NAME: THOMAS OKOT

REG N0: 24/U/0296/GIW/PS

Selected Topic: Remote Patient Health Monitoring System

Industry: *E-Health*

Project Title:

Remote Patient Health Monitoring System Using ESP32 and IoT

Introduction

In the wake of growing healthcare demands and limited accessibility to hospitals in rural areas, there is a need for cost-effective and real-time health monitoring systems. This project focuses on developing a **Remote Patient Health Monitoring System** under the **E-Health industry**, leveraging embedded systems and Internet of Things (IoT) technologies to monitor patient vitals remotely.

Problem Statement / Challenges

- Lack of constant monitoring of patients in rural areas or at home.
- Overcrowding in hospitals due to patients who can be monitored remotely.
- Limited health personnel to attend to all patients individually.
- Delay in identifying sudden changes in a patient's condition (e.g., drop in oxygen levels or abnormal heart rate).

Project Objectives

- To design a portable and low-cost embedded device for real-time monitoring.
- To monitor **temperature**, **heart rate**, and **oxygen saturation (SpO2)** levels remotely.
- To send real-time data to the cloud via Wi-Fi using an ESP32.
- To alert caregivers via SMS or email if any abnormal values are detected.

Technical Description

Components Used

Component	*Function*
ESP32 Wi-Fi Module	Acts as the microcontroller and WiFi interface
MAX30100 / MAX30102 Sensor	Measures heart rate and SpO2
DHT11 / DHT22	Temperature and humidity sensor

OLED Display	To locally display data
Buzzer	For local alerts
Firebase / Thingspeak	Cloud platform to store and visualize data
IFTTT / Twilio	To send SMS/email alerts

Working Principle

1. **Data Collection**:
   The system continuously reads:
   - Heart rate and SpO2 from MAX30100 sensor
   - Body/environment temperature from DHT11 sensor
2. **Processing**:
   The ESP32 processes this data, compares it against safe thresholds.
3. **Data Upload**:
   All data is sent to Firebase or Thingspeak via Wi-Fi for cloud storage and visualization.
4. **Alert System**:
   If a patient's vitals are outside the safe range, the system:
   - Activates a buzzer
   - Sends an SMS or email alert using IFTTT or Twilio
5. **Monitoring**:
   Health workers or family members can monitor the real-time status of the patient from anywhere.

Key Benefits

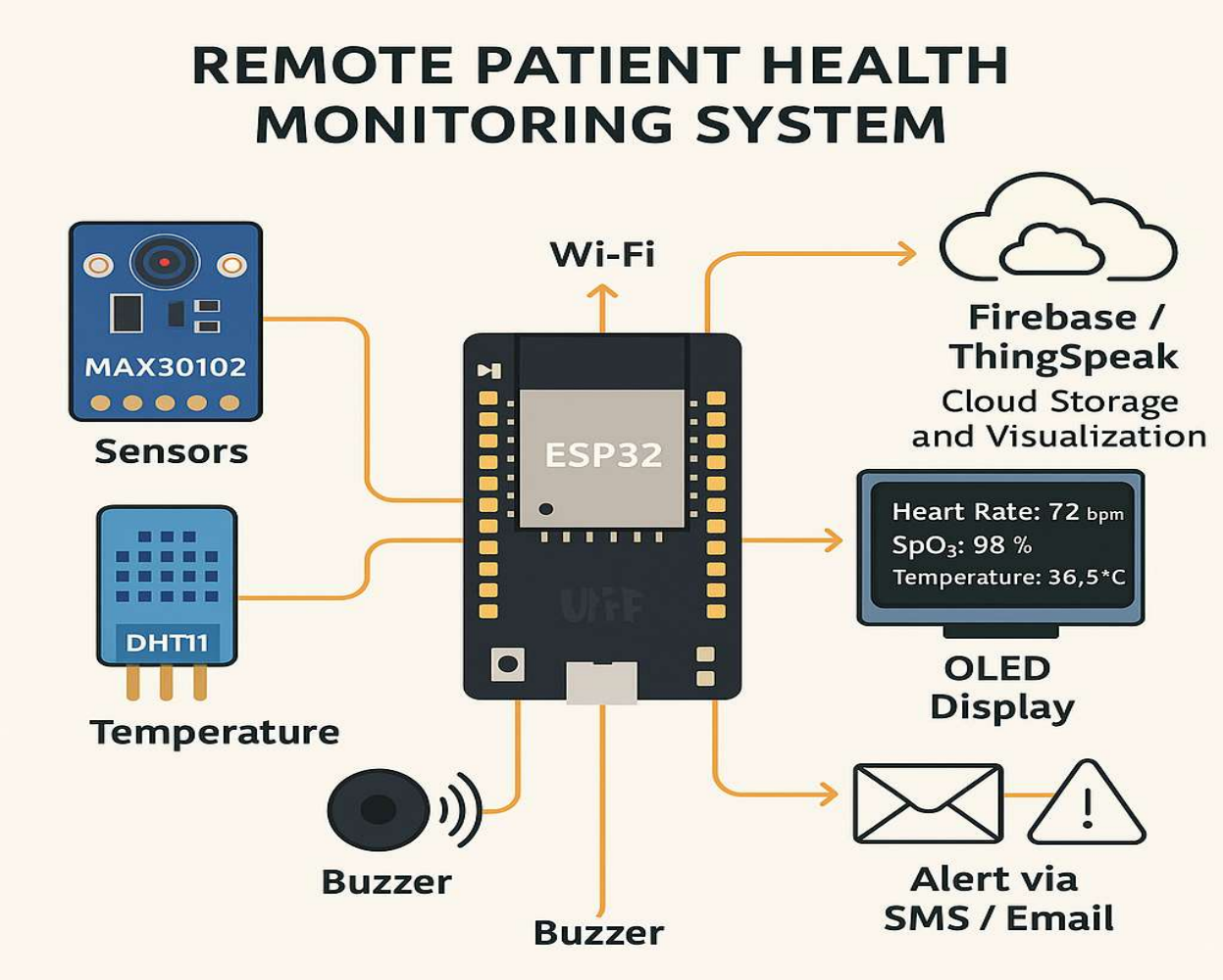
- Enhances home-based care
- Real-time remote monitoring
- Reduces hospital congestion
- Affordable and scalable

