

PROJECT REPORT

Name: Keerathan D

Program: IOT Mentorship

Topic: IOT based smart factory system

Mentioned Tasks of this project:

- **Task 1:** Using one example design the Master-Slave Architecture
- **Task 2:** Study about Operating System, Multithreading Concept

INDEX

| Sl.no | Content | Pg.no |
|-------|---|-------|
| 01 | Abstract of the topic | 02-04 |
| 02 | Introduction to the topic | 05-12 |
| 03 | Methodology of the topic | 13-15 |
| 04 | Tasks and figure attested about the topic | 16-28 |
| 05 | Conclusion | 29-30 |

ABSTRACT OF IOT BASED SMART FACTORY SYSTEM

The core of this revolution is the Internet of Things (IoT), which enables the creation of "Smart Factories" by interconnecting physical machinery, environmental sensors, and data-driven analytics. These smart factories benefit from enhanced operational efficiency, predictive maintenance, automated process control, and significantly improved workplace safety. While large-scale corporations have widely adopted these capital-intensive technologies, a significant gap remains for small-to-medium-sized enterprises (SMEs) that lack the resources for proprietary, large-scale system overhauls. This project addresses this gap by presenting the complete design, analysis, and conceptual framework for a low-cost, modular, and scalable IoT-based Smart Factory system built using accessible, open-source hardware.

The primary objective of this project is to develop a proof-of-concept prototype that integrates multiple critical factory functions into a single, cohesive system. The system is designed to be deployed as a "node" that can monitor a specific workstation, storage area, or piece of machinery. By deploying multiple nodes, a comprehensive smart factory network can be established incrementally. This project focuses on three primary pillars of industrial operation:

- (1) Workplace Safety and Security
- (2) Process Automation and Monitoring
- (3) Energy Management and Environmental Control.

The fundamental problem this system solves is the lack of real-time, actionable data in traditional, non-digitized factory environments. In such settings, safety checks (like gas leak detection) are often manual and periodic, inventory management (like tank levels) relies on physical inspection, and energy is wasted by lighting and machinery left idle in unoccupied areas. Furthermore, management and oversight are restricted to physical presence, with no mechanism for remote monitoring or receiving critical alerts for events like an after-hours intrusion or a critical machine failure. This project proposes an affordable solution that directly digitizes and automates these core functions, providing immediate, tangible value.

The hardware architecture of the proposed system is centered around the **Arduino Uno** microcontroller, which serves as the central processing unit for the node. This controller is responsible for executing the core logic, interfacing with all peripheral components, and (in a full implementation) communicating with a cloud platform. The system's functionality is achieved by integrating a suite of input sensors and output actuators.

Input Subsystems (Sensors): The sensory apparatus is designed for 360-degree situational awareness of the node's environment.

- **Environmental Safety Sensor (MQ-Series):** An MQ gas sensor is integrated to provide real-time monitoring of the air for flammable or toxic gases, such as methane, propane, or carbon monoxide. This component is critical for preventing fires, explosions, or health hazards. It provides an analog signal proportional to the gas concentration, allowing for an early-warning system.
- **Process Monitoring Sensor (HC-SR04 Ultrasonic):** This versatile sensor is employed for non-contact distance measurement. Its primary applications within the factory context are twofold: (a) as a tank level sensor, it can be mounted above a vat or silo to provide continuous, real-time data on raw material levels, automating inventory management; and (b) as a proximity or counting sensor, it can be placed alongside a conveyor belt to count products or create a safety "curtain" that stops a machine if a worker gets too close.
- **Presence & Security Sensor (PIR Motion Sensor):** A Passive Infrared (PIR) sensor detects the presence of personnel. This data is leveraged for two key functions: (a) smart energy management, by automatically controlling lights and non-essential machinery, and (b) security, by detecting unauthorized motion in a restricted area or during off-hours.

Output Subsystems (Actuators): The system acts upon the physical environment using the following components:

- **High-Voltage Control (Relay Modules):** Two relay modules are used as electromechanical switches. They bridge the gap between the Arduino's 5V logic and the 120V/240V AC mains, allowing the system to control industrial-grade lighting, high-power ventilation fans, or the power supply to specific machinery (e.g., for an emergency stop).

- **Local Alerts & Displays:** An audible buzzer provides an immediate, on-site alarm for critical events (e.g., high gas levels, security breach). This is supplemented by a 7-segment digital display that serves as a local dashboard, providing workers with real-time readouts of the product count, tank level, or gas concentration.
- **Status Indication:** The high-voltage lights can be programmed to function as a classic industrial "Andon" system, using different colors (e.g., Green for normal, Yellow for warning/low-material, Red for critical stop) to visually communicate the workstation's status across the factory floor.

This is achieved by integrating the Arduino controller with an (assumed) internet connectivity module, such as an **Wi-Fi chip**. This module transforms the standalone node into a web-connected device. The system employs the lightweight **MQTT (Message Queuing Telemetry Transport)** protocol to publish all sensor data (gas PPM, presence status, distance in cm) to a central MQTT broker hosted on a cloud platform (e.g., AWS IoT, Google Cloud, or Blynk).

This IoT integration unlocks the system's most powerful capabilities. All data is logged historically, allowing for trend analysis and data-driven decision-making. More importantly, it enables a **remote dashboard**, accessible via a web browser or mobile application. A factory manager, from anywhere in the world, can:

- **Visualize** the real-time status of every node.
- **Receive** instant, automated alerts (via email, SMS, or push notification) for critical events, such as a gas leak or an intrusion.
- **Remotely control** actuators, such as manually activating the ventilation fans or turning off a machine in an emergency.

INTRODUCTION TO SMART FACTORY SYSTEM

The New Industrial Age: From Steam to Data

The history of human industry is marked by four distinct transformative periods, or "Industrial Revolutions," each of which fundamentally reshaped how societies produce, work, and live.

- **The First Industrial Revolution (Industry 1.0)**, beginning in the late 18th century, was characterized by mechanization. The invention of the steam engine and the proliferation of water power moved production from the hands of artisans into the first factories. This was the shift from an agrarian society to an industrial one.
- **The Second Industrial Revolution (Industry 2.0)**, starting in the late 19th century, was defined by mass production. The advent of electricity, the assembly line, and new industrial processes like the Bessemer process for steel, allowed for the high-volume, standardized production that built the modern world.
- **The Third Industrial Revolution (Industry 3.0)**, beginning in the 1970s, was the digital revolution. The introduction of computers, programmable logic controllers (PLCs), and information technology (IT) brought automation to the factory floor. This revolution automated individual machines and processes, but these systems often operated in isolated "silos."

Today, we are in the midst of the **Fourth Industrial Revolution**. This new era is not defined by a single invention but by a *fusion* of technologies. It is the convergence of the physical, digital, and biological spheres. At its core, Industry 4.0 is about connection and intelligence. It is powered by the use of automation and data analysis technologies to create "smart factories" where machines, systems, and people communicate with one another and with the wider supply chain in real-time.

