DEPARTMENT OF ELECTRONIC & TELECOMMUNICATION ENGINEERING, UNIVERSITY OF MORATUWA

BM 2210 - Biomedical Device Design

Project Report: Temperature-Controlled Medicine and Vaccine Cooler Box

Group C

COLOMBAGE DM	230108U
RAHMAN MFA	230507R
ABISHEK L	230016K
SANTHOSH S	230581K

Contents

1 Introduction

Vaccines must remain within 2–8°C to preserve potency. However, during field transportation—particularly in rural outreach—healthcare workers lack real-time visibility to detect temperature excursions promptly. Existing passive cold boxes, ice packs, data loggers, and solar fridges fail to provide an affordable, lightweight, continuous monitoring solution.

This project proposes a **portable, low-cost vaccine transport monitoring system** featuring digital temperature sensing, wireless transmission, configurable thresholds, and immediate alerts. A Peltier-based cooling system was planned but not implemented due to time constraints; it is included under future improvements.

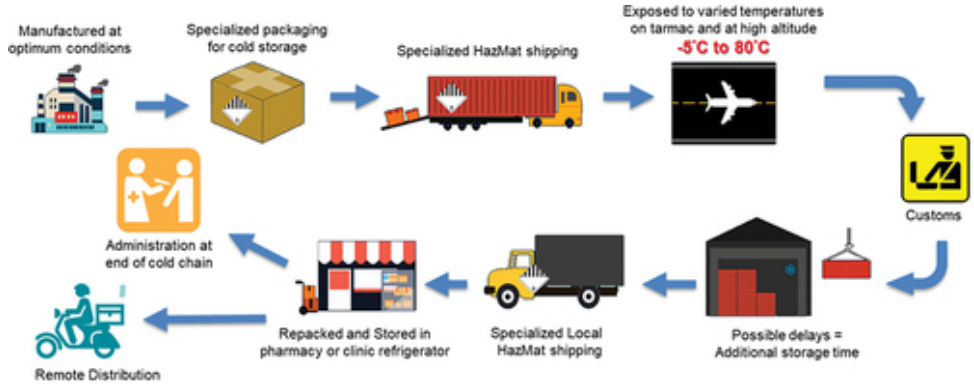

Figure 1: Illustration of vaccine cold-chain workflow (Insert relevant image)

2 Need Statement

"A way to maintain safe temperature conditions for temperature-sensitive vaccines during transport for healthcare workers in rural and low-resource settings to improve the reliability of vaccine delivery."

3 Proposed Solution

The proposed solution is a **Smart Vaccine Temperature Monitoring System** designed to operate independently of external networks using ESP-NOW. The system includes:

- A DS18B20 digital temperature sensor placed inside the insulated vaccine chamber.
- An ESP8266 microcontroller acting as the transmitter inside the box.
- ESP-NOW-based wireless communication to a remote ESP32-S3 receiver.
- An OLED display for real-time readouts.
- Configurable upper and lower temperature thresholds via a 4x4 keypad.
- Visual and audio alerts (LED + buzzer) when excursions occur.
- A basic mobile app to receive notifications and temperature data.

This hybrid approach balances cost, simplicity, and functionality without depending on mobile networks or Wi-Fi infrastructure.

4 Methodology

4.1 Design Flow

The project followed the structured BM2210 biomedical device design process. Each stage contributed to developing a practical, affordable solution suitable for real-world field conditions.

1. Need Identification

A review of cold-chain challenges in Sri Lanka and global rural vaccination programs highlighted that temperature excursions commonly go unnoticed during transport. Interviews with healthcare workers and analysis of WHO cold-chain reports confirmed the need for an affordable, real-time monitoring solution.

2. Ideation

Multiple approaches were brainstormed, including fully passive insulation upgrades, sensor-only monitoring systems, Bluetooth/Wi-Fi connected devices, and hybrid active–passive solutions. Constraints such as cost, weight, power consumption, and field usability guided the ideation process.

3. Concept Selection

Candidate concepts were evaluated using feasibility criteria: technological complexity, component availability, user training requirements, cost, and durability. The hybrid digital monitoring system with optional active cooling (Peltier) was selected as it offered the best balance between practicality and performance.

