# 23ECE211 Microcontrollers and Interfacing

A Project Report on

# Smart Helmet for Workers

Submitted by:


| | |
|---|---|
| **AATHITHYA PRANAV** | **CB.EN.U4ECE23002** |
| **ADARSH B** | **CB.EN.U4ECE23004** |
| **BATTULA UDAY VENUGOPAL** | **CB.EN.U4ECE23010** |
| **EZHILKIRTHIK M** | **CB.EN.U4ECE23016** |

Department of Electronics and Communication Engineering,

Amrita School of Engineering,

Amrita Vishwa Vidyapeetham,

Coimbatore, India – 641112

# Contents

## Introduction

The rapid evolution of industrial operations has significantly heightened the demand for advanced safety solutions to protect workers in hazardous environments. Traditional Personal Protective Equipment (PPE), such as helmets, has long served as a fundamental safeguard against physical injuries. However, these conventional systems often fall short in addressing dynamic risks, such as exposure to toxic gases, extreme temperatures, or physiological distress, due to their inability to provide real-time monitoring or proactive alerts. This limitation underscores the need for innovative technologies that enhance worker safety beyond mere physical protection.

The proposed smart helmet system represents a transformative approach to industrial safety by integrating cutting-edge embedded technology into traditional PPE. Designed around the LPC2148 microcontroller, this system employs Analog-to-Digital Converter (ADC)-based temperature and gas sensors to continuously monitor the worker's physiological state and surrounding environmental conditions. Coupled with a GSM module interfaced via UART for SMS-based alerts, the helmet enables real-time detection and communication of critical anomalies, such as elevated temperatures or hazardous gas levels, to designated personnel. This fusion of sensor technology, data processing, and wireless communication aims to mitigate risks, facilitate timely interventions, and ultimately reduce workplace accidents and health-related incidents. By transitioning from reactive to proactive safety measures, the smart helmet exemplifies the potential of data-driven solutions in fostering safer and more efficient industrial workplaces.

## Motivation

Industrial environments are inherently fraught with risks, ranging from exposure to toxic gases like carbon monoxide (CO) and hydrogen sulphide (H2S) to extreme temperatures that can lead to heatstroke or hypothermia. According to occupational safety statistics, workplace accidents remain a persistent challenge, claiming numerous lives and causing substantial economic losses annually due to downtime, medical costs, and compensation. Traditional PPE, while essential, is largely passive and lacks the capability to detect or respond to invisible yet life-threatening hazards in real time. This gap in safety infrastructure motivates the development of advanced systems capable of pre-empting dangers before they escalate into emergencies.

The motivation for this project stems from the urgent need to enhance worker safety and operational efficiency through technology-driven solutions. The advent of affordable microcontrollers, such as the LPC2148, and the availability of reliable sensors and GSM communication modules present an opportunity to revolutionize PPE. By equipping industrial workers with a smart helmet that actively monitors vital parameters and environmental conditions, this system promises to empower both workers and supervisors with actionable insights. The potential to prevent accidents, reduce response times in emergencies, and

improve overall well-being in high-risk settings provides a compelling impetus for this innovation. Furthermore, the scalability and adaptability of this technology could set a new standard for safety across diverse industries, from mining and manufacturing to construction.

## Problem Statement

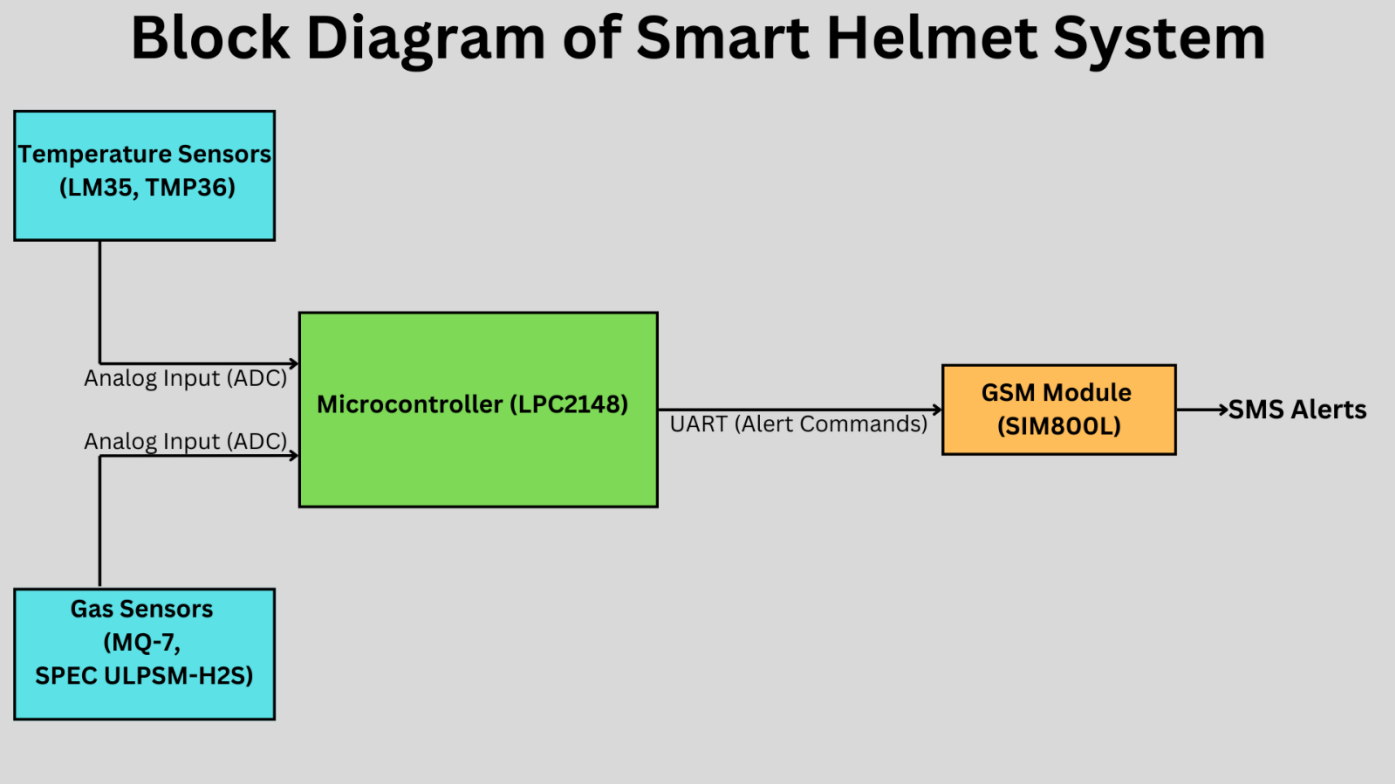
Despite advancements in industrial safety protocols, workers in hazardous environments continue to face significant risks due to the limitations of conventional PPE. Traditional helmets excel at providing physical protection against impacts but are ill-equipped to address critical health and environmental threats, such as gas leaks, extreme temperatures, or early signs of physiological distress like fatigue or heat stress. The absence of real-time monitoring and alert mechanisms in these systems often results in delayed detection of hazards, leaving little room for preventive action. Consequently, industrial workers remain vulnerable to accidents and health issues that could be mitigated with timely intervention.