Unlike Industry 3.0, which focused on automating tasks, Industry 4.0 focuses on automating and optimizing the entire *system*. It leverages data as its primary raw material to reduce defects, predict equipment failures, respond to changing customer demands, and create a production environment that is intelligent, adaptive, and autonomous.

A "Smart Factory" is the practical manifestation of Industry principles. It is not simply a factory with robots; it is a highly connected and intelligent manufacturing facility that uses a network of interconnected systems and machinery to generate, share, and analyze data in real-time. This continuous flow of information is used to improve and optimize every aspect of the production process, from the supply chain to the end-of-line quality check.

While a traditional automated factory (Industry 3.0) follows pre-programmed, linear instructions, a smart factory (Industry 4.0) is a dynamic, self-optimizing ecosystem. Its key characteristics are built on a set of core design principles:

1. **Interoperability:** The ability of machines, sensors, devices, and people to connect and communicate with each other. This is the foundational layer, often enabled by the Industrial Internet of Things (IIoT).
2. **Information Transparency (Virtualization):** The ability to create a "digital twin," or a virtual copy, of the physical factory. This digital model is fed with real-time sensor data, allowing for simulation, monitoring, and analysis without disrupting physical operations.
3. **Decentralization:** The ability of individual machines and cyber-physical systems to make their own autonomous decisions. Instead of a single central "brain" controlling everything, a decentralized system empowers a machine to, for example, detect a fault and automatically slow itself down or request maintenance.
4. **Real-Time Capability:** The ability to collect and analyze data and make decisions almost instantaneously. This allows a smart factory to react immediately to a quality defect, a supply chain shortage, or a sudden change in a customer order.
5. **Service Orientation:** Shifting from a product-based model to a service-based one, where internal and external services (e.g., maintenance, analytics) are offered and utilized across the network.
6. **Modularity & Flexibility:** The ability to easily reconfigure the factory to adapt to changing needs. A modular smart factory can scale production up or down, or switch to a different product (mass customization), with minimal downtime and retooling.

The Role of IoT: The Factory's Nervous System

If the smart factory is the "body," the **Internet of Things (IoT)** is its digital nervous system. In an industrial context, this is often called the **IIoT (Industrial Internet of Things)**. The IIoT refers to the vast network of interconnected sensors, actuators, machines, and other industrial devices that collect, exchange, and analyze data within the manufacturing environment. It is the foundational technology that makes the smart factory's characteristics (like interoperability and real-time capability) possible.

The role of IoT in the factory can be understood in a simple loop:

1. **Sense (Data Acquisition):** This is the "heart" of IoT. Sensors are the digital "eyes and ears" of the factory floor. They are embedded in machines, on conveyor belts, in storage tanks, and throughout the facility to measure critical parameters. Examples include:
 - **Temperature & Humidity Sensors:** Monitoring environmental conditions.
 - **Vibration Sensors:** Attached to motors to detect early signs of wear.
 - **Ultrasonic & Proximity Sensors:** To count products or measure liquid levels.
 - **Gas & Smoke Sensors:** For critical environmental safety.
2. **Communicate (Data Transport):** The data from these sensors is then transmitted—via protocols like Wi-Fi, Ethernet, LoRa, WAN, or 5G—from the "edge" to a central processing system, which is typically in the cloud.
3. **Analyze (Data Processing):** Raw data is processed and analyzed to identify patterns, anomalies, and insights.
4. **Act (Actuation):** Based on the analysis, the system sends a command back to an **actuator** on the factory floor to perform a physical action. An actuator is a component that *acts* on the environment, such as:
 - **Relays:** Switching on a high-power ventilation fan.
 - **Motors:** Adjusting a valve or moving a robotic arm.
 - **Alarms:** Triggering a buzzer or a warning light.

This "sense-communicate-analyze-act" loop is the fundamental mechanism of the IIoT.

Key Enabling Technologies (Part 1): Data and Cloud

The IIoT generates an unprecedented volume, velocity, and variety of data. Two other technologies are essential to handle and find value in this data "deluge": Cloud Computing and Big Data Analytics.

Cloud Computing: The Factory's External Brain

In the past, all factory software and data—from scheduling systems (MES) to enterprise resource planning (ERP)—was stored on local, on-premise servers. This was expensive to maintain, difficult to scale, and created "data silos" where information was trapped within one department.

Cloud Computing acts as the scalable, centralized, and accessible "backbone" for the smart factory. It provides on-demand computing resources (storage, processing power, software) over the internet. Its role is:

- **Massive Data Storage:** It is the only cost-effective way to store the terabytes of data generated by thousands of IoT sensors every day.
- **Centralized Platform:** It breaks down data silos by providing a single "source of truth." Data from the factory floor (OT, or Operational Technology) can be seamlessly integrated with data from the front office (IT, or Information Technology), such as sales orders or inventory levels.
- **Scalability:** A manufacturer can scale their computing resources up or down almost instantly to meet demand, paying only for what they use, rather than investing in massive, underutilized servers.
- **Accessibility:** It enables remote monitoring and control. A factory manager can access a real-time dashboard of their facility from a laptop or smartphone anywhere in the world.

Big Data & Analytics: Finding the "Signal in the Noise"

Collecting data is useless if it is not analyzed for actionable insights. **Big Data & Analytics** refers to the processes and tools used to analyze these massive, complex datasets. In a smart factory, analytics moves beyond simple reporting (what happened) to advanced prediction and prescription (what *will* happen and what *should* we do).

Key applications include:

- **Predictive Maintenance:** By analyzing vibration and temperature data, an analytics model can predict that a specific motor will fail in the next 72 hours, automatically scheduling maintenance *before* the costly breakdown occurs.
- **Real-Time Quality Control:** By using cameras and sensors, an analytics system can spot a microscopic defect on a product moving at high speed, immediately flagging it for removal.
- **Process Optimization:** Analytics can identify hidden bottlenecks or inefficiencies in a production line that a human operator would never see, allowing for continuous improvement.

Key Enabling Technologies (Part 2): Intelligence and Simulation

Beyond data storage and analysis, two other technologies provide the higher-level intelligence and simulation capabilities that define a truly smart factory: AI and Cyber-Physical Systems (including Digital Twins).

Artificial Intelligence (AI) and Machine Learning (ML)

If analytics can find patterns, **Artificial Intelligence (AI)** and **Machine Learning (ML)** can learn from those patterns to make intelligent, autonomous decisions. AI is the engine that drives optimization and self-adaptation.

- **AI-powered robots** can learn to pick and place unfamiliar objects.
- **ML models** can analyze real-time supply chain data and automatically adjust production schedules to account for a predicted shipping delay.
- **AI-driven quality control** can learn the difference between a cosmetic scratch and a critical failure, improving accuracy over time.
- **Generative AI** can even suggest new product designs or factory layouts that are optimized for efficiency and material usage.

AI and ML are what allow the smart factory to move from being simply "connected" to being "intelligent."