4. Prototyping

Subsystems—including sensor circuitry, ESP-NOW communication links, keypad threshold input, alarm mechanisms, and enclosure layout—were developed and tested incrementally. A functional prototype incorporating all essential features was assembled using available components.

5. Testing

The prototype was tested in both controlled and outdoor conditions to evaluate wireless range, temperature measurement stability, alert responsiveness, and battery performance. Simulated excursions verified correct activation of thresholds and communication between transmitter and receiver.

6. Refinement

Based on test results, improvements were made to code efficiency, display readability, alarm timing, and enclosure insulation. The Peltier cooling subsystem was postponed for future work due to time constraints but left structurally supported within the design for later integration.

4.2 Block Diagram

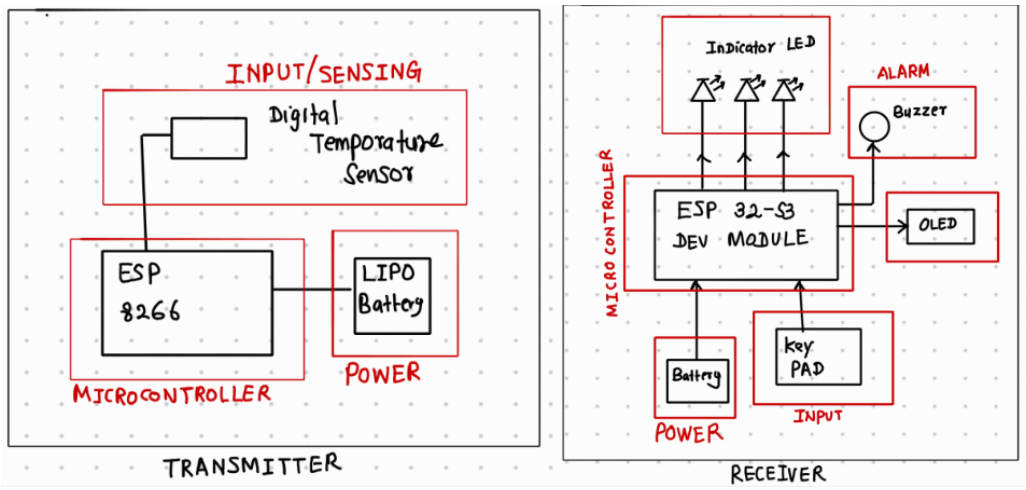

Figure 2: System block diagram showing transmitter and receiver

5 Device Details

5.1 Component Selection

Components used include:

- ESP8266 NodeMCU (Transmitter)
- ESP32-S3 (Receiver)
- DS18B20 digital temperature sensor
- 4x4 matrix keypad
- 0.96” OLED display (I2C)
- Active buzzer and LED indicators
- 18650 Li-ion battery with boost converter

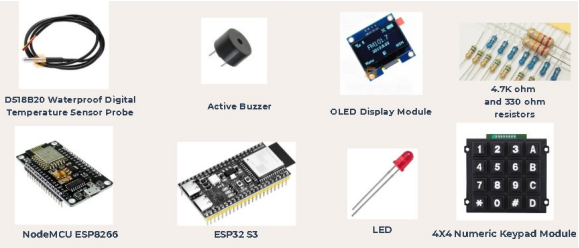

Figure 3: Major components used in the system (Part 1)

Figure 4: Major components used in the system (Part 2)

