart manufacturing.

- Cyber-Physical Systems (CPS): Real-world processes monitored and controlled digitally
- IoT Integration: Machines, sensors, and devices interconnected over the internet
- Cloud Computing: Data storage, analytics, and remote access at scale
- Smart Factories: Automated, self-optimizing production environments
- Key Technologies: AI, Big Data, Robotics, 5G, and Edge Computing



IIoT vs Consumer IoT



Industrial IoT (IIoT): Designed for mission-critical operations in factories, energy, and infrastructure. Prioritizes reliability, safety, real-time control, long lifecycle, and high scalability with strict security standards.



Consumer IoT: Focused on convenience and lifestyle — smart home devices, wearables, and entertainment. Optimized for ease of use, low cost, and connectivity, with less emphasis on industrial-grade reliability or safety.

Industry 4.0 Pillars



Big Data Analytics: Collects and processes massive volumes of industrial sensor data in real time. Enables predictive maintenance, quality control, and operational optimization through data-driven decision making.

Autonomous Systems & Horizontal Integration: Self-operating machines and robots reduce human intervention on the factory floor. Horizontal integration connects supply chains, partners, and systems across organizations for seamless end-to-end automation.

Module 3: IoT Ecosystem Components

Sensors • Actuators • Gateways • Cloud • Edge



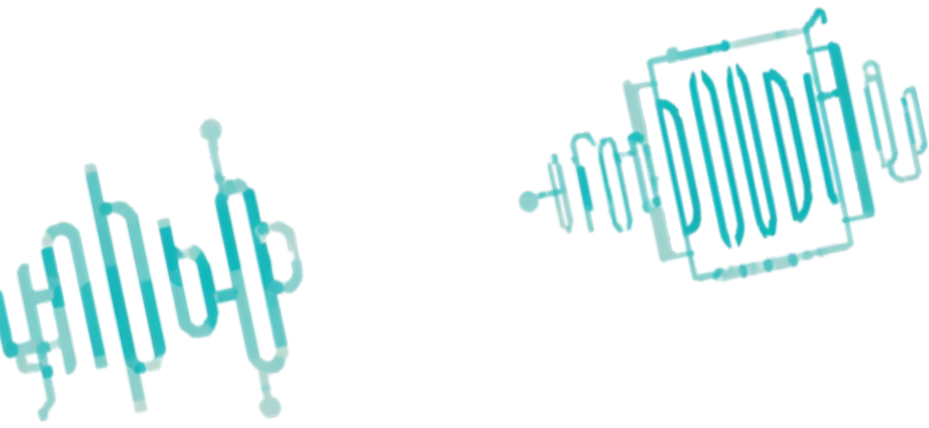
Sensors in IIoT

Sensors are devices that detect and measure physical properties from the environment and convert them into electrical signals for processing.

- Temperature Sensors – Measure heat levels (e.g., thermistors, RTDs, DHT11)
- Pressure Sensors – Monitor force per unit area in industrial pipelines
- Motion Sensors – Detect movement using PIR or accelerometers
- Light Sensors – Measure ambient light intensity (e.g., LDR, photodiode)
- Humidity Sensors – Track moisture levels in air or materials

Sensors are the "eyes and ears" of any IIoT system, feeding real-time data to microcontrollers like the ESP32.





SENSOR TYPES

Analog sensors produce continuous electrical signals proportional to the measured physical quantity (e.g., temperature, pressure, light). Output is a voltage or current that varies smoothly over a range, ideal for precise real-world measurements.

Digital sensors output discrete binary values (HIGH/LOW or 0/1). They detect the presence or absence of a condition and interface directly with microcontrollers without ADC conversion. Examples: PIR motion sensors, limit switches, encoders.

Smart sensors integrate onboard processing, calibration, and communication logic. They perform local data processing, self-diagnosis, and transmit structured data via protocols like I2C, SPI, or UART directly to IoT gateways.

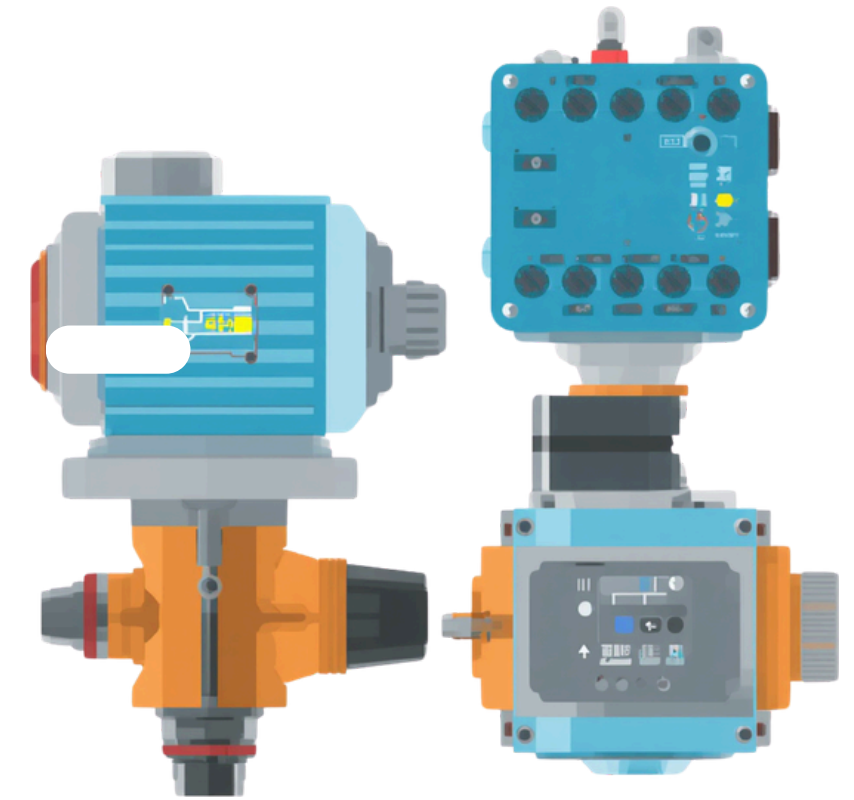


Actuators

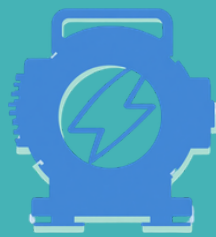
Actuators convert electrical signals into physical actions, enabling machines to respond to commands.

- Motor: Converts electrical energy into rotational or linear motion (e.g., conveyor belts, pumps)
- Valve: Controls fluid or gas flow in pipelines (e.g., solenoid valves in water systems)
- Relay: Electrically operated switch that controls high-power circuits using low-power signals
- Servo Motor: Provides precise angular positioning — used in robotics and CNC machines
- Pneumatic Actuator: Uses compressed air to produce mechanical motion

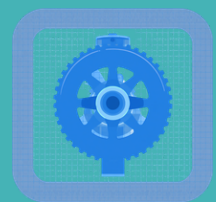
In IIoT, actuators receive commands from ESP32 or gateway devices to perform real-world actions based on sensor data and cloud logic.



Actuator Types



Electric Actuators: Convert electrical energy into mechanical motion. Types include DC motors (speed/direction control), stepper motors (precise positioning), servo motors (angular control), and solenoids (linear push/pull action). Widely used in robotics, conveyor systems, and automated valves.



Mechanical Actuators: Convert rotary or linear motion using mechanical components. Include gears (torque multiplication), pulleys & belts (power transmission), lead screws (linear motion), and cams (motion conversion). Essential in CNC machines, industrial presses, and assembly lines.

IoT Gateways



Acts as a bridge between edge devices (sensors, actuators) and the cloud or enterprise systems, enabling seamless data flow across heterogeneous networks.

Key functions: Protocol translation (MQTT, CoAP, Modbus), data aggregation, local preprocessing, security enforcement, and reliable upstream communication to cloud platforms.