Cyber-Physical Systems (CPS) & The Digital Twin

A **Cyber-Physical System (CPS)** is the tight integration of computation, networking, and physical processes. A machine on the factory floor equipped with sensors, processing power, and a network connection is a CPS. It is "aware" of its own state and its environment, and it can communicate that state to other systems. The entire smart factory is, in effect, a large-scale system of interconnected CPS.

The ultimate expression of a CPS is the **Digital Twin**.

A Digital Twin is a highly detailed, dynamic, and virtual replica of a physical object or an entire process. This is far more than a static 3D model; it is a living simulation that is fed real-time data from its physical counterpart via IoT sensors.

- **Monitoring:** A manager can look at the Digital Twin of their factory on a screen and see the real-time status, speed, and health of every single machine.
- **Simulation:** A company can test a change to the production line (e.g., "What if we increase the speed of this conveyor belt by 10%?") on the Digital Twin *first*. The simulation can predict the impact on downstream machines, energy use, and output, preventing costly real-world errors.
- **Optimization:** An AI can run thousands of "what-if" scenarios on the Digital Twin overnight to discover the single most efficient configuration for the next day's production run.

Benefits and Value Proposition

The integration of these technologies—IIoT, Cloud, Big Data, AI, and Digital Twins—delivers a powerful and transformative value proposition for manufacturers. The benefits of adopting a smart factory model are comprehensive, impacting every part of the business.

1. **Massive Gains in Efficiency and Productivity:** This is the most significant benefit.
 - **Reduced Downtime:** Predictive maintenance prevents unexpected machine failures, moving from a reactive "fix it when it breaks" model to a proactive "fix it before it breaks" model.
 - **Improved OEE (Overall Equipment Effectiveness):** By optimizing processes and reducing downtime, OEE—a key metric of manufacturing productivity—is dramatically increased.
2. **Enhanced Product Quality:**
 - **Real-Time Monitoring:** Systems can catch production errors or quality defects the second they happen, not at the end of the line, reducing waste and scrap.
 - **Consistency:** Automated processes ensure that every product is manufactured to the exact same standard, improving consistency and customer satisfaction.
3. **Improved Workplace Safety and Sustainability:**
 - **Worker Safety:** IoT sensors can monitor hazardous environments (e.g., for gas leaks, high temperatures) 24/7. Robots can be deployed to perform dangerous or ergonomically difficult tasks, reducing workplace injuries.
 - **Energy Optimization:** By monitoring energy consumption in real-time, the factory can intelligently power down idle machines and optimize lighting and HVAC systems, significantly reducing energy costs and a firm's carbon footprint.
4. **Greater Flexibility and Agility:**
 - **Mass Customization:** A flexible, modular smart factory can quickly and cheaply reconfigure itself to produce small batches of customized products, meeting modern consumer demands.
 - **Resilient Supply Chains:** With real-time visibility into inventory and production, companies can respond instantly to supply chain disruptions or sudden spikes in demand.

Project Objectives

The primary objective of this project is to design, analyze, and model a proof-of-concept, IoT-based Smart Factory system using accessible, open-source hardware and software components. This system will serve as a modular "node" that can be deployed to a specific workstation, storage room, or machine to provide intelligent monitoring and control.

The specific objectives are:

1. **To design the hardware architecture** of a multi-functional monitoring system integrating environmental, presence, and process sensors with actuators like relays, alarms, and displays.
2. **To perform a complete circuit and component analysis**, detailing the role and industrial application of each sensor and actuator within a smart factory context.
3. **To model the software methodology and logic** required to integrate sensor data and trigger intelligent, automated responses. This includes defining the logic for safety, energy management, and process automation scenarios.
4. **To define the IoT integration framework**, explaining how the system would connect to a cloud platform (via MQTT) to enable a remote dashboard, real-time data logging, and automated alerts for management.
5. **To explore the practical applications, uses, and scalability** of this system, demonstrating its value as a low-cost entry point for SMEs into the world of Industry 4.0.

METHODOLOGY OF SMART FACTORY SYSTEM

This model involves building a functional, small-scale version of the system to demonstrate the concept, validate the design logic, and test the integration of various hardware and software components.

The methodology is broken down into three primary phases:

1. **Phase 1: System Design and Component Selection**
2. **Phase 2: Hardware Implementation and Software Development (implemented)**
3. **Phase 3: IoT Integration and System Validation (implemented)**

Phase : System Design and Component Selection

This initial phase focused on translating the project's high-level objectives (safety, efficiency, automation) into a tangible system architecture and a specific bill of materials.

System Architecture Design

A layered architecture was chosen to ensure the system is modular and scalable. This architecture consists of four distinct layers:

1. **Perception Layer (Sensors):** This is the lowest layer, responsible for interacting with the physical environment and collecting raw data. This includes all the sensors in the circuit (PIR, Ultrasonic, Gas).
2. **Processing Layer (Controller):** This is the "brain" of the local node. It aggregates data from the Perception Layer, executes the core logic, and makes real-time decisions. The Arduino Uno serves this function.
3. **Actuation Layer (Actuators):** This layer acts upon the physical environment based on commands from the Processing Layer. This includes the relays, lights, buzzer, fan, and local display.
4. **Network & Application Layer (IoT):** This is the highest layer, responsible for transmitting data to the cloud, receiving remote commands, and providing a dashboard for remote monitoring and data analysis.

Component Selection and Rationale

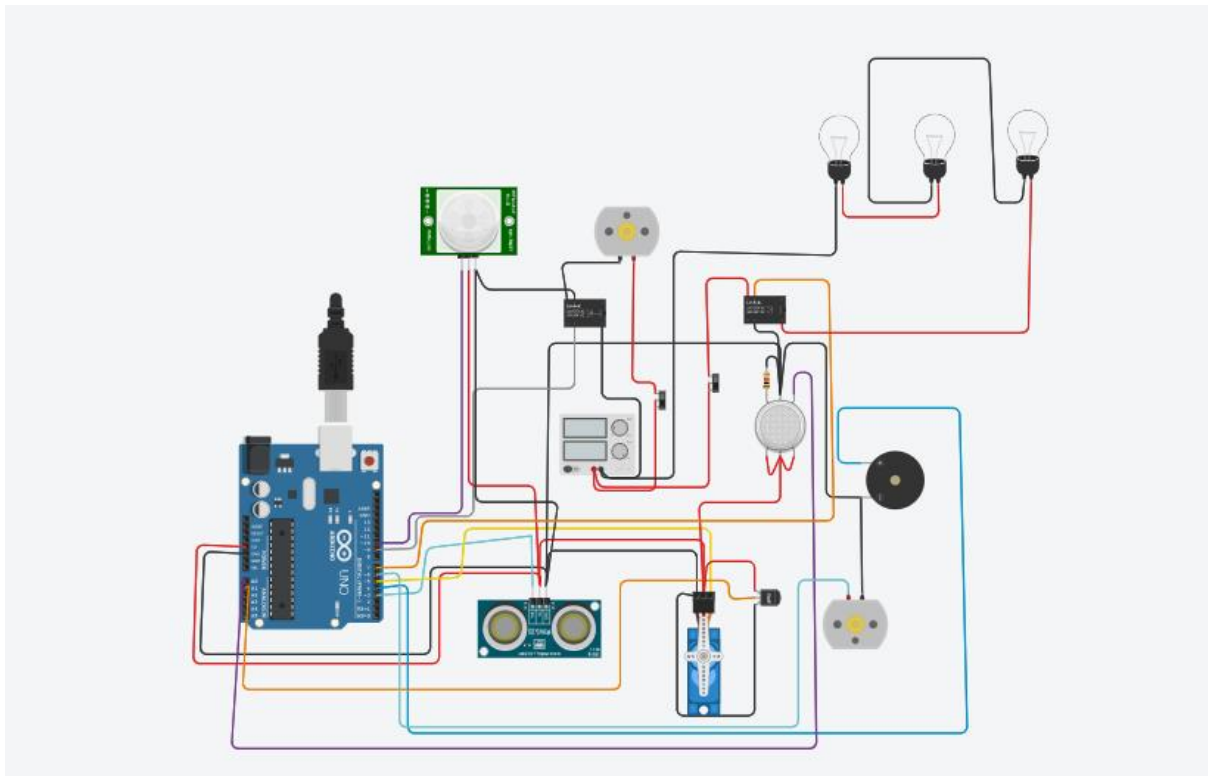
The components shown in the circuit diagram were selected based on their low cost, high availability, extensive documentation, and suitability for proving the core concepts of a smart factory.