Future Enhancements

- Add blood pressure sensor
- Include GPS tracking
- Integrate with mobile health apps
- Add machine learning for health prediction

2. Project Designs and Source Codes

Hardware Diagram



Arduino Source Code (ESP32)

```
#include <WiFi.h>
#include "MAX30100_PulseOximeter.h"
#include <DHT.h>
#include <ThingSpeak.h>

#define REPORTING_PERIOD_MS 1000
#define DHTPIN 4
#define DHTTYPE DHT11
```

```
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
WiFiClient client;

// ThingSpeak Settings
unsigned long channelID = YOUR_CHANNEL_ID;
const char* writeAPIKey = "YOUR_WRITE_API_KEY";

PulseOximeter pox;
DHT dht(DHTPIN, DHTTYPE);
uint32_t tsLastReport = 0;

void onBeatDetected() {
    Serial.println("Beat!");
}

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500); Serial.print(".");
    }
    Serial.println("WiFi Connected.");

    ThingSpeak.begin(client);
    dht.begin();

    if (!pox.begin()) {
        Serial.println("PulseOximeter init failed.");
        while(1);
    }

    pox.setOnBeatDetectedCallback(onBeatDetected);
}

void loop() {
    pox.update();

    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
        tsLastReport = millis();

        float bpm = pox.getHeartRate();
        float spo2 = pox.getSpO2();
        float temp = dht.readTemperature();

        Serial.print("Heart rate: "); Serial.println(bpm);
        Serial.print("SpO2: "); Serial.println(spo2);
        Serial.print("Temperature: "); Serial.println(temp);

        ThingSpeak.setField(1, bpm);
        ThingSpeak.setField(2, spo2);
        ThingSpeak.setField(3, temp);
        ThingSpeak.writeFields(channelID, writeAPIKey);

        // Alert Condition
```

```
        if (bpm < 50 || bpm > 120 || spo2 < 90 || temp > 38) {
            Serial.println("Abnormal condition detected!");
            // Add buzzer or IFTTT alert code here
        }
    }
}
```

Detailed Explanation on how the system works

1. Sensing Patient Vitals

Sensors Involved:

- **MAX30102 Sensor**: Measures **heart rate** and **oxygen saturation (SpO_2)**
- **DHT11 Sensor**: Measures **body temperature** (or environmental if not placed on body)

These sensors are **connected to the ESP32**, which reads the data through its analog or I2C/GPIO pins.

2. Processing with ESP32

The **ESP32** acts as the brain of the system. It performs the following tasks:

- Continuously **reads data** from the MAX30102 and DHT11 sensors
- **Processes** the data to determine if it's within normal health thresholds
- Displays data on the **OLED screen** for local visibility
- Sends data to a **cloud platform** for remote access

3. Wi-Fi and Cloud Connection

- The ESP32 has **built-in Wi-Fi**, which connects to a router.
- It transmits the data to a **cloud platform** like **Firebase** or **ThingSpeak**, where health workers can **visualize and monitor data in real-time**.