The core problem this project seeks to address is the lack of an integrated, proactive safety system capable of monitoring and responding to dynamic workplace hazards. Specifically, there is a need for a solution that can:

- Continuously track the worker's body temperature and the presence of hazardous gases
- Process this data in real time to identify anomalies
- Instantly notify supervisors or emergency responders when predefined safety thresholds are breached

Current systems either lack such capabilities or are prohibitively expensive and complex for widespread adoption. This project proposes a cost-effective, reliable smart helmet system using the LPC2148 microcontroller, ADC-based sensors, and GSM-based SMS alerts to bridge this gap, ensuring enhanced safety and rapid response in industrial settings.

## 4. Block Diagram:




## 5. Components and Software Requirements:

### Components:

1. **Microcontroller**: The LPC2148, a 32-bit ARM7TDMI-S with built-in ADC, seems suitable for processing sensor data.
2. **Temperature Sensors**: LM35 and TMP36 are common, providing analog outputs for ADC conversion, ideal for monitoring worker safety.
3. **Gas Sensors**: MQ-7 for carbon monoxide and SPEC ULPSM-H2S for hydrogen sulfide are likely, both ADC-based for detecting hazardous gases.
4. **GSM Module**: SIM800L is typically used for sending SMS alerts via UART, ensuring workers get timely notifications.
5. **Power and Accessories**: Stable power sources (5V, 3.3V, 3.7V-4.2V), voltage regulators, and rugged enclosures for durability in industrial settings.

### Software:

1. **Keil µVision 4:** Programming the LPC2148 in embedded C, compiling, debugging, simulation.
2. **Proteus 8 Professional:** Circuit simulation and design verification.

## 6.Working Procedure:

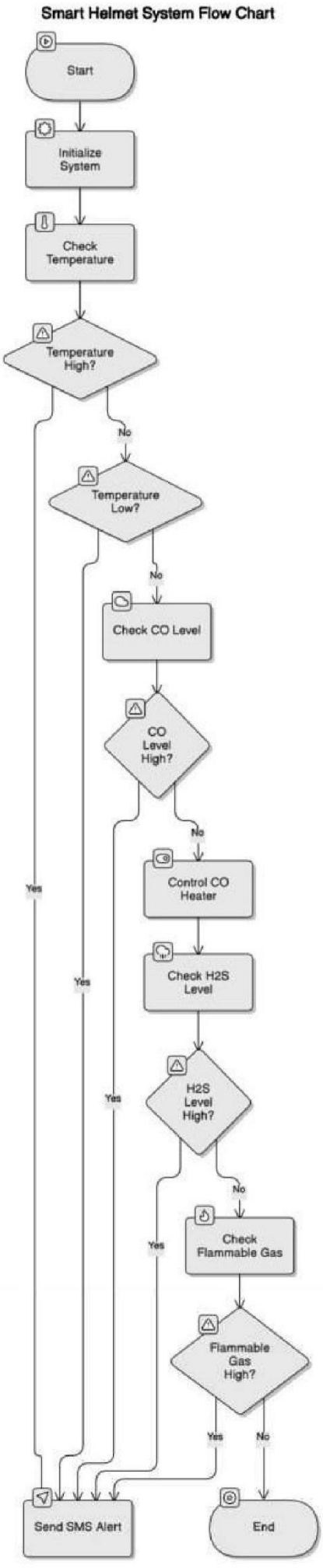
### 1. System Initialization

- Power-Up and Configuration: The process begins with powering on the LPC2148 microcontroller, the central processing unit of the smart helmet system. The microcontroller initializes its peripherals, including:
    - ADC Modules: The LPC2148 features two 10-bit ADC modules (ADC0 with 6 channels and ADC1 with 8 channels), configured with a reference voltage of 3.3V. The ADC clock is set via the CLKDIV bits in the AD0CR register to operate at ≤ 4.5 MHz, and the Power Down (PDN) bit is enabled.
    - UART: The UART (UART0 or UART1) is initialized at a baud rate (e.g., 9600 or 115200 bps) for communication with the GSM module, with settings for data bits, parity, and stop bits configured via the U0LCR/U1LCR register.
    - GPIO Pins: General Purpose I/O pins are set up to interface with sensors and control additional functions (e.g., gas sensor heater cycles).