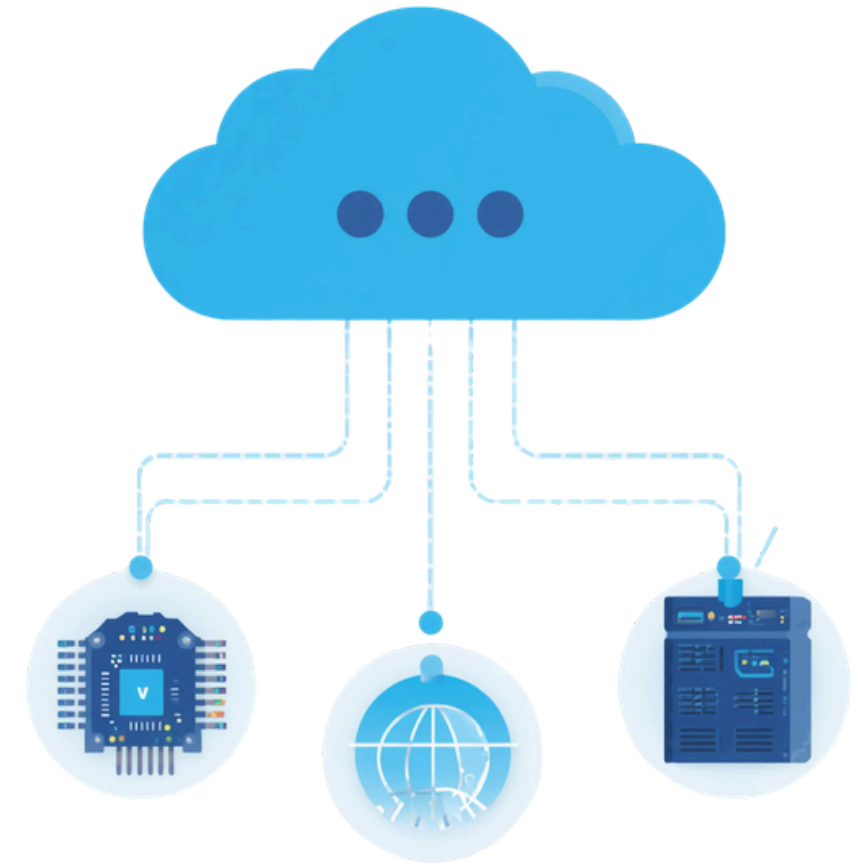
Module 4: Cloud & Edge Computing

Industrial IoT Training — Day 1 & Day 2



Cloud Computing in IoT

Cloud computing provides remote servers for data storage, processing, and analytics in IoT systems:



- Data Storage: IoT devices upload sensor data to cloud platforms (AWS IoT, Azure IoT Hub, Google Cloud IoT)
- Remote Processing: Cloud handles heavy computation offloaded from edge devices
- Real-Time Analytics: Dashboards and analytics engines process millions of data points
- Scalability: Cloud infrastructure scales dynamically with device growth
- Integration: APIs connect IoT data with enterprise systems and AI/ML models



IoT Cloud Platforms

AWS IoT Core: Connects billions of devices, supports MQTT, HTTP, and WebSockets, integrates with AWS Lambda, Amazon S3, and DynamoDB for real-time data processing and storage.

Azure IoT Hub: Microsoft's managed service for bidirectional device-cloud communication, supports MQTT, AMQP, and HTTPS, offers Device Twins and Direct Methods, and integrates seamlessly with Azure Stream Analytics.

Google Cloud IoT: Fully managed platform supporting MQTT and HTTP bridge, integrates with Cloud Pub/Sub, Dataflow, and BigQuery for scalable data ingestion and analytics.

Common Features: Device registry, secure authentication, OTA firmware updates, telemetry ingestion, cloud analytics, real-time monitoring, dashboards, and alerts.

Edge Computing

Edge Computing processes data closer to the source on devices or local gateways rather than sending everything to the cloud. This reduces latency, saves bandwidth, and enables real-time decisions in industrial environments.

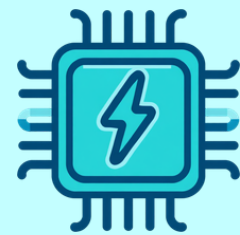
- Lower Latency: Instant response without cloud round-trips
- Bandwidth Savings: Only relevant data is sent upstream
- Offline Resilience: Operates even without internet connectivity
- Enhanced Security: Sensitive data stays on-site
- Real-Time Control: Critical for IIoT automation & safety



Cloud vs Edge



CLOUD COMPUTING: Sends data to remote servers for heavy analytics, machine learning, long-term storage, and enterprise dashboards. Ideal for batch processing, historical trend analysis, and scalable data management across multiple IIoT deployments.



EDGE COMPUTING: Processes data locally on the device or gateway for real-time decisions, ultra-low latency responses, and offline operation. Critical for time-sensitive industrial control, predictive maintenance alerts, and bandwidth-efficient IIoT systems.

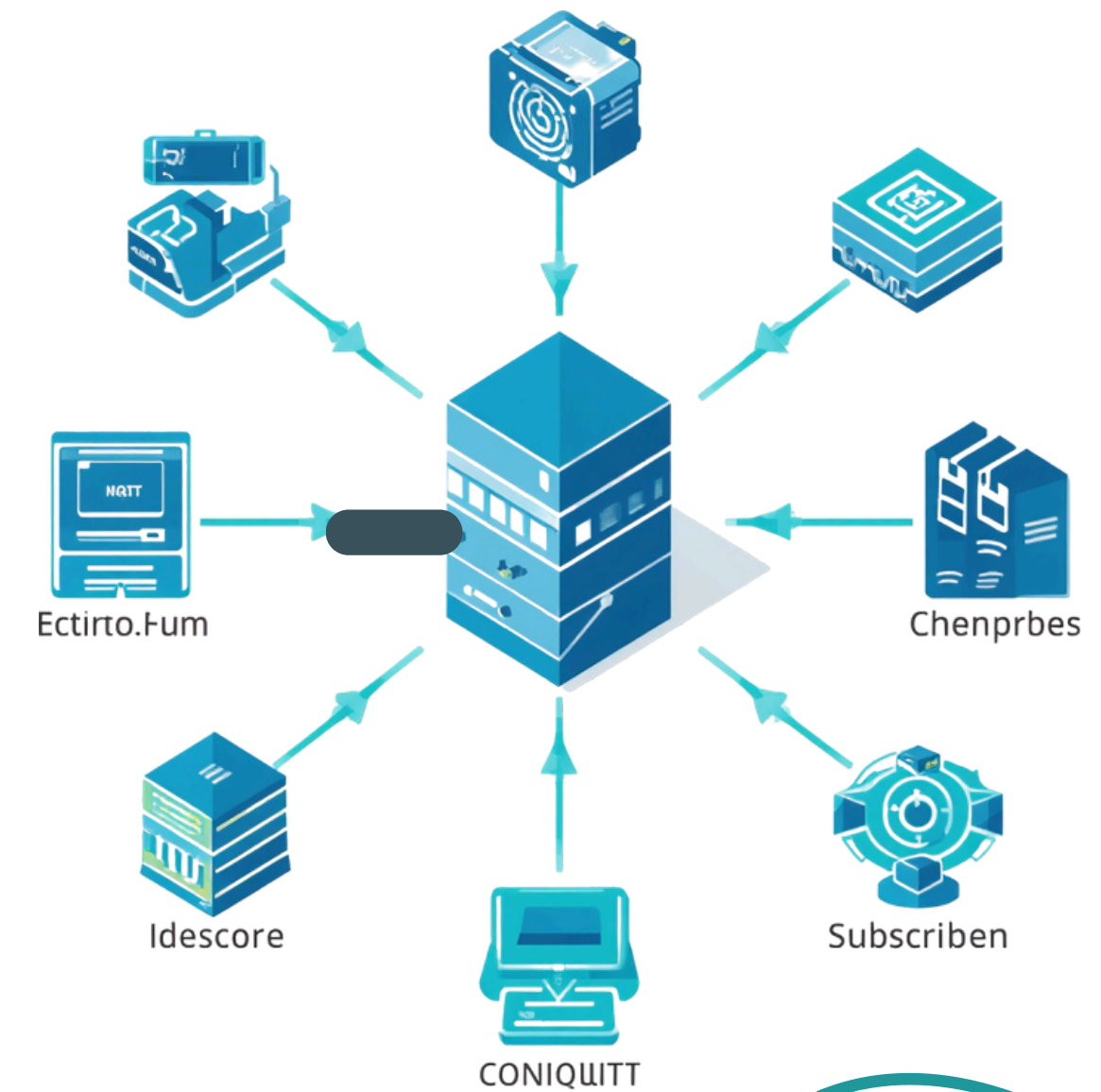
Module 5: Communication Protocols

MQTT, CoAP & IIoT Messaging Standards



MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe messaging protocol designed for constrained devices and low-bandwidth, high-latency networks. It uses a broker-based architecture where devices (clients) publish messages to topics, and other clients subscribe to receive them. MQTT is ideal for IIoT due to its minimal overhead, low power consumption, and reliable delivery with QoS levels 0, 1, and 2.