- **Central Controller (Arduino Uno):** The Arduino Uno was chosen as the central processing unit for this prototype. Its ATmega328P microcontroller has sufficient digital and analog I/O pins to manage all the sensors and actuators in this design. The vast open-source community, simple C/C++ based Arduino IDE, and abundant libraries make it the ideal platform for rapid prototyping.
- **PIR Motion Sensor (e.g., HC-SR501):** Selected for the "Energy Management" and "Security" functions. This sensor provides a simple, binary HIGH/LOW digital output, making it extremely easy to integrate. Its low power consumption and adjustable sensitivity are perfect for detecting human presence to control lighting or trigger security alerts.
- **Ultrasonic Sensor (e.g., HC-SR04):** Selected for the "Process Automation" function. This sensor's ability to provide non-contact distance measurements makes it highly versatile. It was chosen to simulate one of two key industrial processes:
 - **Inventory Management:** By mounting it above a "tank" or "silo," it can measure the fill level of raw materials (liquids or solids).
 - **Production Counting:** By mounting it on the side of a "conveyor," it can detect and count products as they pass. Its Trig/Echo pin mechanism provides precise distance data to be calculated by the Arduino.
- **Gas/Smoke Sensor (e.g., MQ-2):** Selected for the critical "Workplace Safety" function. An MQ-series sensor was chosen because it provides an *analog* output voltage that is proportional to the concentration of a target gas (e.g., smoke, LPG, methane). This analog signal is crucial, as it allows the software to define multiple thresholds (e.g., a low-level "Warning" and a high-level "Danger") rather than a simple on/off alarm.
- **Relay Modules (5V):** These are the most critical components for real-world actuation. Relays were selected because they are electromechanical switches that allow the Arduino's low-voltage (5V) DC logic to safely control high-voltage (120V/240V AC) industrial equipment like overhead lights, high-power ventilation

fans, or the main power to machinery. This provides the necessary electrical isolation between the delicate microcontroller and the high-power factory environment.

- **Actuators (Lights, Fan, Buzzer):**

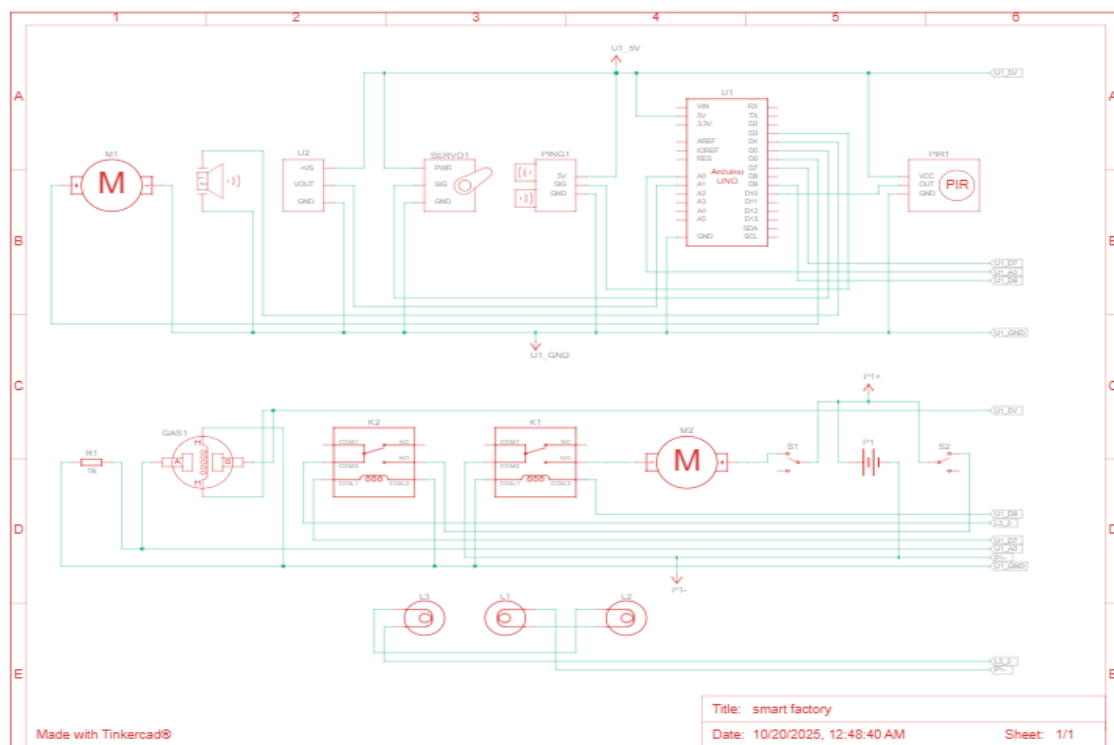
- **Lights (AC):** Represent the factory's main lighting system, controlled by a relay for smart energy management. They can also serve as an "Andon" status light.
- **Fan (DC):** Represents a ventilation/exhaust fan, controlled by a relay and linked to the gas sensor for automated safety.
- **Buzzer:** Provides an immediate, on-site audible alarm for critical events, ensuring workers are alerted even without access to the dashboard.
- **Digital Display (e.g., 7-Segment):** This component was selected to provide a "local dashboard" for on-site workers, displaying critical information like product count or gas levels directly at the workstation.