5.2 Circuit Diagrams

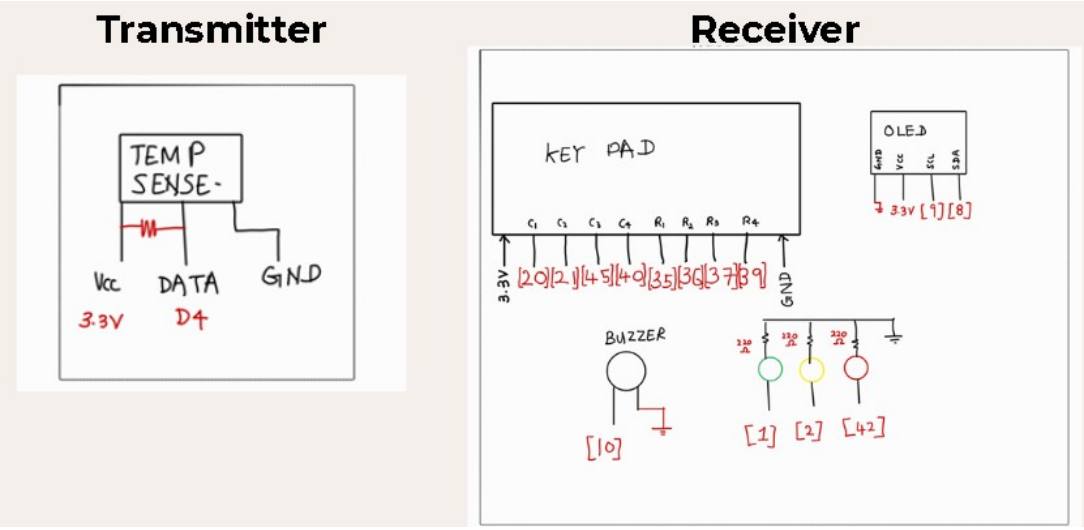

Figure 5: Transmitter circuit diagram

5.3 Enclosure Design

The enclosure was designed to securely house the transmitter, receiver, sensors, and other electronics while providing insulation for the vaccine compartment. The design focused on:

- Proper spacing for airflow and heat dissipation.
- Easy access for battery replacement and component maintenance.
- Mounting provisions for the buzzer, LEDs, keypad, and OLED display.
- Compatibility with future Peltier cooling integration.
- Lightweight and durable material selection for portability.