MQTT Components

The Broker acts as the central message router in an MQTT network. It receives all published messages, filters them by topic, and forwards them to the appropriate subscribers. Examples include Mosquitto, HiveMQ, and AWS IoT Core.

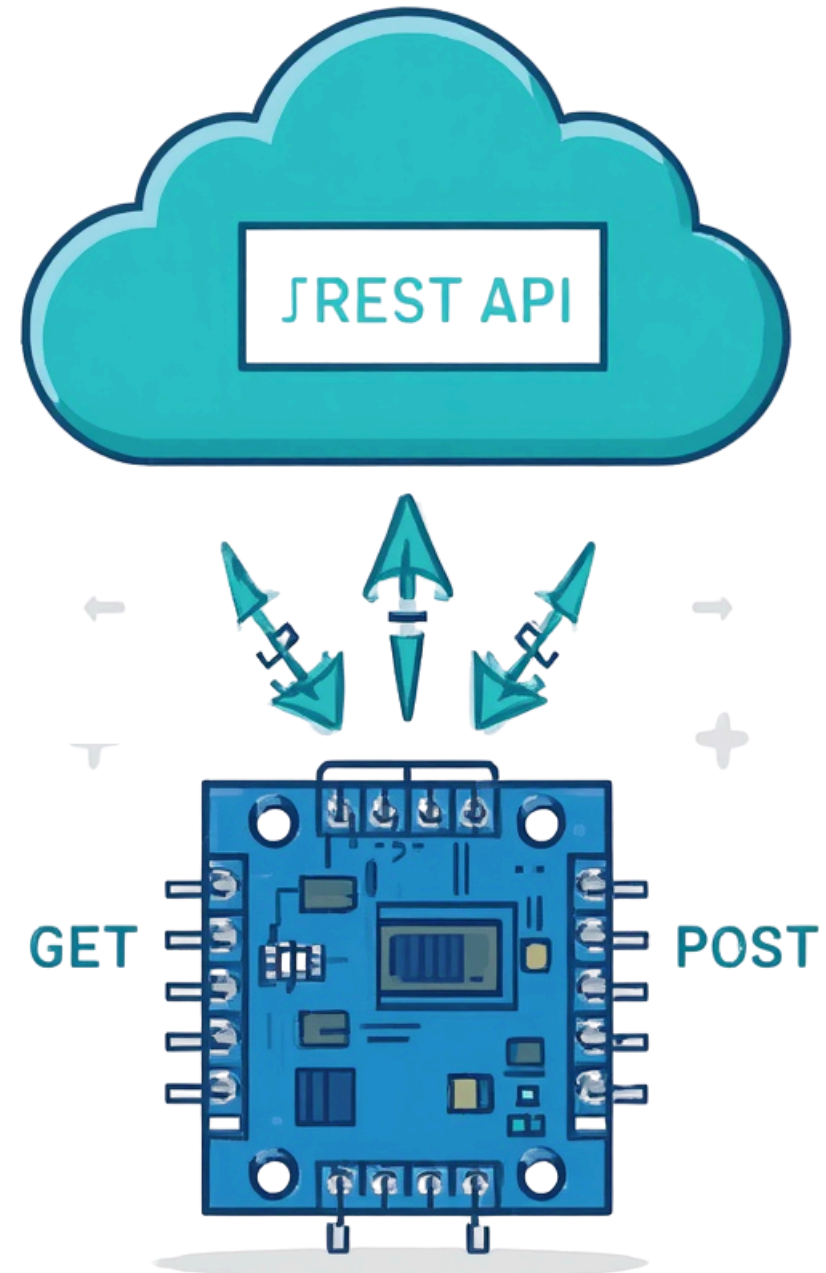
The Publisher is any device or client that sends messages to the MQTT Broker. It publishes data to a specific topic (e.g., "sensor/temperature") without needing to know who receives it. ESP32 commonly acts as a publisher.

The Subscriber is a client that registers interest in one or more topics with the Broker. It receives all messages published to those topics in real time. A single device can be both publisher and subscriber simultaneously.

CoAP Protocol

CoAP (Constrained Application Protocol) is a lightweight RESTful protocol designed for resource-limited IoT devices and low-power networks. Key features:

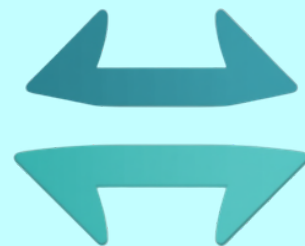
- Uses UDP (not TCP) for minimal overhead
- Supports GET, POST, PUT, DELETE methods like HTTP
- Built-in support for Observe, multicast, and low power modes
- Ideal for M2M communication in constrained environments
- Works over 6LoWPAN, Zigbee, and other IoT networks



MQTT vs CoAP



MQTT (Message Queuing Telemetry Transport): Lightweight publish-subscribe protocol over TCP/IP. Ideal for reliable, persistent messaging in IIoT. Uses a broker to route messages between devices. Supports QoS levels 0, 1, and 2 for delivery guarantees. Best for low-bandwidth, high-latency networks.



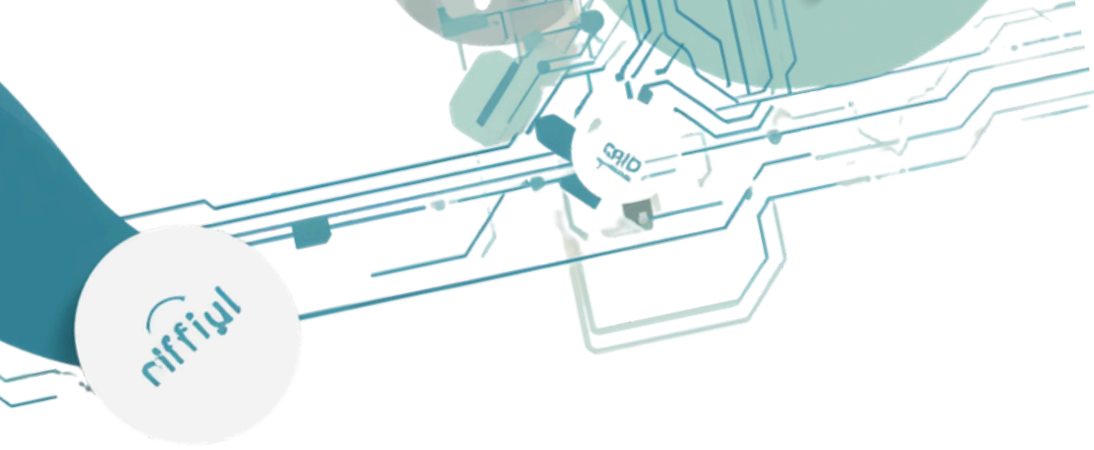
CoAP (Constrained Application Protocol): RESTful protocol designed for constrained devices over UDP. Uses GET, POST, PUT, DELETE methods like HTTP. Supports multicast, low overhead, and asynchronous messaging. Ideal for M2M communication in resource-limited environments. Best for real-time, request-response interactions.

RFID Technology



Radio-Frequency Identification (RFID) uses electromagnetic fields to automatically identify and track tags attached to objects. Key components include: RFID Tag (passive/active), RFID Reader/Antenna, and Backend System. Passive tags have no battery and are powered by the reader's signal, while active tags have their own power source for longer range.