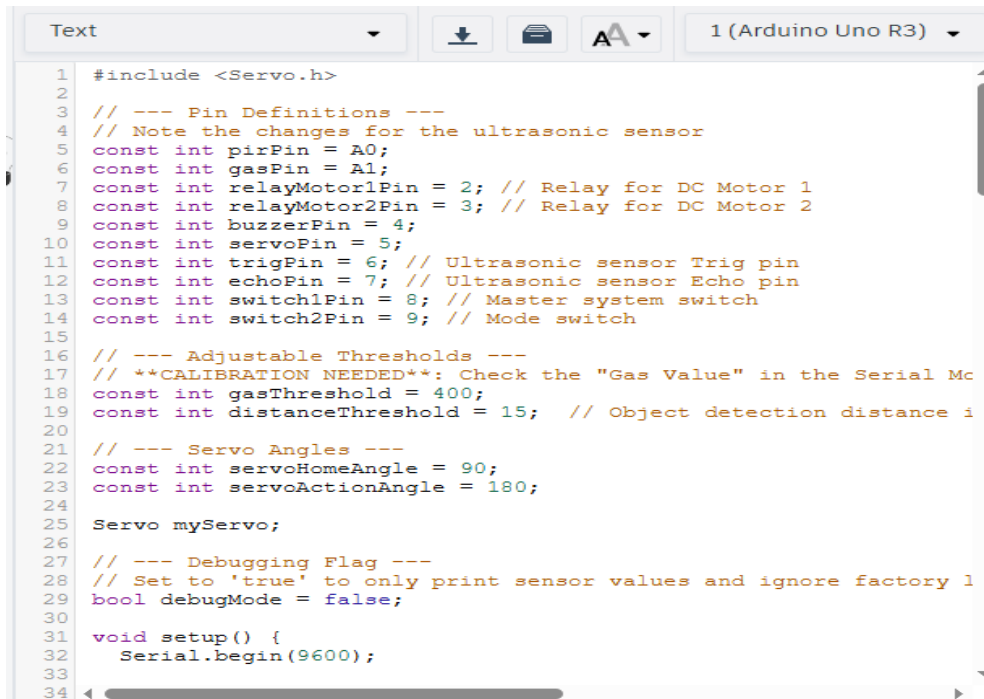
(i) The circuit diagram of the smart factory system

| Name | Quantity | Component |
|----------------|----------|----------------------------|
| U1 | 1 | Arduino Uno R3 |
| PING1 | 1 | Ultrasonic Distance Sensor |
| SERV01 | 1 | Continuous Micro Servo |
| GAS1 | 1 | Gas Sensor |
| PIEZ01 | 1 | Piezo |
| M1 M2 | 2 | DC Motor |
| U2 | 1 | Temperature Sensor [TMP36] |
| R1 | 1 | 1 kΩ Resistor |
| K1 K2 | 2 | Relay SPDT |
| PIR1 | 1 | PIR Sensor |
| P1 | 1 | 5, 5 Power Supply |
| S1 S2 | 2 | Slideswitch |
| L1 L2 L3 | 3 | Light bulb |

(ii) The components list of smart factory system



(iii) The schematic view of the smart factory system



```
1 #include <Servo.h>
2
3 // --- Pin Definitions ---
4 // Note the changes for the ultrasonic sensor
5 const int pirPin = A0;
6 const int gasPin = A1;
7 const int relayMotor1Pin = 2; // Relay for DC Motor 1
8 const int relayMotor2Pin = 3; // Relay for DC Motor 2
9 const int buzzerPin = 4;
10 const int servoPin = 5;
11 const int trigPin = 6; // Ultrasonic sensor Trig pin
12 const int echoPin = 7; // Ultrasonic sensor Echo pin
13 const int switch1Pin = 8; // Master system switch
14 const int switch2Pin = 9; // Mode switch
15
16 // --- Adjustable Thresholds ---
17 // **CALIBRATION NEEDED**: Check the "Gas Value" in the Serial Monitor
18 const int gasThreshold = 400;
19 const int distanceThreshold = 15; // Object detection distance in cm
20
21 // --- Servo Angles ---
22 const int servoHomeAngle = 90;
23 const int servoActionAngle = 180;
24
25 Servo myServo;
26
27 // --- Debugging Flag ---
28 // Set to 'true' to only print sensor values and ignore factory logic
29 bool debugMode = false;
30
31 void setup() {
32     Serial.begin(9600);
33 }
34
```

(iv) the code terminal of smart factory system

The code used for stimulation of the project- smart factory system:

```
#include <Servo.h>
// --- Pin Definitions ---
// Note the changes for the ultrasonic sensor
const int pirPin = A0;
const int gasPin = A1;
const int relayMotor1Pin = 2; // Relay for DC Motor 1
const int relayMotor2Pin = 3; // Relay for DC Motor 2
const int buzzerPin = 4;
const int servoPin = 5;
const int trigPin = 6; // Ultrasonic sensor Trig pin
const int echoPin = 7; // Ultrasonic sensor Echo pin
const int switch1Pin = 8; // Master system switch
const int switch2Pin = 9; // Mode switch

// --- Adjustable Thresholds ---
// **CALIBRATION NEEDED**: Check the "Gas Value" in the Serial Monitor in clean air
// and set this just above that value.
const int gasThreshold = 400;
const int distanceThreshold = 15; // Object detection distance in cm

// --- Servo Angles ---
```

```

const int servoHomeAngle = 90;
const int servoActionAngle = 180;

Servo myServo;

// --- Debugging Flag ---
// Set to 'true' to only print sensor values and ignore factory logic. Helps with calibration.
bool debugMode = false;

void setup() {
  Serial.begin(9600);

  // Set pin modes
  pinMode(pirPin, INPUT);
  pinMode(gasPin, INPUT);
  pinMode(switch1Pin, INPUT_PULLUP);
  pinMode(switch2Pin, INPUT_PULLUP);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  pinMode(relayMotor1Pin, OUTPUT);
  pinMode(relayMotor2Pin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  myServo.attach(servoPin);
  myServo.write(servoHomeAngle);

  // Set initial relay state (Motors OFF). Assumes LOW-trigger relays.
  digitalWrite(relayMotor1Pin, HIGH);
  digitalWrite(relayMotor2Pin, HIGH);

  Serial.println("Smart Factory System Initializing...");
  Serial.println("Allowing 20 seconds for gas sensor to warm up...");
  delay(20000); // **IMPORTANT**: MQ-series gas sensors need time to heat up for
  accurate readings.

  if(debugMode) {
    Serial.println("\n*** DEBUG MODE IS ON ***");
    Serial.println("The system will only print sensor readings.\n");
  } else {
    Serial.println("Warm-up complete. System is active.");
  }
}