**Sensor Setup:**

- Temperature Sensor: An analog sensor like the LM35 (10 mV/°C output) or TMP36 is connected to an ADC channel (e.g., AD0.1 on pin P0.28). The LM35 requires a 5V supply, while the TMP36 operates at 3.3V, aligning with the LPC2148's VREF.
- Gas Sensors:
    - MQ-7 (CO): Connected to AD0.2, with a voltage divider scaling its 0-5V output to 0-3.3V. Its heater cycle (5V for 60s, 1.4V for 90s) is managed via GPIO pins and timers.
    - SPEC ULPSM-H2S (H2S): Connected to AD0.3, with a 0-3V output directly compatible with the ADC.
    - MQ-2 (Flammable Gases): Connected to AD0.4, scaled similarly to the MQ-7 if needed.

**GSM Module Initialization**: The SIM800L GSM module is powered with a 3.7V-4.2V supply (capable of 2A peak current). Its RXD and TXD pins connect to the LPC2148's UART pins, with a voltage divider on the TXD line to match the SIM800L's 2.8V logic level. A SIM card and antenna are installed.



**2. Check Temperature**

- **Data Acquisition**: The system monitors the worker's physiological state using the temperature sensor. The LM35's analog output is fed to AD0.1. The ADC is configured by setting PINSEL1 (bits 24-25 to '01'), enabling the channel in AD0CR, and triggering conversion with the START bit. The 10-bit result (0-1023) is read from AD0DR1 when the DONE flag in AD0GDR is set.
- **Conversion**: The temperature is calculated as:
    - For LM35: Temperature (°C) = ADC Reading * (3.3 / 1024) * 100.
    - For TMP36: Temperature (°C) = [(ADC Reading * (3.3 / 1024)) - 0.5] / 0.01, accounting for its 500mV offset.

**3. Temperature High Check**

- **Threshold Comparison**: The temperature is compared to an upper threshold (e.g., 38°C, per industrial safety standards). If exceeded (Yes), the system proceeds to send an SMS alert. If not (No), it checks the low threshold.

**4. Temperature Low Check**

- **Threshold Comparison**: The temperature is compared to a lower threshold (e.g., 15°C). If below this value (Yes), an SMS alert is triggered. If not (No), the system moves to gas monitoring, indicating a safe temperature range.

**5. Check CO Level**

- **Data Acquisition**: The MQ-7 sensor measures Carbon Monoxide (CO, 10-1000 ppm). Its analog output is scaled and fed to AD0.2. The ADC process mirrors the temperature sensor's, with the heater cycle controlled by GPIO pins toggling between 5V and 1.4V using timers.
- **Evaluation**: The raw ADC value is compared to a threshold (e.g., corresponding to 100 ppm), or converted to PPM using the sensor's Rs/Ro ratio and datasheet curves if calibrated.

**6. CO Level High Check**

- **Threshold Comparison**: If the CO level exceeds its threshold (Yes), a "Control CO" action (e.g., alert or mitigation) is initiated, followed by an SMS alert. If not (No), the system proceeds to the next gas check.

**7. Check H2S Level**

- **Data Acquisition**: The SPEC ULPSM-H2S sensor (0-50 ppm, 0-3V output) on AD0.3 provides an analog signal, converted by the ADC as described previously.
- **Evaluation**: The ADC value is compared to a threshold (e.g., 50 ppm) or converted to PPM based on the sensor's linear output.

**8. H2S Level High Check**

- **Threshold Comparison**: If the H2S level is high (Yes), an SMS alert is sent. If not (No), the system checks flammable gases.

**9. Check Flammable Gas**

- **Data Acquisition**: The MQ-2 sensor (200-10000 ppm) on AD0.4 detects flammable gases (e.g., Methane, Propane). Its output is scaled if necessary and converted via the ADC.
- **Evaluation**: The value is compared to a threshold indicating a hazard (e.g., 1000 ppm).

**10. Flammable Gas High Check**

- **Threshold Comparison**: If the flammable gas level is high (Yes), an SMS alert is triggered. If not (No), the system loops back to step 2 for continuous monitoring.

**11. Send SMS Alert**

- **Alert Generation**: When any threshold is exceeded (temperature, CO, H2S, or flammable gas), the LPC2148 formats a message (e.g., “High Temperature Alert: 40°C” or “CO Gas Detected: 150 ppm”).
- **UART Communication**: The UART sends AT commands to the SIM800L:
  - AT+CMGF=1: Sets text mode.
  - AT+CMGS="phone_number": Specifies the recipient.
  - Message text, followed by Ctrl+Z (ASCII 26), sends the SMS.
- **Confirmation**: The system awaits a response from the SIM800L to confirm transmission.

**12. Continuous Monitoring and End**

- **Loop Operation**: The process repeats, sampling sensors periodically (e.g., every few seconds, based on the ADC’s 2.44 µs conversion time and software timing).
- **Power Management**: The LPC2148 and sensors enter sleep modes when idle, and the SIM800L uses its low-power mode to optimize battery life. The process “ends” only if powered off, otherwise continuing indefinitely.