Industrial IIoT Applications of RFID: Asset tracking in manufacturing plants, inventory management in warehouses, access control systems, supply chain visibility, equipment maintenance tracking, and automated production line monitoring. RFID operates at LF (125 kHz), HF (13.56 MHz), and UHF (860–960 MHz) frequency bands.



RFID System

Tags are small transponders attached to objects. Passive tags have no battery and are powered by the reader's electromagnetic field, while active tags have an onboard battery for longer range and continuous broadcasting.

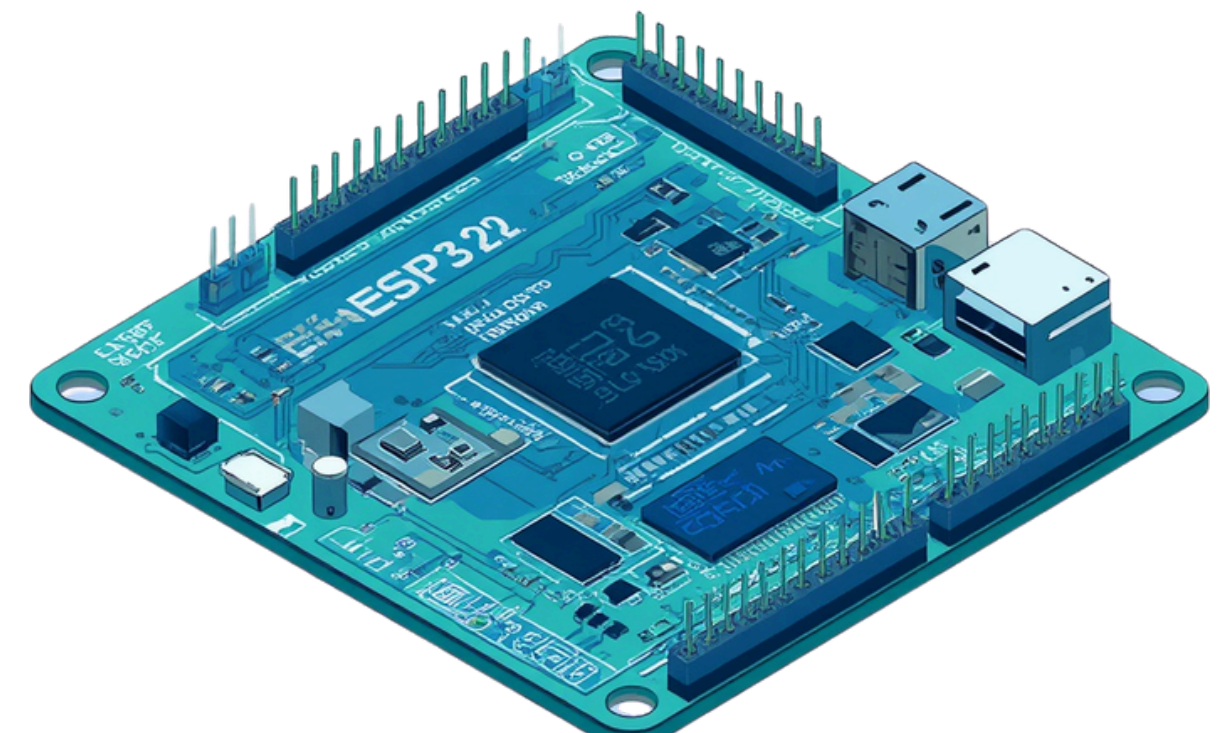
Readers (Interrogators) emit radio frequency signals to communicate with RFID tags. They capture tag data and transmit it to the backend system via wired or wireless interfaces such as USB, Ethernet, or Wi-Fi.

Middleware acts as the bridge between RFID readers and enterprise applications. It filters, aggregates, and processes raw tag data, enabling real-time inventory tracking, asset management, and integration with ERP/IoT platforms.



Module 6: ESP32 Microcontroller Overview

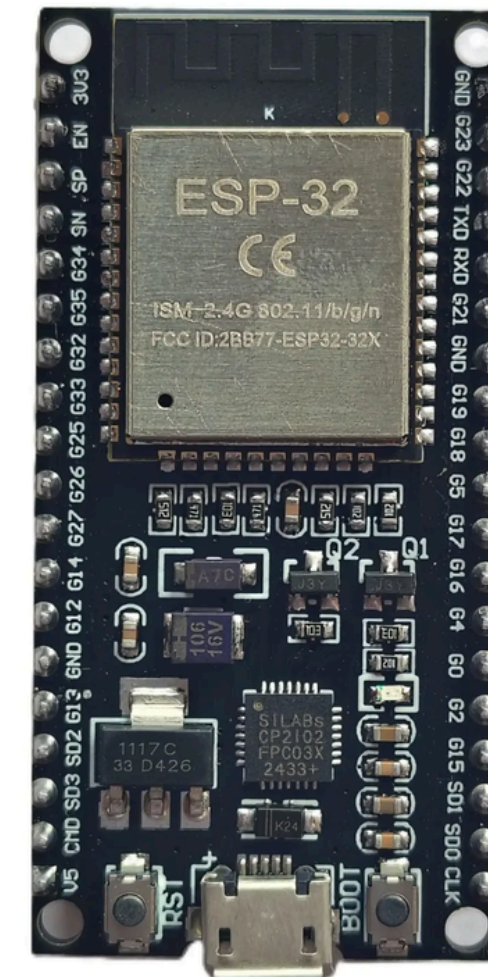
Industrial Internet of Things (IIoT) Training — Day 1



ESP32 Microcontroller

The ESP32 is a powerful, low-cost dual-core 32-bit microcontroller developed by Espressif Systems. It features built-in WiFi (802.11 b/g/n) and Bluetooth (Classic + BLE), making it ideal for Industrial IoT applications.

- Dual-core Xtensa LX6 processor @ up to 240 MHz
- 520 KB SRAM, up to 4 MB Flash memory
- 34 programmable GPIO pins
- Built-in WiFi & Bluetooth connectivity
- Supports SPI, I2C, UART, PWM, ADC, DAC
- Ultra-low power consumption modes
- Widely used in smart sensors, gateways & edge devices





ESP32 SPECIFICATIONS

Dual-core Xtensa LX6 processor running at up to 240MHz, enabling high-speed parallel task execution for real-time IoT applications and concurrent communication handling.

520KB SRAM for runtime data and 4MB Flash memory for program storage, providing ample space for complex IIoT firmware, OTA updates, and data buffering.

Integrated WiFi 802.11 b/g/n and Bluetooth 4.2 (Classic + BLE), enabling seamless wireless connectivity for industrial sensor networks and IoT cloud communication.