```

```

void loop() {
  if (debugMode) {
    runDiagnostics(); // Run the debugging function and skip all other logic
    return;
  }

  // 1. --- Safety Check (Highest Priority) ---
  int gasValue = analogRead(gasPin);
  Serial.print("Gas Value: ");
  Serial.println(gasValue);

  if (gasValue > gasThreshold) {
    Serial.println("!!! GAS ALERT !!!");
    triggerAlarm();
    return;
  } else {
    noTone(buzzerPin);
  }

  // 2. --- System Operation ---
  if (digitalRead(switch1Pin) == LOW) { // System is ON
    int pirState = digitalRead(pirPin);
    long distance = getUltrasonicDistance_HCSR04();

    Serial.print("PIR: ");
    Serial.print(pirState);
    Serial.print(" | Distance: ");
    Serial.print(distance);
    Serial.println(" cm");

    if (distance > 0 && distance < distanceThreshold) {
      Serial.println("Item detected. Processing...");
      digitalWrite(relayMotor1Pin, HIGH); // Stop conveyor
      myServo.write(servoActionAngle);
      delay(1000);
      digitalWrite(relayMotor2Pin, LOW); // Start processing motor
      delay(3000);
      digitalWrite(relayMotor2Pin, HIGH); // Stop processing motor
      myServo.write(servoHomeAngle);
      delay(1000);
    } else if (pirState == HIGH) {
      Serial.println("Motion detected. Conveyor running.");
      digitalWrite(relayMotor1Pin, LOW); // Start conveyor
    }
  }
}

```

```

    } else {
        digitalWrite(relayMotor1Pin, HIGH); // Stop conveyor
    }

    } else { // System is OFF
        shutdownSystem();
        // Add a small delay in the off-state to prevent spamming the serial monitor
        delay(1000);
    }

    delay(200);
}

// --- Utility Functions ---

long getUltrasonicDistance_HCSR04() {
    // Clears the trigPin
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    long duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    // Speed of sound wave divided by 2 (go and back)
    return duration * 0.034 / 2;
}

void triggerAlarm() {
    tone(buzzerPin, 1000, 500);
    digitalWrite(relayMotor1Pin, HIGH);
    digitalWrite(relayMotor2Pin, HIGH);
}

void shutdownSystem() {
    digitalWrite(relayMotor1Pin, HIGH);
    digitalWrite(relayMotor2Pin, HIGH);
    myServo.write(servoHomeAngle);
    noTone(buzzerPin);
}

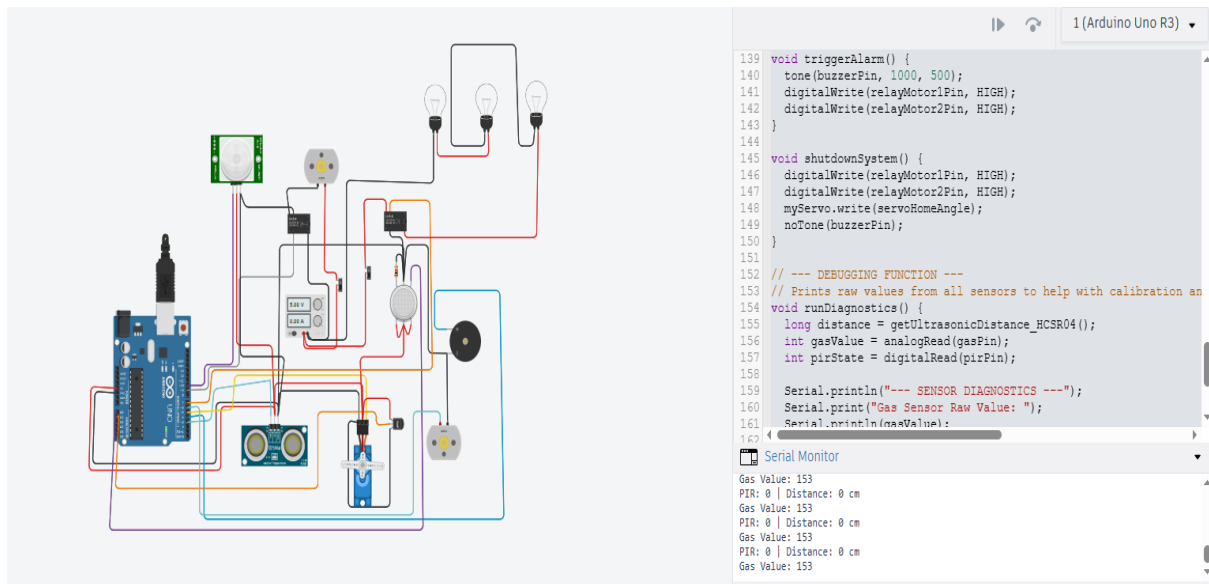
// --- DEBUGGING FUNCTION ---

```

// Prints raw values from all sensors to help with calibration and troubleshooting.

```
void runDiagnostics() {  
  long distance = getUltrasonicDistance_HCSR04();  
  int gasValue = analogRead(gasPin);  
  int pirState = digitalRead(pirPin);  
  
  Serial.println("--- SENSOR DIAGNOSTICS ---");  
  Serial.print("Gas Sensor Raw Value: ");  
  Serial.println(gasValue);  
  Serial.print("Ultrasonic Distance (cm): ");  
  Serial.println(distance);  
  Serial.print("PIR Motion State (0/1): ");  
  Serial.println(pirState);  
  Serial.println("-----\n");  
  
  delay(1000); // Print values every second  
}
```

Output of the stimulation:



TASK 1: Master-Slave Architecture

Overview of Master-Slave Architecture

The Master-Slave architecture is a fundamental design pattern in distributed systems, where a central "Master" node controls and coordinates several "Slave" nodes. The Master typically issues commands, collects data, and manages the overall system, while the Slaves execute specific tasks and report their status or data back to the Master. This hierarchical structure offers clear control, simplifies system management, and is particularly well-suited for industrial automation and sensor networks where centralized coordination is beneficial.

Application in an IoT-Based Smart Factory

For our IoT-based Smart Factory system, a Master-Slave architecture provides an effective way to scale the prototype from a single unit to cover multiple distinct areas or "zones" within a factory. Instead of each zone independently connecting to a remote cloud, a Master node can oversee a cluster of these zones, providing an intelligent intermediate layer.

Consider a factory with several operational zones, such as:

- **Zone A:** Chemical Storage (gas detection, temperature monitoring).
- **Zone B:** Assembly Line (product counting, machine status).
- **Zone C:** Material Handling (tank level monitoring, conveyor control).

Each of these zones can be managed by a dedicated **Slave Node**, while a central **Master Node** coordinates these Slaves and acts as a gateway to the broader IoT cloud platform.

Components of the Architecture

Slave Nodes (Zone-Specific Units)

Each of our previously designed IoT Smart Factory prototypes would serve as a Slave node.

- **Hardware:** An Arduino (or ESP32) microcontroller, equipped with zone-specific sensors (PIR, Ultrasonic, Gas) and actuators (Relays for lights/fans, Buzzer, 7-segment display). Each Slave would also include a communication module (e.g., an RS-485 transceiver for Modbus RTU) to interface with the Master.

- **Functionality:**
 - **Local Sensing:** Continuously collects real-time data from its designated zone.
 - **Local Actuation:** Executes immediate, time-critical responses autonomously (e.g., activating a fan upon gas detection, sounding a local alarm for intrusion) without waiting for Master or cloud commands.
 - **Data Aggregation:** Packages its sensor data for transmission to the Master.
 - **Communication:** Responds to requests from the Master, sending data and receiving commands.