## 7. Keil uvision 4 Code:

```
#include <LPC214x.h>  // LPC2148 header file
#include <stdint.h>   // For uint16_t, uint32_t
#include <stdio.h>    // For sprintf

// Thresholds
#define TEMP_HIGH_THRESHOLD    38.0  // °C
#define TEMP_LOW_THRESHOLD     15.0  // °C
#define CO_THRESHOLD           50    // ppm (simplified)
#define H2S_THRESHOLD          20    // ppm (simplified)
#define FLAMMABLE_THRESHOLD    100   // ppm (simplified)

// UART0 Baud Rate (9600 @ 60MHz PCLK)
#define BAUD_RATE   9600
#define UART_DIV    ((60000000 / 16) / BAUD_RATE)

// ADC Channels
#define TEMP_SENSOR_CHANNEL    1  // AD0.1
#define CO_SENSOR_CHANNEL      2  // AD0.2
#define H2S_SENSOR_CHANNEL     3  // AD0.3
#define FLAMMABLE_SENSOR_CHANNEL 4  // AD0.4

// GPIO for LEDs (P1.16 - P1.20)
#define LED_TEMP_HIGH  (1 << 16)
#define LED_TEMP_LOW   (1 << 17)
#define LED_CO         (1 << 18)
#define LED_H2S        (1 << 19)
#define LED_FLAMMABLE  (1 << 20)

// MQ-7 Heater Pin
#define MQ7_HEATER_PIN (1 << 10)

// Function Prototypes
void init_PLL(void);
void init_UART0(void);
void init_ADC(void);
void init_GPIO(void);
void delay_ms(uint32_t ms);
float convert_temp(uint16_t adc_value);
uint16_t convert_CO(uint16_t adc_value);
uint16_t convert_H2S(uint16_t adc_value);
uint16_t convert_flammable(uint16_t adc_value);
void send_UART0(const char *str);
void send_SMS(const char *message);
```

```
void check_and_alert(float temp, uint16_t co, uint16_t h2s, uint16_t flammable);
uint16_t ADC_Read(uint16_t channel);
void control_MQ7_heater(void);

// Main Function
int main(void) {
    float temperature;
    uint16_t co_level, h2s_level, flammable_level;

    // Initialize system
    init_PLL();    // Set CPU to 60 MHz
    init_UART0();  // Initialize UART0 for GSM
    init_ADC();    // Initialize ADC
    init_GPIO();   // Initialize GPIO for LEDs and MQ-7 heater

    // Initial GSM setup
    send_UART0("AT\r\n");
    delay_ms(1000);
    send_UART0("AT+CMGF=1\r\n");
    delay_ms(1000);

    while (1) {
        // Read sensor data
        temperature = convert_temp(ADC_Read(TEMP_SENSOR_CHANNEL));
        co_level = convert_CO(ADC_Read(CO_SENSOR_CHANNEL));
        h2s_level = convert_H2S(ADC_Read(H2S_SENSOR_CHANNEL));
        flammable_level =
convert_flammable(ADC_Read(FLAMMABLE_SENSOR_CHANNEL));

        // Check thresholds and send alerts
        check_and_alert(temperature, co_level, h2s_level, flammable_level);

        delay_ms(5000); // Check every 5 seconds
    }
}

// PLL Initialization (60 MHz)
void init_PLL(void) {
    PLL0CON = 0x01;       // Enable PLL
    PLL0CFG = 0x24;       // M=5, P=2 (CCLK = 60 MHz, Fosc = 12 MHz)
    PLL0FEED = 0xAA;
    PLL0FEED = 0x55;
    while (!(PLL0STAT & 0x400)); // Wait for PLL lock
    PLL0CON = 0x03;       // Connect PLL
    PLL0FEED = 0xAA;
```

```
    PLL0FEED = 0x55;
    VPBDIV = 0x01;        // PCLK = CCLK (60 MHz)
}

// UART0 Initialization
void init_UART0(void) {
    PINSEL0 |= (1 << 0) | (1 << 2); // P0.0 TXD0, P0.1 RXD0
    U0LCR = 0x83;       // 8-bit, 1 stop, no parity, DLAB=1
    U0DLL = UART_DIV & 0xFF;
    U0DLM = (UART_DIV >> 8) & 0xFF;
    U0LCR = 0x03;       // DLAB=0, 8-bit data
}

// ADC Initialization
void init_ADC(void) {
    PINSEL1 |= (1 << 24) | (1 << 26) | (1 << 28) | (1 << 30); // AD0.1, AD0.2, AD0.3, AD0.4
    AD0CR = (1 << 21) | (11 << 8); // PDN = 1, CLKDIV = 11
}

// GPIO Initialization for LEDs and MQ-7 Heater
void init_GPIO(void) {
    IODIR1 |= LED_TEMP_HIGH | LED_TEMP_LOW | LED_CO | LED_H2S |
LED_FLAMMABLE; // LEDs as outputs
    IODIR0 |= MQ7_HEATER_PIN; // MQ-7 heater pin as output
    IO0SET = MQ7_HEATER_PIN;  // Initially high (5V for MQ-7)
}

// Simple Delay Function
void delay_ms(uint32_t ms) {
    uint32_t i, j;
    for (i = 0; i < ms; i++)
        for (j = 0; j < 6000; j++); // Calibrated for 60 MHz
}

// Convert ADC to Temperature (°C)
float convert_temp(uint16_t adc_value) {
    float voltage = (adc_value * 3.3) / 1024.0; // VREF = 3.3V
    return voltage * 100.0; // LM35: 10mV/°C
}

// Convert ADC to CO Level (Simplified PPM)
uint16_t convert_CO(uint16_t adc_value) {
    return (adc_value * 1000) / 1024; // Map to 0-1000 ppm
}
```

```
// Convert ADC to H2S Level (Simplified PPM)
uint16_t convert_H2S(uint16_t adc_value) {
    return (adc_value * 500) / 1024; // Map to 0-500 ppm
}

// Convert ADC to Flammable Gas Level (Simplified PPM)
uint16_t convert_flammable(uint16_t adc_value) {
    return (adc_value * 1000) / 1024; // Map to 0-1000 ppm
}

// Send String via UART0
void send_UART0(const char *str) {
    while (*str) {
        while (!(U0LSR & 0x20)); // Wait for THR empty
        U0THR = *str++;
    }
}

// Send SMS Alert
void send_SMS(const char *message) {
    char cmd[50];
    sprintf(cmd, "AT+CMGS=\"+918778459668\"\r\n"); // Replace with actual number
    send_UART0(cmd);
    delay_ms(500);
    send_UART0(message);
    delay_ms(500);
    send_UART0("\x1A\r\n"); // Ctrl+Z to send
    delay_ms(2000);
}

// Control MQ-7 Heater Cycle
void control_MQ7_heater(void) {
    static uint32_t cycle_counter = 0;
    cycle_counter++;
    if (cycle_counter < 6) { // Simulate 60s at 5V
        IO0SET = MQ7_HEATER_PIN;
    } else { // Simulate 90s at 1.4V (low)
        IO0CLR = MQ7_HEATER_PIN;
    }
    if (cycle_counter >= 15) cycle_counter = 0;
}

// Check Thresholds and Send Alerts
void check_and_alert(float temp, uint16_t co, uint16_t h2s, uint16_t flammable) {
    char msg[50];
```

```
    // Temperature Checks
    if (temp > TEMP_HIGH_THRESHOLD) {
        sprintf(msg, "High Temp Alert: %.1f C", temp);
        send_SMS(msg);
        IO1SET = LED_TEMP_HIGH; // Turn on high temp LED
    } else {
        IO1CLR = LED_TEMP_HIGH; // Turn off high temp LED
    }
    if (temp < TEMP_LOW_THRESHOLD) {
        sprintf(msg, "Low Temp Alert: %.1f C", temp);
        send_SMS(msg);
        IO1SET = LED_TEMP_LOW; // Turn on low temp LED
    } else {
        IO1CLR = LED_TEMP_LOW; // Turn off low temp LED
    }

    // CO Check
    control_MQ7_heater(); // Manage heater before reading
    if (co > CO_THRESHOLD) {
        sprintf(msg, "CO Gas Alert: %u ppm", co);
        send_SMS(msg);
        IO1SET = LED_CO; // Turn on CO LED
    } else {
        IO1CLR = LED_CO; // Turn off CO LED
    }

    // H2S Check
    if (h2s > H2S_THRESHOLD) {
        sprintf(msg, "H2S Gas Alert: %u ppm", h2s);
        send_SMS(msg);
        IO1SET = LED_H2S; // Turn on H2S LED
    } else {
        IO1CLR = LED_H2S; // Turn off H2S LED
    }

    // Flammable Gas Check
    if (flammable > FLAMMABLE_THRESHOLD) {
        sprintf(msg, "Flammable Gas Alert: %u ppm", flammable);
        send_SMS(msg);
        IO1SET = LED_FLAMMABLE; // Turn on flammable LED
    } else {
        IO1CLR = LED_FLAMMABLE; // Turn off flammable LED
    }
}
```