4. Display & Feedback

- An **OLED display** shows:
  - Heart rate (bpm)
  - Oxygen level (SpO_2 %)
  - Temperature (°C)
- This helps patients or attendants view current readings without needing internet access.

5. Alert System

- If the ESP32 detects abnormal values (e.g., low SpO_2, high fever), it:
  - Triggers a **buzzer** to alert the patient or caregiver nearby.
  - Sends a **real-time alert via SMS or email** using **IFTTT or Twilio**, integrated with cloud services.

For example:

If the SpO_2 drops below 90%, or temperature exceeds 38°C, the system sounds the buzzer and sends an emergency SMS to a registered number.

6. Power Supply

The system is powered by:

- USB (for testing)
- Rechargeable battery or power bank (for portability)

Summary of How It Works:

Step	Action
Step one	Sensors read heart rate, oxygen, and temperature
Step two	ESP32 processes and displays the data
Step three	ESP32 uploads data to Firebase/ThingSpeak
Step four	If abnormal, a buzzer sounds + SMS/Email is sent
Step five	Medical staff can access the data online

Component Breakdown with Power & Voltage Requirements

Component	Function	Voltage (V)	Current (mA)	Power Source
ESP32	Main controller, Wi-Fi, data processing	3.0 – 3.3 V	160 – 240 mA (WiFi on)	LDO Regulator or 3.3V supply
MAX30102	Heart rate and SpO_2 sensor	1.8V (core), 3.3V (I/O)	~1 – 4 mA	3.3V from ESP32
DHT11	Temperature and humidity sensor	3.0 – 5.5 V	0.5 – 2.5 mA	Direct from ESP32 GPIO
OLED Display (0.96")	Displays heart rate, SpO_2, temp	3.3 – 5V	~20 – 30 mA	3.3V or 5V pin
Buzzer (Passive/Active)	Alerts when abnormal readings are detected	3.0 – 5V	30 – 50 mA	5V or via transistor from ESP32
Power Source	Battery/USB for powering entire system	5V input	Varies (200–500 mA total)	Li-Ion battery / USB / 5V adapter

Power Supply Design

Typical Setup

- **Power input**: 5V (from USB, power bank, or Li-ion battery with charger module)
- **Voltage regulation**:
  - Use **AMS1117-3.3** or onboard regulator to step down from 5V to 3.3V for ESP32 and other 3.3V components
- Total system consumption: ~200–300 mA (during Wi-Fi transmission)

Backup & Portability

- Optionally use a **3.7V Li-ion battery (e.g., 18650)** with **TP4056 charging module** and boost converter to maintain 5V output.
- Add a **switch** for turning system on/off.

How the System Works (with Power Flow)

1. Powering the System

- The **5V input** (from USB or battery) is supplied to the ESP32.
- ESP32's onboard regulator converts 5V to **3.3V** for itself and connected sensors like MAX30102 and OLED.
- The **DHT11** is tolerant to 3.3V–5V and can connect directly to a GPIO.

2. Sensor Operation

- **MAX30102** uses IR and red LEDs to detect blood oxygen and pulse.
    - It draws minimal current unless LEDs are active.
- **DHT11** reads temperature and humidity.
    - It operates at low frequency and power.

3. Data Processing

- ESP32 reads sensor data via I2C or GPIO.
- Data is processed and:
    - Displayed on the **OLED**
    - Sent to **cloud storage (Firebase/ThingSpeak)** via built-in Wi-Fi

4. Alerts

- If vitals cross critical thresholds:
    - The **buzzer** is triggered (e.g., $SpO_2 < 90\%$ or temperature > 38°C).
    - An **SMS or email** is sent via cloud automation (IFTTT/Twilio).

5. Cloud Monitoring

- Wi-Fi sends data to a cloud platform:
    - **Firebase** or **ThingSpeak**
- Caregivers can view real-time patient data remotely.

Example of power Budget

Component	Voltage	Current (max)	Power (mW)
ESP32	3.3V	240mA	792mW
MAX30102	3.3V	4mA	13.2 mW
DHT11	3.3V	2mA	6.6 mW
OLED Display	3.3V	30mA	99 mW
Buzzer	5V	40mA	200 mW
Total	**-**	**~316 mA**	**~1.1 W**

A **500mAh battery** would last approximately **1.5 to 2 hours** under full operation, more with optimizations.

Final Notes

- All components are **low power**, ideal for battery-powered wearable or portable solutions.
- The system can be expanded with **battery management** (solar charging, deep sleep, etc.).
- Care should be taken to match voltage levels — especially between ESP32 (3.3V logic) and any 5V sensors or modules.