Master Node (Zonal Coordinator/Gateway)

The Master node is a more powerful computing unit designed to oversee multiple Slaves.

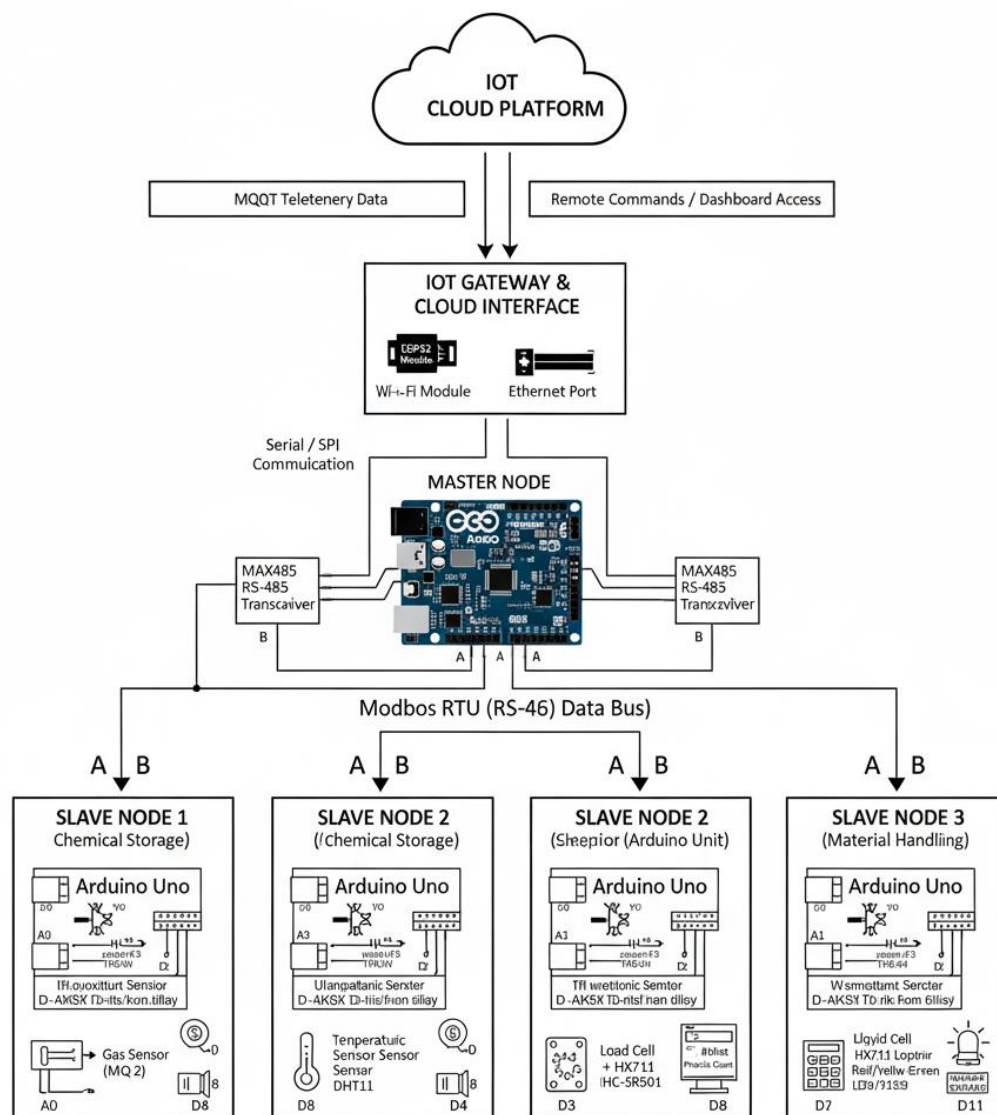
- **Hardware:** Typically a more capable microcontroller or a Single-Board Computer (SBC) like a Raspberry Pi or an advanced ESP32 board. It includes robust communication interfaces (e.g., Ethernet, Wi-Fi, RS-485 transceiver) and potentially a local display.
- **Functionality:**
 - **Polling Slaves:** Actively requests and collects data from all connected Slave nodes.
 - **Gateway to Cloud:** Serves as the single point of entry for its cluster of Slaves to the IoT cloud platform. It uploads consolidated data and manages cloud communication, reducing the network load and complexity for the individual Slaves.

Communication and Data Flow

- **Master-Slave Communication:** A reliable industrial protocol like **Modbus RTU over RS-485** is ideal for communication between the Master and its Slaves. The Master sends polling requests (e.g., "read gas value from Slave 1"), and Slaves respond with their data.
- **Local Autonomy:** Slaves retain the ability to act immediately on critical local events, ensuring safety even if the Master or cloud connection is temporarily unavailable.

Benefits of this Architecture

- **Scalability:** Easily add more Slave nodes (new factory zones) without reconfiguring the entire cloud setup.
- **Centralized Oversight:** A single Master provides a unified view and control point for a cluster of factory zones.



TASK 2: Operating Systems and Multithreading Concept

Operating Systems (OS): The Manager of Computer Resources

An Operating System (OS) is the most critical software that runs on a computer. It's the master control program that manages all the computer's hardware and software resources. Without an OS, a computer is essentially a useless collection of electronic components.

Core Functions of an Operating System

The OS acts as an intermediary between the user/applications and the computer hardware, performing several crucial functions:

- **Process Management:**
 - **What is a Process?** A process is an instance of a computer program that is being executed. It's an active entity that requires system resources (CPU time, memory, I/O devices) to complete its task. Examples include your web browser, a word processor, or a background antivirus scan.
 - **OS Role:** The OS is responsible for creating, scheduling, terminating, and synchronizing processes. It allocates CPU time to processes (CPU scheduling), manages their states (running, waiting, ready), and ensures that processes don't interfere with each other.
- **Memory Management:**
 - **OS Role:** The OS manages the computer's main memory (RAM). It keeps track of which parts of memory are being used by which processes, allocates memory space when processes need it, and deallocates it when processes terminate. It also ensures that processes cannot access each other's memory, providing protection. Techniques like paging and swapping are used to efficiently utilize memory and give the illusion of more memory than physically available.
- **File System Management:**
 - **OS Role:** The OS organizes and manages files and directories on storage devices (hard drives, SSDs). It provides a logical view of storage, handles file creation, deletion, access, and permissions, ensuring data integrity and security.

- **Device Management (I/O System):**
 - **OS Role:** The OS manages all input/output (I/O) devices, such as keyboards, mice, printers, network cards, and monitors. It translates user/application requests into device-specific commands, handles device drivers, and buffers I/O operations to improve efficiency.
- **Security and Protection:**
 - **OS Role:** The OS protects system resources from unauthorized access and prevents malicious programs from harming the system. It implements user authentication, access control lists (ACLs) for files and resources, and memory protection mechanisms.
- **User Interface:**
 - **OS Role:** Provides a way for users to interact with the computer. This can be a Graphical User Interface (GUI) with windows, icons, menus, and pointers (like Windows, macOS), or a Command Line Interface (CLI) where users type commands (like Linux shells).