ESP32 Pinout

GPIO Pins: 34 programmable pins (GPIO0–GPIO39); GPIOs 34–39 are input-only

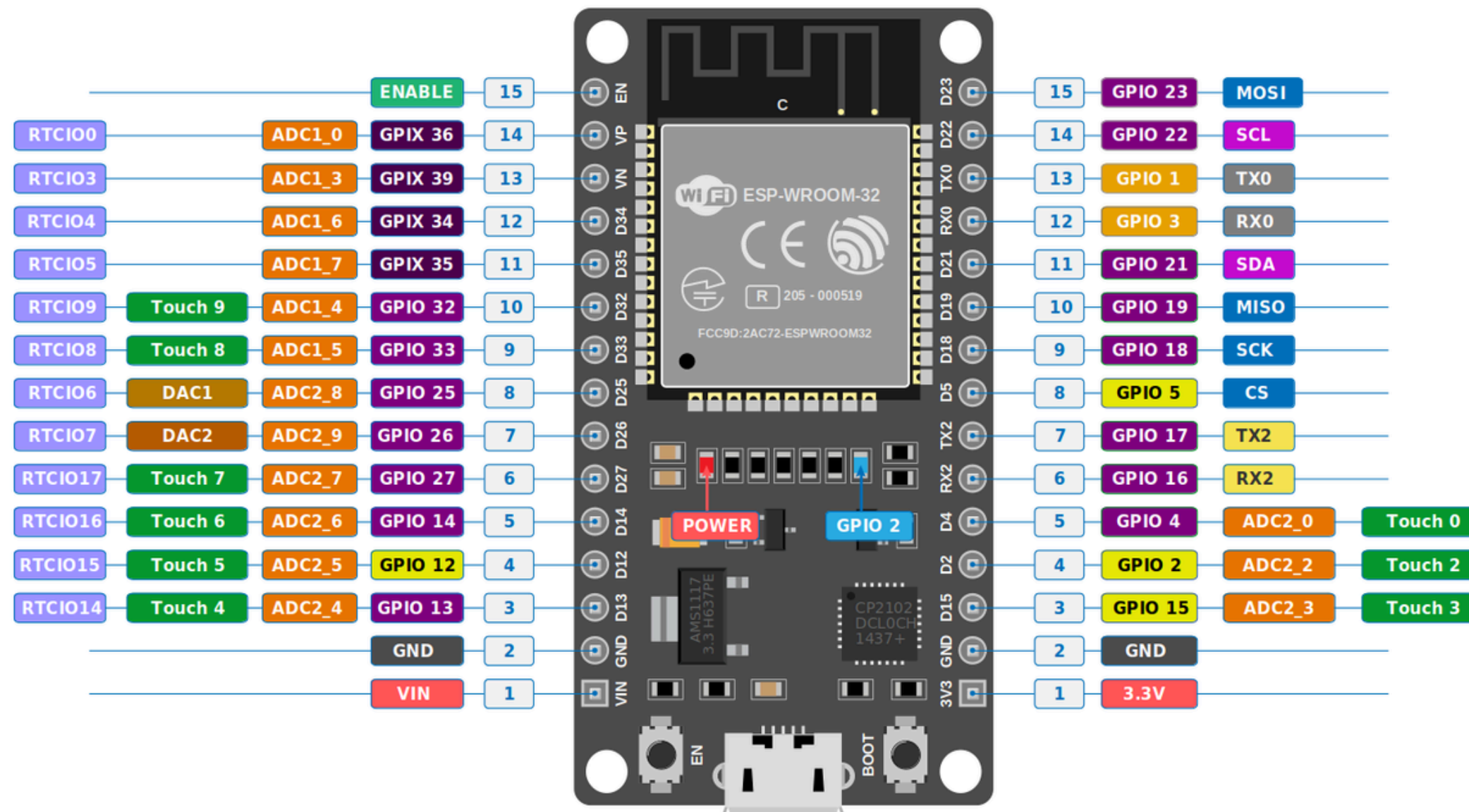
ADC Channels: 18 channels (ADC1 & ADC2) for analog sensor readings (12-bit resolution, 0–4095)

DAC Outputs: 2 channels (GPIO25, GPIO26) for analog voltage output

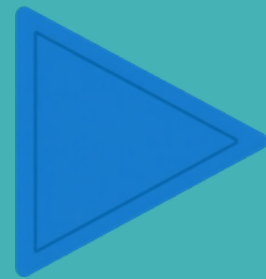
Touch Sensors: 10 capacitive touch pins (T0–T9) for touch-based input

Communication Interfaces: UART (3x), SPI (2x), I2C (2x), I2S, PWM on most GPIO pins

Power Pins: 3.3V, 5V (VIN), GND; logic level is 3.3V (NOT 5V tolerant!)



ESP32 Advantages



Low Power Consumption with Sleep Modes

The ESP32 supports multiple low-power sleep modes including Light Sleep, Deep Sleep, and Hibernation. Deep Sleep reduces current draw to as low as 10 μ A, making it ideal for battery-powered IIoT sensor nodes that must operate for months or years without recharging.



Rich Peripheral Support

The ESP32 integrates a wide range of built-in peripherals: dual-core CPU, Wi-Fi, Bluetooth, 34 GPIO pins, ADC, DAC, UART, SPI, I2C, PWM, and touch sensors. This rich ecosystem reduces external components and simplifies industrial sensor and actuator interfacing.

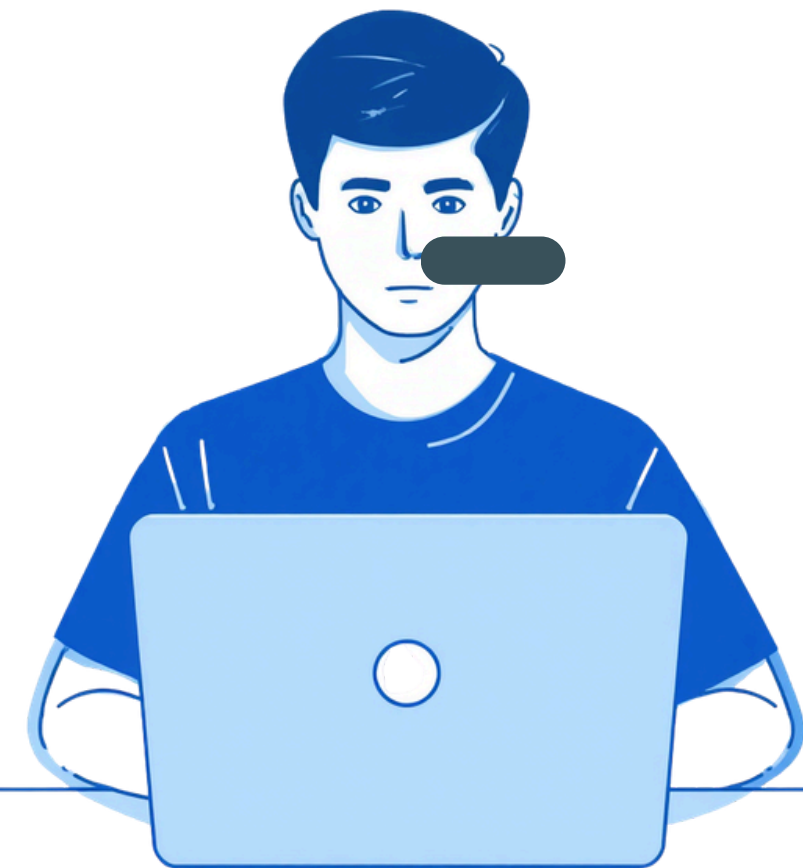
Module 7: Development Environment Setup

Arduino IDE, Wokwi Simulator & ESP32 Toolchain



Arduino IDE

The Arduino IDE (Integrated Development Environment) is a free, open-source platform used to write, compile, and upload code to the ESP32 microcontroller. It uses a simplified version of C/C++, making it accessible for beginners and professionals alike. Key features include a code editor with syntax highlighting, a serial monitor for real-time debugging, a built-in library manager, and one-click code upload via USB. To program the ESP32, simply install the ESP32 board package through the Board Manager, select the correct board and COM port, write your sketch, and click Upload.



Arduino IDE Setup

Download the Arduino IDE from arduino.cc/en/software. Choose the installer for your OS (Windows, macOS, or Linux) and run the setup wizard to complete installation.

Open Arduino IDE, go to File > Preferences, and paste the ESP32 board manager URL into the "Additional Boards Manager URLs" field: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Navigate to Tools > Board > Boards Manager, search for "esp32" by Espressif Systems, and click Install. Once complete, select "ESP32 Dev Module" under Tools > Board to begin programming.