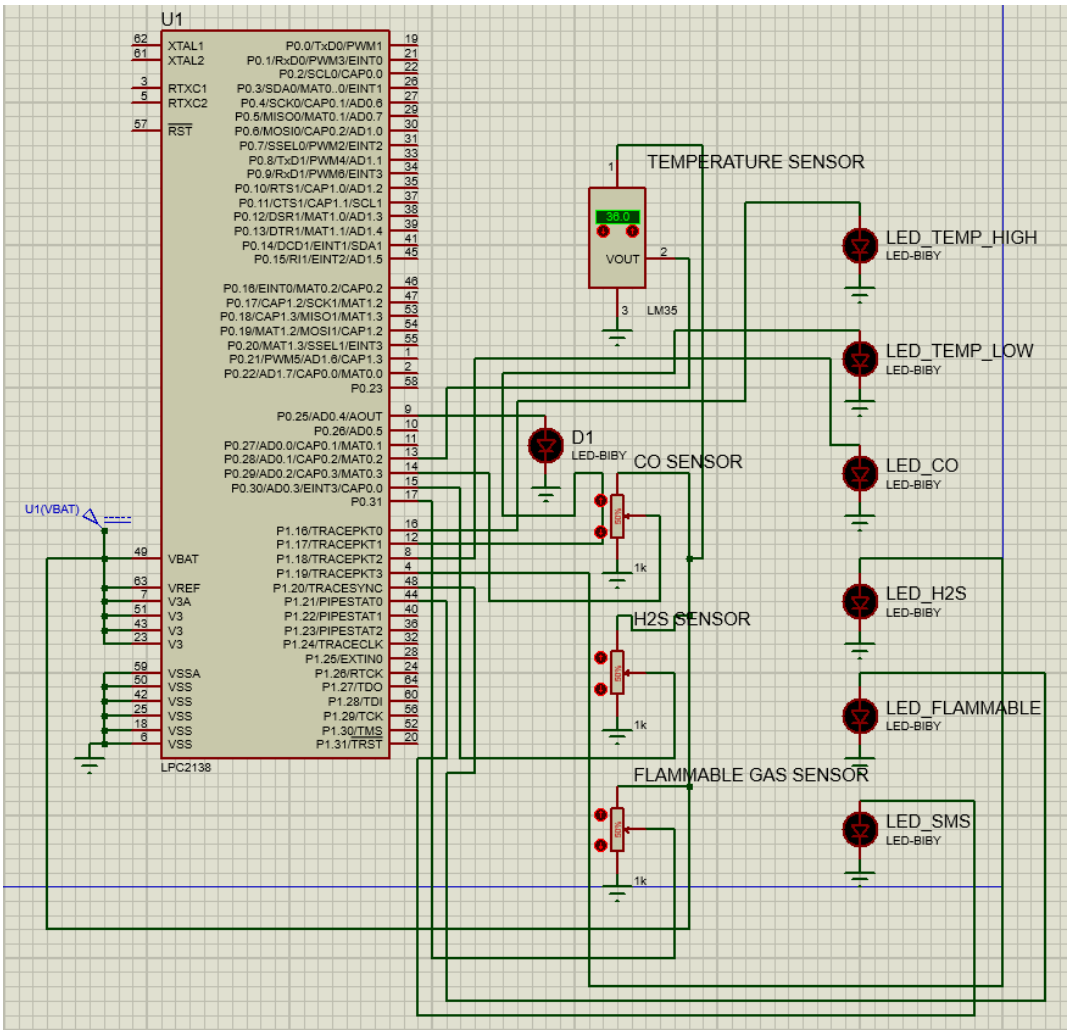
```
// ADC Read Function
uint16_t ADC_Read(uint16_t channel) {
    AD0CR &= ~(0xFF);        // Clear channel selection
    AD0CR |= (1 << channel);  // Select channel
    AD0CR |= (1 << 24);       // Start conversion
    while (!(AD0GDR & (1U << 31))); // Wait for completion
    return (AD0GDR >> 6) & 0x3FF;   // Extract 10-bit result
}
```