Types of Operating Systems

OS come in various forms, tailored for different purposes:

- **Batch OS:** Processes jobs in batches without direct user interaction (early mainframes).
- **Time-Sharing OS:** Allows multiple users to share a computer simultaneously by giving each user a slice of CPU time (e.g., UNIX, Linux servers).
- **Real-Time OS (RTOS):** Guarantees that critical operations complete within a specified deadline, crucial for industrial control, medical devices, and embedded systems.
- **Distributed OS:** Manages a group of independent computers and makes them appear as a single coherent system.
- **Network OS:** Runs on a server and allows client machines to access shared files and resources.
- **Embedded OS:** Designed for specific embedded systems (e.g., IoT devices, smart appliances).

Multithreading Concept: Concurrency within a Process

While an OS manages multiple *processes*, **multithreading** deals with running multiple parts of a single *process* concurrently. It's a way to achieve parallel execution within a single program, leading to better resource utilization and responsiveness.

Processes vs. Threads

To understand multithreading, it's essential to differentiate between processes and threads:

| Feature | Process | Thread |
|-----------------|--|--|
| Definition | An executing instance of a program. | A single sequence of execution within a process. |
| Independence | Independent; has its own memory space, resources. | Part of a process; shares its parent's memory/resources. |
| Overhead | High overhead (creation, context switching). | Low overhead (creation, context switching). |
| Resource Share | No (unless explicitly shared via IPC). | Yes, shares code, data, files of its parent process. |
| Fault Isolation | High; if one process crashes, others are unaffected. | Low; if one thread crashes, the entire process crashes. |

A thread is often called a "lightweight process" or a "unit of execution." Multiple threads can exist within a single process, sharing the process's resources (like memory space, open files, and global variables) but executing independently.²⁷ Each thread has its own program counter, stack, and set of registers.

When a program is multithreaded, it can divide its workload into smaller, independent parts that can be executed simultaneously.

- **Concurrency:** On a single-core CPU, the OS scheduler rapidly switches between threads (context switching), giving the *illusion* of simultaneous execution.³⁰ This is called **concurrency**.
- **Parallelism:** On a multi-core CPU, different threads can truly execute at the exact same time on different cores.³¹ This is called **parallelism**.

Benefits of Multithreading

- **Increased Responsiveness:** A multithreaded application can remain responsive to user input while performing long-running background tasks.³² For example, a word processor can check spelling in one thread while the user types in another.
- **Resource Sharing:** Threads within the same process share memory and resources, making inter-thread communication more efficient than inter-process communication (IPC).
- **Economy:** Creating and managing threads is generally less resource-intensive (lower overhead) than creating and managing processes.
- **Utilization of Multi-core Processors:** Multithreading allows programs to fully leverage the power of modern multi-core CPUs, achieving true parallel execution and speeding up computation for suitable tasks.
- **Better Program Structure:** Complex applications can be structured into simpler, concurrent tasks, improving code readability and maintainability.

Challenges of Multithreading

While powerful, multithreading introduces complexities:

- **Synchronization Issues:** Since threads share resources, uncontrolled access can lead to race conditions (where the outcome depends on the unpredictable timing of multiple threads) and data corruption.³⁸ Mechanisms like mutexes, semaphores, and locks are required to synchronize access to shared data.
- **Deadlocks:** Threads can get stuck waiting for each other to release resources, leading to a standstill.
- **Complexity:** Designing, debugging, and testing multithreaded applications is significantly more challenging than single-threaded ones.

CONCLUSION

The journey through the design, analysis, and architectural considerations of the IoT-Based Smart Factory System underscores a pivotal shift in industrial operations driven by the Fourth Industrial Revolution. This project has successfully conceptualized a modular, cost-effective, and scalable solution aimed at empowering small-to-medium-sized enterprises (SMEs) to embark on their Industry 4.0 transformation without the prohibitive investments typically associated with large-scale industrial digitization. By integrating a suite of readily available sensors and actuators with a robust communication architecture, the proposed system offers a practical blueprint for enhancing safety, optimizing efficiency, and enabling remote oversight within industrial environments.

Our detailed analysis of the circuit diagram revealed a sophisticated yet approachable hardware architecture, with the Arduino microcontroller serving as the central processing unit. The careful selection of components – including PIR sensors for presence detection, ultrasonic sensors for process monitoring (such as tank levels or product counting), and MQ-series gas sensors for critical environmental safety – forms the perception layer, enabling the system to gather diverse real-time data from the factory floor. These inputs are coupled with powerful actuation capabilities, primarily through relay modules that control high-voltage industrial equipment like lighting and ventilation systems, supplemented by local alarms (buzzers) and digital displays for immediate, on-site feedback. The integration of these elements ensures that the system is not merely a data collector but an active, responsive agent within the factory environment.

The methodology developed articulates a clear path from conceptual design to functional implementation. It emphasizes a structured software approach, using Arduino's versatile programming environment to create event-driven logic that intelligently processes sensor data and triggers appropriate automated responses. The core logic scenarios, such as automated gas leak detection with tiered warnings and emergency ventilation control, smart lighting based on occupancy, and real-time inventory monitoring, demonstrate the system's ability to automate critical tasks that historically required manual intervention. This automation not only reduces human error and operational costs but also significantly improves response times to critical events, thus elevating overall workplace safety and efficiency.

Crucially, the "IoT" dimension of this smart factory system is achieved through a well-defined integration framework. By incorporating an internet connectivity module and leveraging the lightweight MQTT protocol, the system transcends local automation, becoming a truly connected entity. This allows for real-time data telemetry to a cloud-based IoT platform, enabling remote managers to access dashboards, visualize operational data, receive instant alerts, and even issue remote commands to factory floor actuators. This remote oversight capability is transformative, allowing for proactive decision-making and ensuring business continuity regardless of physical location. The ability to collect and analyze historical data in the cloud further paves the way for advanced analytics, predictive maintenance, and continuous process optimization.

In essence, this project provides a comprehensive and practical example of how fundamental embedded systems engineering, coupled with robust IoT principles, can be leveraged to create intelligent industrial solutions. It highlights that Industry 4.0 is not an exclusive domain for large corporations but is increasingly accessible through modular, open-source technologies. The proposed IoT-based Smart Factory System, while a prototype, demonstrates the immense potential to enhance safety protocols, streamline production processes, manage energy consumption efficiently, and provide unprecedented visibility into factory operations. Its adoption can lead to reduced operational costs, improved productivity, and a safer working environment, positioning SMEs competitively in the evolving landscape of global manufacturing. The foundation laid here encourages further expansion with more advanced sensors, AI-driven analytics, and tighter integration with existing enterprise systems, paving the way for truly autonomous and self-optimizing smart factories of the future.