Wokwi Simulator

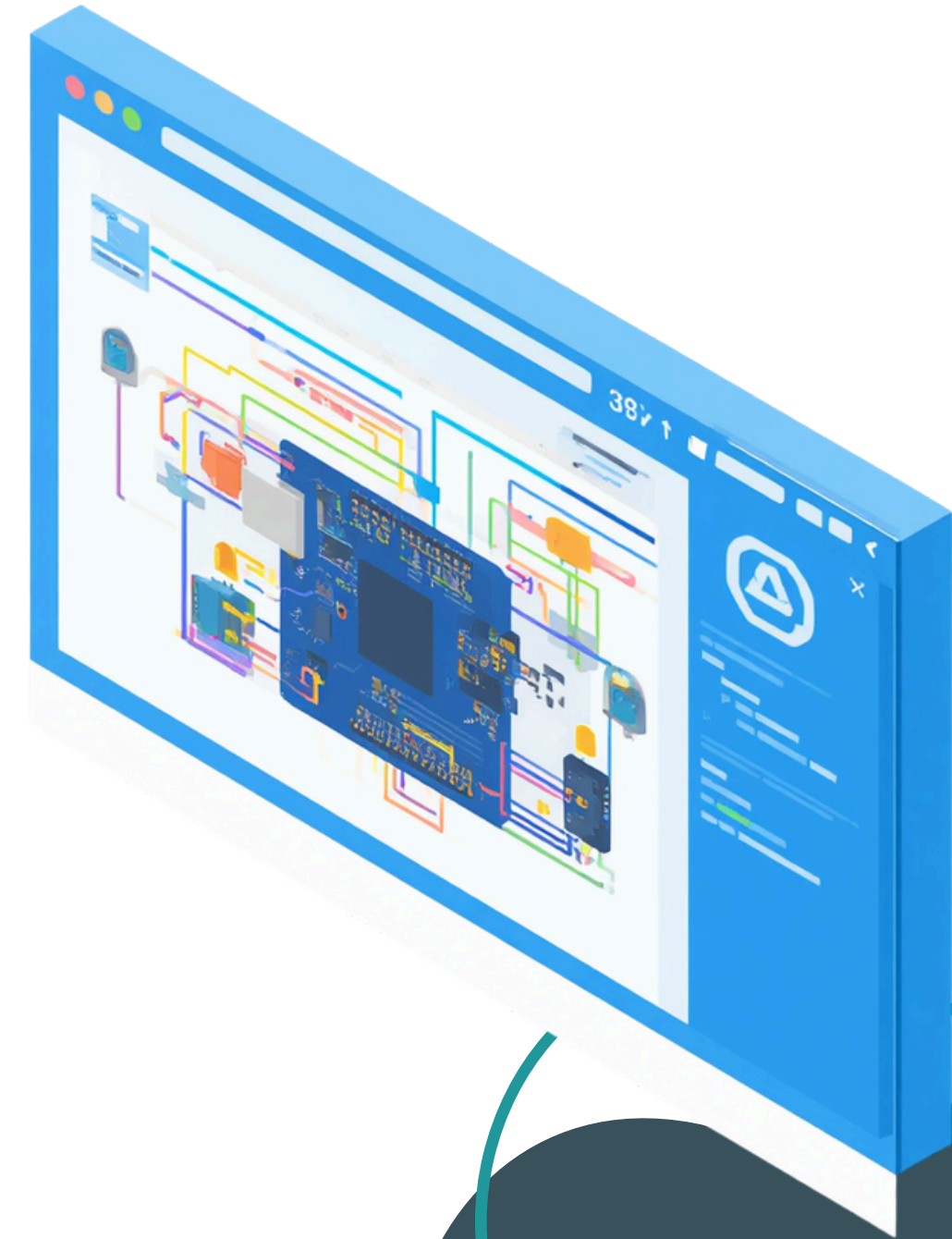
Wokwi is a free, browser-based platform for simulating ESP32, Arduino, and IoT circuits — no physical hardware required.

- Access at wokwi.com — no installation needed
- Supports ESP32, sensors, LEDs, LCDs, motors & more
- Write and run Arduino/MicroPython code in real time
- Visualize GPIO, Serial Monitor, and sensor outputs
- Share and collaborate via project links
- Ideal for prototyping and classroom exercises

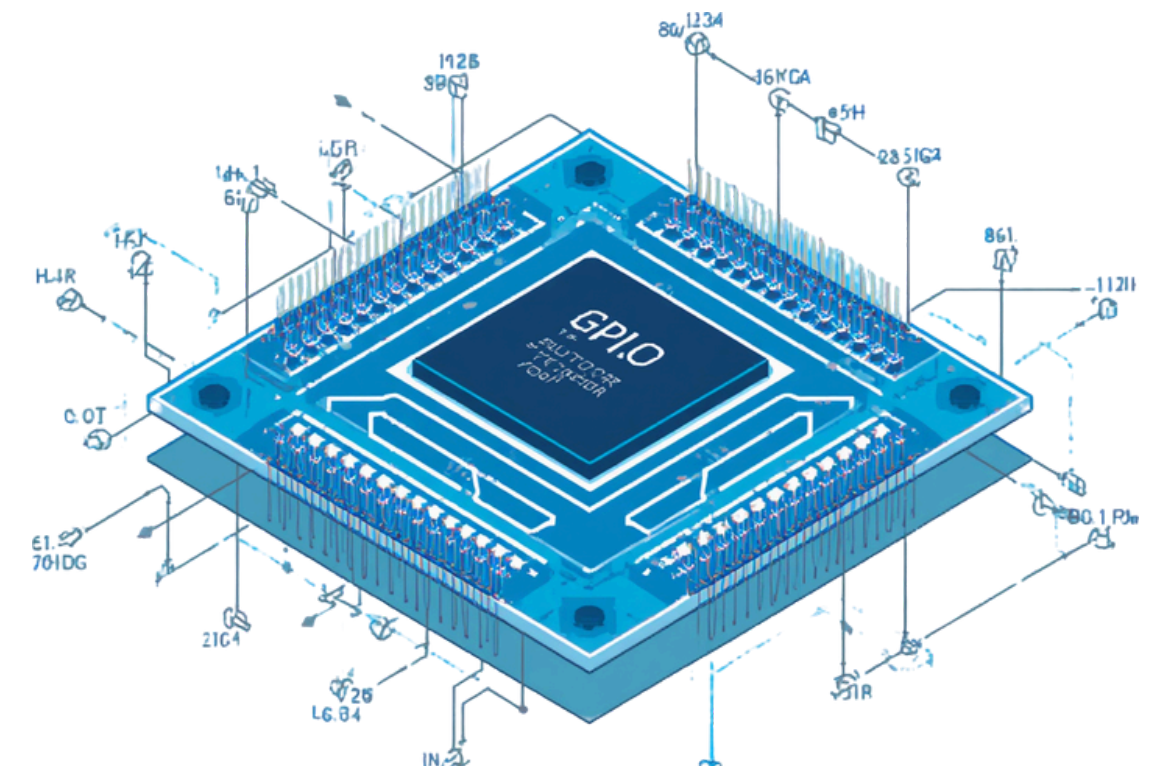


Wokwi Features

Virtual Components: Simulate ESP32, LEDs, sensors & more without hardware.
Real-Time Simulation: Instantly run and test Arduino/ESP32 code in the browser.
Code Debugging: Use Serial Monitor, breakpoints, and live pin state inspection.
Circuit Sharing: Share circuits via URL for collaboration and peer review.
Free, browser-based — no installation required!