## Test Cases:

| Temp ADC Input (Normal: 46-118) | Temp (°C) (Normal: 15-38°C) | CO ADC Input (Normal: 0-51) | CO (ppm) (Normal: 0-50 ppm) | H2S ADC Input (Normal: 0-41) |
|---|---|---|---|---|
| 77 | ~24.8°C | 30 | ~29 ppm | 20 |
| 124 | ~39.9°C | 40 | ~39 ppm | 30 |
| 43 | ~13.8°C | 20 | ~19 ppm | 15 |
| 93 | ~29.9°C | 100 | ~97 ppm | 25 |
| 155 | ~49.9°C | 512 | ~500 ppm | 35 |
| 31 | ~10.0°C | 200 | ~195 ppm | 10 |
| 310 | ~99.9°C | 1023 | ~999 ppm | 50 |

| H2S (ppm) (Normal: 0-20 ppm) | Flammable Gas ADC Input (Normal: 0-102) | Flammable Gas (ppm) (Normal: 0-100 ppm) | Case |
|---|---|---|---|
| ~9.8 ppm | 50 | ~48 ppm | 1: Normal Conditions |
| ~14.6 ppm | 80 | ~78 ppm | 2: High Temp Only |
| ~7.3 ppm | 40 | ~39 ppm | 3: Low Temp Only |
| ~12.2 ppm | 70 | ~68 ppm | 4: High CO Only |
| ~17.1 ppm | 90 | ~87 ppm | 5: High Temp + High CO |
| ~4.9 ppm | 60 | ~58 ppm | 6: Low Temp + High CO |
| ~24.4 ppm | 150 | ~146 ppm | 7: Extreme Conditions (High Temp + High CO + High H2S + High Flammable) |

## 8. Implementation of Circuit in Proteus:



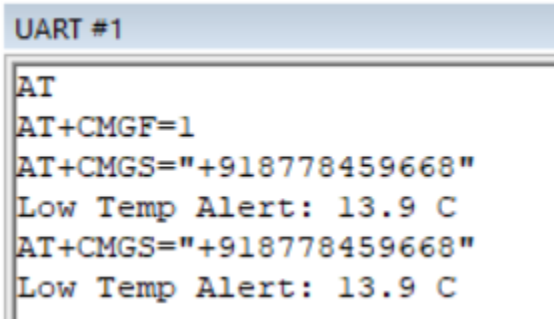

## Code Output:

Case-1: High Temp Only:

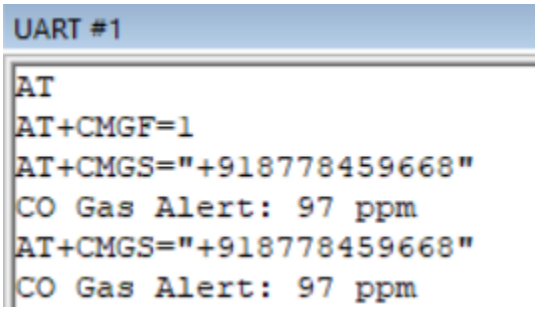
UART #1

```
AT
AT+CMGF=1
AT+CMGS="+918778459668"
High Temp Alert: 40.0 C
AT+CMGS="+918778459668"
High Temp Alert: 40.0 C
```

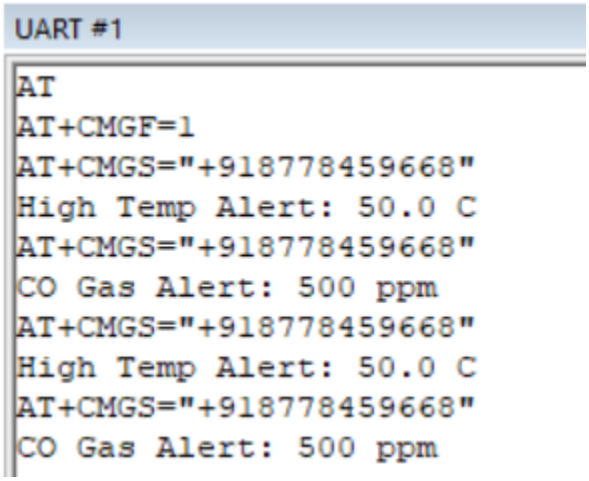
Case-2: Low Temp Only:



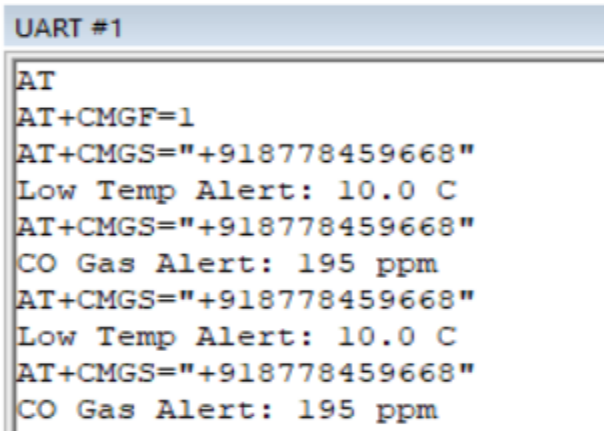

Case-3: High CO Only:



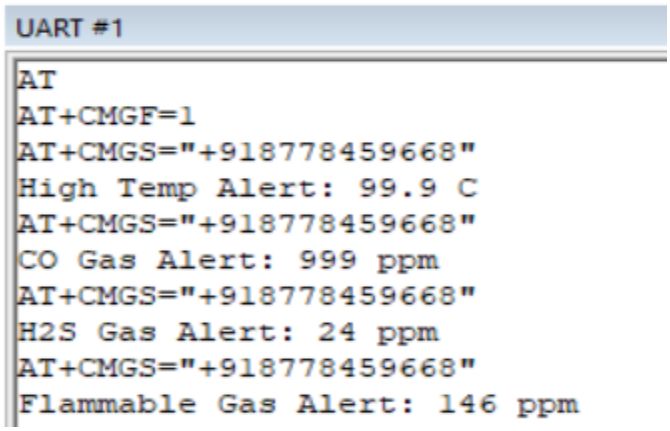

Case-4: High Temp and High CO:




Case-5: Low Temp and High CO:



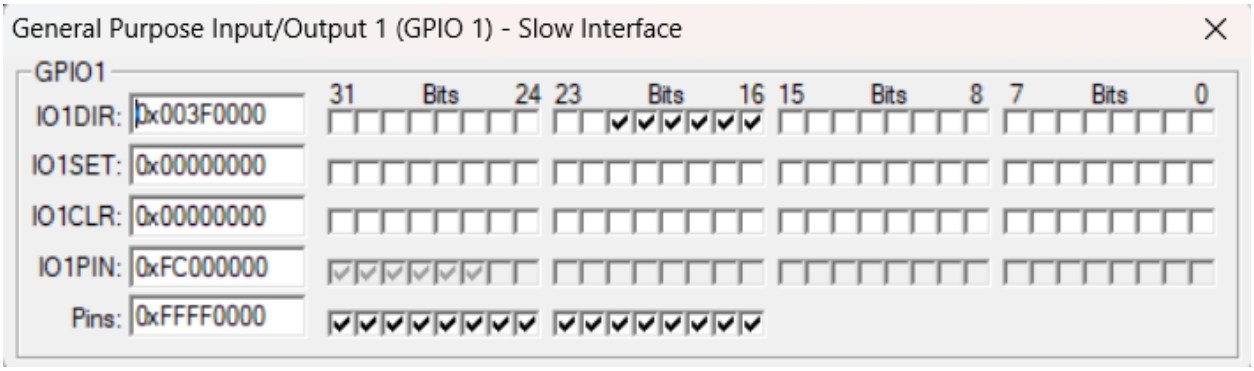
Case-6: Extreme Conditions:



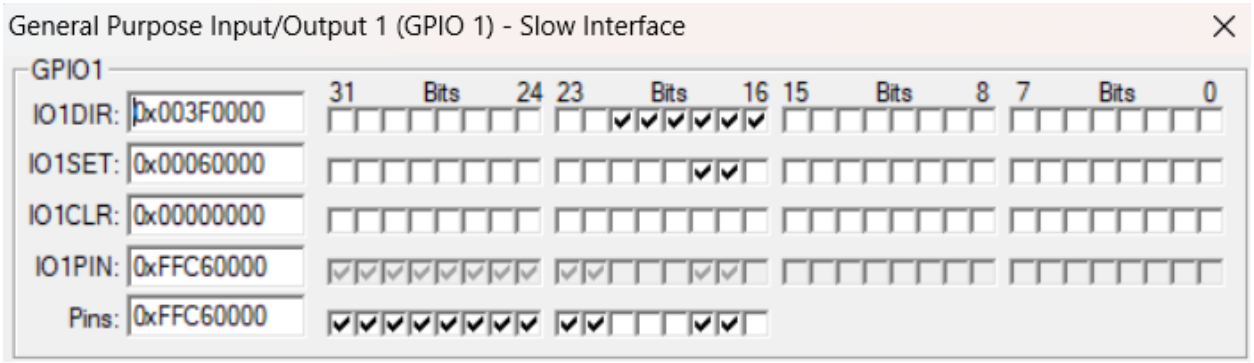

LED GPIO Configuration:

- LED_TEMP_HIGH (P1.16).
- LED_TEMP_LOW (P1.17).
- LED_CO (P1.18).
- LED_H2S (P1.19).
- LED_FLAMMABLE (P1.20).
- LED_SMS (P1.21).

1. GPIO Pins in Clear State:




2. GPIO Pins in Set State:



## 9. Ideas for the Future:

- **More Sensors**: Add sensors for heart rate (to check if a worker is stressed or sick), air pressure (for high-altitude work), or even motion (to detect falls). This would make the helmet a full health and safety guard.
- **Better Alerts:** Instead of just texting, it could call someone, send data to an app on a supervisor's phone, or even sound an alarm on the helmet itself to warn the worker directly.
- **Data Logging:** Save all the sensor readings (like temperature and gas levels) over time. Later, bosses could look at this data to spot patterns—like if a certain area is always too hot—and fix problems before they get worse.
- **Wireless Options:** Use Bluetooth to connect to nearby devices (like a worker's phone) or LoRaWAN (a long-range, low-power signal) to send data far away without needing cell service.
- **Smart Features:** Add a small screen or voice system inside the helmet to tell workers, "Hey, it's getting hot—take a break!" or "Gas detected—move out!"
- **Power Boost:** Make it solar-powered with tiny panels on the helmet, so it charges itself while you work in the sun.

**Materials Required for Future Scope:**

**Hardware Materials:**

**Extra Sensors:**

Heart rate sensor (like a pulse oximeter, e.g., MAX30100) – a tiny chip that clips inside the helmet to check your pulse.

Accelerometer (e.g., MPU6050) – a small device to detect falls or sudden movements.

Pressure sensor (e.g., BMP280) – to measure air pressure in high places.

**Display or Speaker:**

Tiny OLED screen (e.g., 0.96-inch SSD1306) – to show messages like "Temp: 40°C".

Small buzzer or speaker – to beep or talk to the worker.