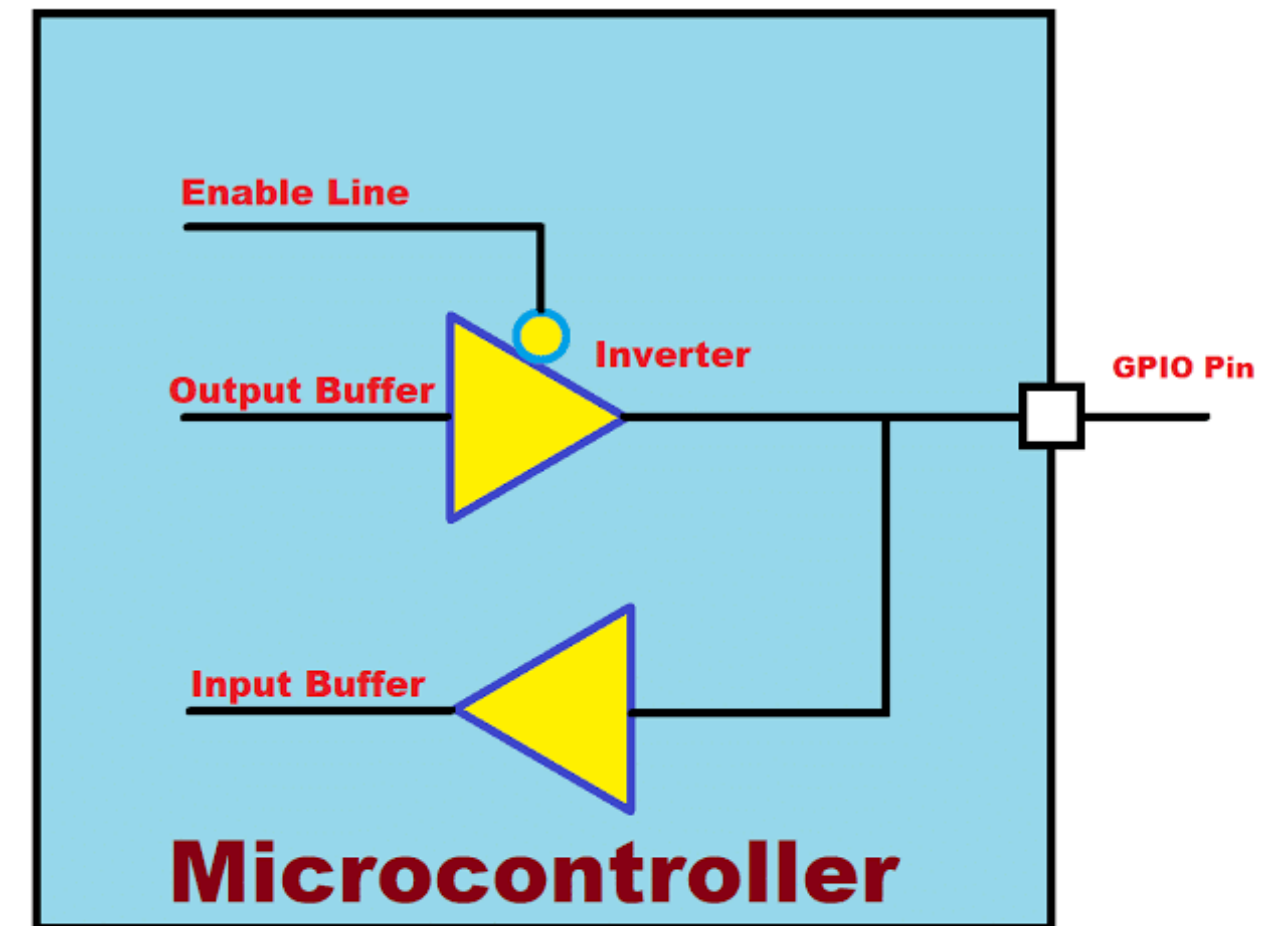


Digital I/O, Pin Modes, and ESP32 Control Logic

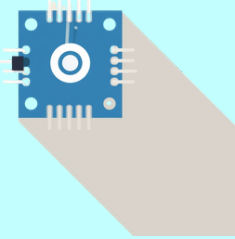


GPIO: General Purpose I/O

GPIO (General Purpose Input/Output): GPIO pins allow the ESP32 to interface with external devices such as sensors, buttons, LEDs, and relays. Each GPIO can be configured as an INPUT (to read signals) or an OUTPUT (to control devices). The ESP32 provides 34 programmable GPIO pins, with many supporting advanced features like ADC, DAC, PWM, I2C, SPI, and UART. In Arduino programming, use `pinMode()` to configure the pin, `digitalRead()` to read an input, and `digitalWrite()` to set an output to HIGH or LOW.

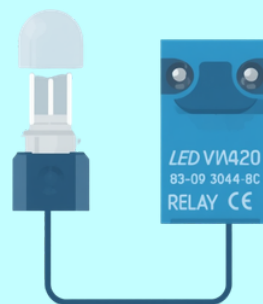


GPIO: Digital I/O Modes



INPUT MODE — Reading Digital Signals

Configured with `pinMode(pin, INPUT)` or `INPUT_PULLUP`. Reads HIGH or LOW from sensors, push buttons, limit switches, and digital detectors. Used with `digitalRead()` to capture real-world binary states and trigger logic in the program.



OUTPUT MODE — Controlling Digital Actuators

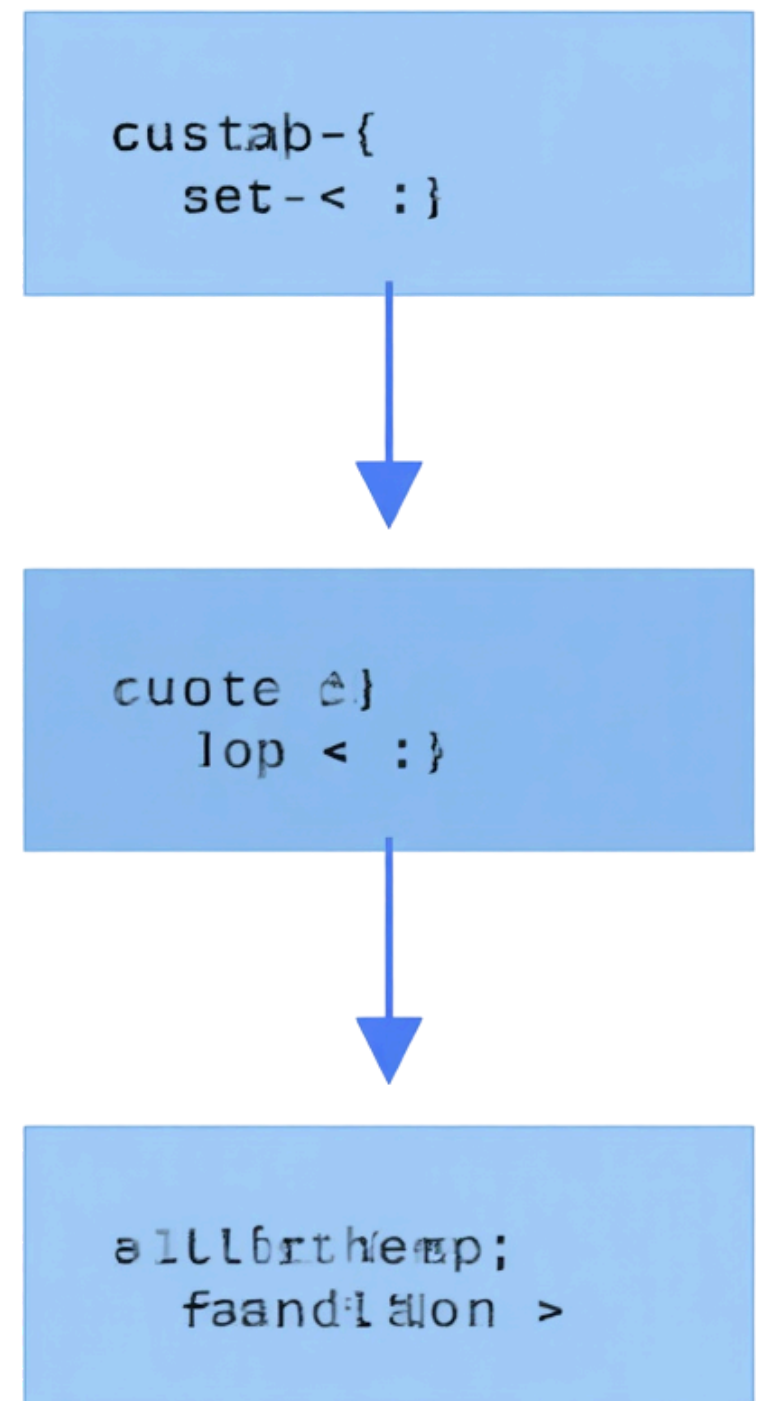
Configured with `pinMode(pin, OUTPUT)`. Drives HIGH (3.3V) or LOW (0V) signals using `digitalWrite()`. Controls LEDs, relays, buzzers, motor drivers, and indicator lights — the foundation of ESP32 actuation in IIoT systems.

ESP32 Program Structure

`setup()` — Runs once at startup. Used to initialize pins, serial communication, sensors, and peripherals. Called automatically before the main loop begins.

`loop()` — Runs continuously after `setup()`. Contains the main program logic: reading sensors, controlling outputs, and handling communication. Executes repeatedly until power is off.

Functions — Reusable blocks of code defined outside `setup()` and `loop()`. Improve readability, modularity, and maintainability. Called by name with parameters and return values as needed.



Serial Monitor

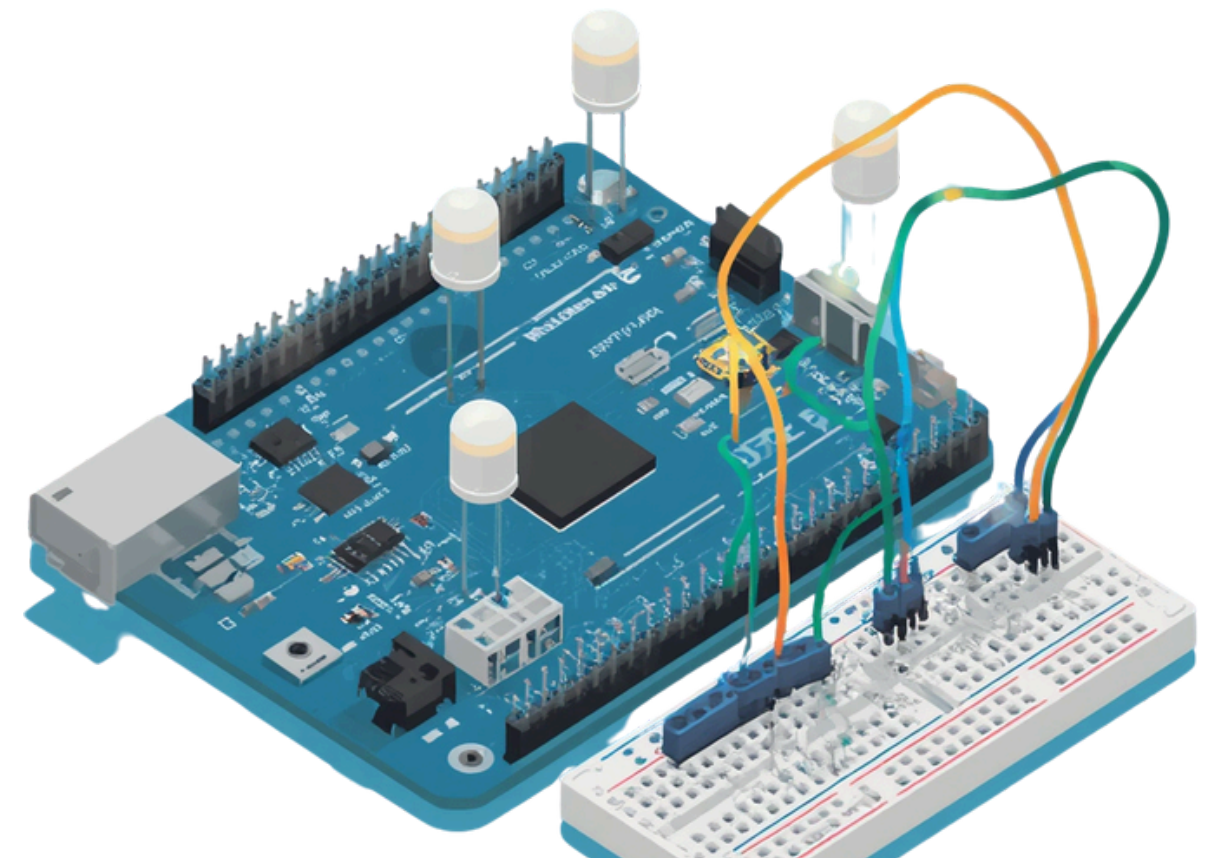
The Serial Monitor is a built-in tool in Arduino IDE for debugging and communication between the ESP32 and a computer via UART.

- Open via: Tools > Serial Monitor (Ctrl+Shift+M)
- Set baud rate to match `Serial.begin()` — e.g., 115200
- Use `Serial.print()` and `Serial.println()` to output values
- Use `Serial.read()` to receive input from the monitor
- Essential for testing sensors, variables, and program flow



Module 9: Practical Projects

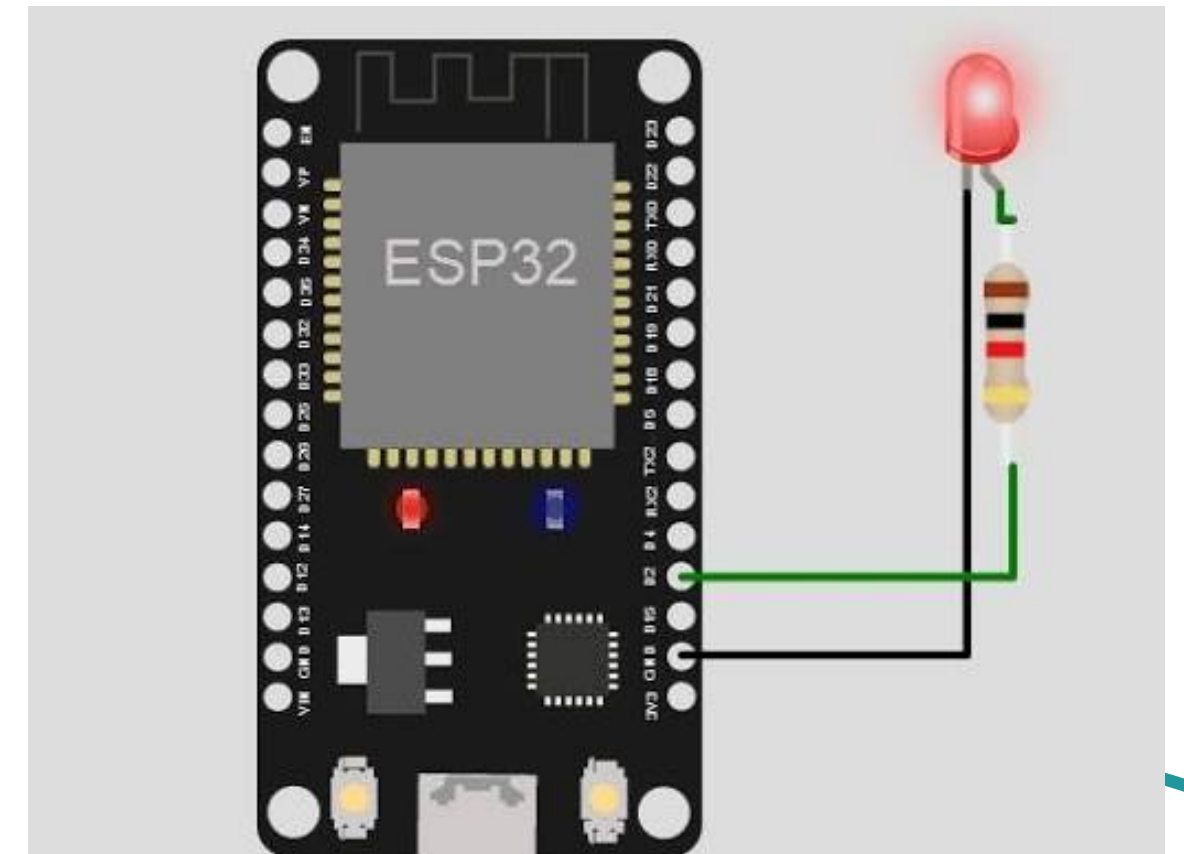
Hands-On ESP32 IoT Project Exercises



LED Blinking Project

First ESP32 project to understand GPIO output and timing:

- Connect LED to GPIO pin 2 with a 220 Ω resistor to GND
- Use `pinMode(2, OUTPUT)` to configure the pin
- Use `digitalWrite(2, HIGH/LOW)` to turn LED on/off
- Use `delay(1000)` for 1-second timing intervals
- Upload sketch via Arduino IDE or test in Wokwi
- Observe LED blinking every second — the "Hello World" of hardware!





LED BLINKING

Connect the LED anode to a GPIO pin (e.g., GPIO 2) on the ESP32 and cathode to GND through a 220Ω resistor. Ensure correct polarity to avoid damage to the LED or the microcontroller pin.

In your Arduino sketch, use `pinMode(2, OUTPUT)` in the `setup()` function to configure GPIO 2 as a digital output pin. This enables the ESP32 to drive the pin HIGH or LOW to control the LED.

In the `loop()` function, use `digitalWrite(2, HIGH)` to turn the LED on, `delay(1000)` to wait 1 second, `digitalWrite(2, LOW)` to turn it off, then `delay(1000)` again to create a 1 Hz blinking cycle.