**Wireless Modules:**

Bluetooth module (e.g., HC-05) – for short-range connection to phones.

LoRa module (e.g., SX1278) – for long-range data sending in remote areas.

**Bigger Battery or Solar Panels:**

Rechargeable lithium battery (e.g., 3.7V, 2000mAh) – to power all the new stuff.

Small solar cells – to charge the battery in sunlight.

**Casing:**

Waterproof, dustproof enclosure – to protect the new parts from rain or dirt.

**Software Materials:**

**Programming Tools:**

Updated code for the LPC2148 – to handle new sensors and features.

Mobile app code (e.g., using Android Studio) – for supervisors to see alerts and data.

**Libraries:**

Software snippets (like Arduino libraries) – to make new sensors talk to the LPC2148 easily.

**Reasoning:**

More features mean more parts. A heart rate sensor needs a chip, and a screen needs wires and space. Wireless stuff needs antennas. Everything has to fit in the helmet without making it heavy or uncomfortable.

**Additional Hardware:**

**Heart Rate Sensor (MAX30100):**

Measures your pulse by shining light through your skin (near the forehead, maybe). If your heart's racing, it could mean heat stress or panic—time to alert someone!

**Accelerometer (MPU6050):**

Detects if you fall or trip by sensing sudden jerks. If it feels a big drop, it could text, "Worker fell at Location X!"

**Pressure Sensor (BMP280):**

Checks air pressure, which changes with height. Useful in mines or tall buildings to warn about unsafe altitudes.

**OLED Screen (SSD1306):**

A tiny screen inside the helmet to show warnings or stats. Imagine seeing “CO: 50 ppm” flash in front of you.

**Bluetooth Module (HC-05):**

Connects the helmet to a phone or tablet nearby. You could see live data or get alerts without waiting for a text.

**LoRa Module (SX1278):**

Sends data miles away with little power—great for remote sites where cell service is spotty.

**Solar Panels:**

Tiny panels on top of the helmet to catch sunlight and recharge the battery, so it lasts longer.

**Additional Software/Code:**

**Sensor Code:**

Write instructions to read the heart rate sensor (e.g., “Check pulse every 5 seconds”) or accelerometer (“If motion stops suddenly, send an alert”).

**Display Code**:

Tell the OLED screen what to show, like “if temp > 38°C, display ‘Too Hot!’”.

**Wireless Code:**

For Bluetooth: “Send temp and gas data to the phone every minute.”

For LoRa: “Send an alert 5 miles to base if gas hits 50 ppm.”

App Development:

Build a phone app where supervisors see live data (like a graph of temperature over time) and get pop-up alerts.

Data Logging Code:

Save sensor readings in the LPC2148’s memory (e.g., “Store temp every 10 minutes”) and send them later for analysis.

**Power Management:**

Add rules like “Turn off the screen when not needed” to save battery.

## 10. Challenges and Risks

**Fast battery drain:**

More sensors and wireless modules use more battery. If it dies too fast, the helmet is useless. Solar panels help, but not at night or indoors.

**Too Heavy:**

Extra parts could make the helmet bulky or uncomfortable. Workers won't wear it if it hurts their necks!

**Costly:**

New sensors, screens, and wireless modules cost money. If it's too expensive, companies might skip it for cheaper, dumber helmets.

**False Alarms:**

Sensors might mess up—like thinking a hot day is a fever or dust is gas. Too many fake alerts annoy people and waste time.

**Breaking Down:**

Factories are rough—dust, water, or bangs could wreck the new parts. We need tough designs, but that's hard and pricey.

**Signal Issues:**

In deep mines or remote spots, GSM, Bluetooth, or LoRa might not work. No signal = no alerts.

**Learning Curve:**

Workers and bosses need to learn how to use the app or read the screen. If it's confusing, they might ignore it.

## 11. Conclusion

The smart helmet is an innovative solution for monitoring temperature, detecting hazardous gases, and sending emergency alerts. However, the proposed future enhancements elevate its capabilities significantly, integrating additional sensors, improved alert systems, and advanced connectivity features. The inclusion of heart rate monitoring, fall detection, and wireless communication technologies has the potential to revolutionize worker safety across various industries.

While challenges such as power consumption, cost, and durability must be addressed, overcoming these obstacles could establish the smart helmet as an essential safety tool in

hazardous work environments. By incorporating these advancements, the helmet could provide a more comprehensive approach to workplace safety, ensuring real-time hazard detection and proactive response mechanisms. The continued evolution of this technology holds immense promise for enhancing occupational safety and operational efficiency.

## 12. Reference:


1. Smart Helmet for industrial workers with health monitoring, accessed on March 21, 2025, https://ijecsd.com/images/convertforms/Vol5_issue2_paper_4.pdf

2. LPC2148 Datasheet (PDF) - NXP Semiconductors - Aldatasheet.com, accessed on March 21,2025, https://www.alldatasheet.com/datasheetpdf/pdf/432847/NXP/LPC2148.html

3. LM35 Datasheet, PDF - ALLDATASHEET.COM, accessed on March 21, 2025, https://www.alldatasheet.com/view.jsp?Searchword=lm35

4. TMP36 Temperature Sensor Pinout, Features, Equivalent, Circuit & Datasheet, accessed on March 21, 2025, https://components101.com/sensors/tmp36-temperature-sensor

5. Flammable Gas & Smoke Sensor MQ-2 - Pololu, accessed on March 21, 2025, https://www.pololu.com/product/1480

6. MQ-7 CARBON MONOXIDE CO GAS SENSOR DETECTOR ..., accessed on March 21, 2025, https://support.envistiamall.com/kb/mq-7-carbon-monoxide-co-gas-sensor-detector-module/

7. MQ7 - Carbon Monoxide Gas Sensor Module - visha world, accessed on March 21, 2025, https://vishaworld.com/products/mq7-carbon-monoxide-sensor-module