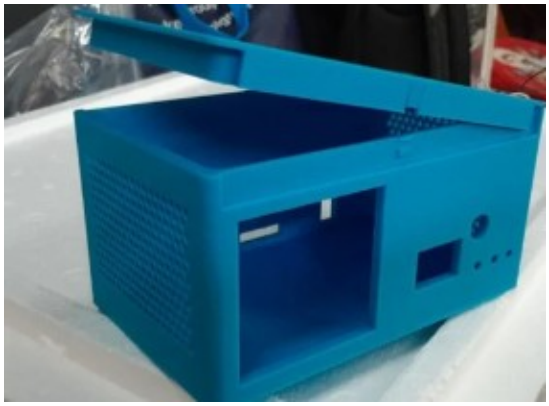

Figure 6: Front view of the Enclosure

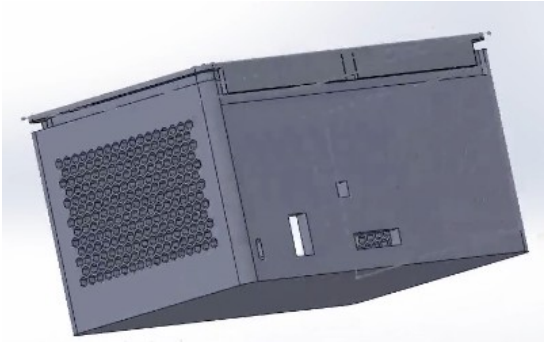

Figure 7: Side view

5.4 PCB Layout

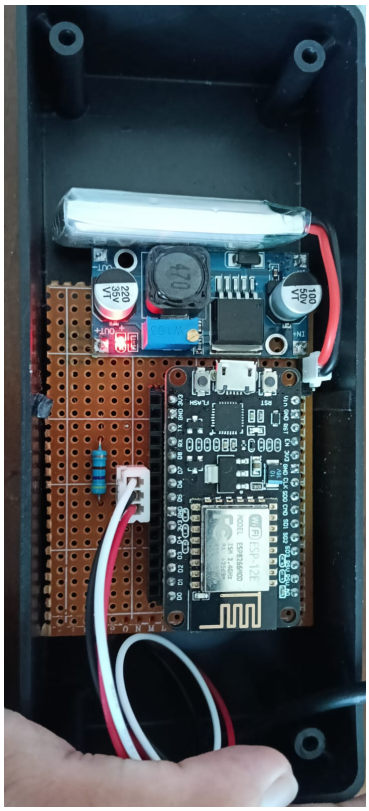

Figure 8: Transmitter PCB / veroboard layout

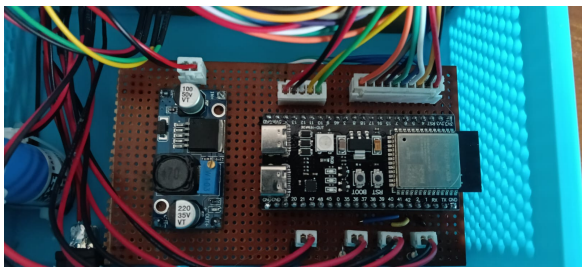

Figure 9: Receiver PCB / veroboard layout

5.5 Software Code (ESP-NOW Implementation)

Below is the core transmitter logic:

```
#include <ESP8266WiFi.h>
#include <espnow.h>
#include <OneWire.h>
#include <DallasTemperature.h>


// ================================
// DS18B20 Setup
// ================================
#define ONE_WIRE_BUS 2   // D4 = GPIO2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);


// ================================
// Receiver MAC Address (ESP32-S3)
// ================================
uint8_t receiverMAC[] = {0x30, 0xED, 0xA0, 0xB9, 0xCD, 0xE0};
```

```
// ================================
// Data Structure
// ================================
typedef struct struct_message {
  float temperature;
} struct_message;


struct_message data;


// ================================
// Setup
// ================================
void setup() {
  Serial.begin(115200);
  sensors.begin();


  WiFi.mode(WIFI_STA);  // MUST be station mode for ESP-NOW


  if (esp_now_init() != 0) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }


  // Set device role
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);


  // Add receiver peer
  if (esp_now_add_peer(receiverMAC, ESP_NOW_ROLE_SLAVE, 1, NULL, 0)
     != 0) {
    Serial.println("Failed to add peer");
    return;
  }


  Serial.println("ESP8266 DS18B20 Transmitter Ready");
}


// ================================
// Loop
// ================================
void loop() {
  sensors.requestTemperatures();
```

```
  float tempC = sensors.getTempCByIndex(0);


  if (tempC == DEVICE_DISCONNECTED_C) {
    Serial.println("DS18B20 Error: Sensor not detected");
    delay(1000);
    return;
  }


  data.temperature = tempC;


  // Send structure via ESP-NOW
  esp_now_send(receiverMAC, (uint8_t *)&data, sizeof(data));


  Serial.printf("Sent -> Temperature: %.2f C\n", data.temperature);


  delay(1000);
}
```

Below is the receiver code logic:

```
#include <WiFi.h>
#include <esp_now.h>
#include <Keypad.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// ================================
// OLED SETUP
// ================================
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SDA_PIN 8
#define SCL_PIN 9

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
   OLED_RESET);

// ================================
// Transmitter MAC Address (ESP8266)
// ================================
uint8_t senderMAC[] = {0xEC, 0xFA, 0xBC, 0xCA, 0x2A, 0xF3};

// ================================
// Data Structure
// ================================
```

```
typedef struct struct_message {
  float temperature;
} struct_message;

struct_message incomingData;
bool dataReceived = false;
unsigned long lastPacketTime = 0;
#define DATA_TIMEOUT 2000

// ================================
// Keypad Setup
// ================================
#define ROWS 4
#define COLS 4
char keyMap[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'#','0','*','D'}
};
byte rowPins[ROWS] = {35, 36, 37, 39};
byte colPins[COLS] = {20, 21, 45, 40};
Keypad keypad = Keypad(makeKeymap(keyMap), rowPins, colPins, ROWS,
   COLS);

// ================================
// Thresholds
// ================================
float lowerThreshold = 0.0;
float upperThreshold = 100.0;
bool thresholdsSet = false;
bool inThresholdInput = false;

// ================================
// BUZZER AND LED SETUP
// ================================
#define BUZZER_PIN 10
bool buzzerActive = false;
unsigned long lastBuzzToggle = 0;
bool buzzState = false;

#define GREEN_LED 1
#define YELLOW_LED 2
#define RED_LED 42
unsigned long lastRedBlink = 0;
bool redState = false;

// ================================
// Callback when data is received
// ================================
void OnDataRecv(const esp_now_recv_info_t *recv_info, const uint8_t
```

```
   *incomingDataBytes, int len) {
  memcpy(&incomingData, incomingDataBytes, sizeof(incomingData));
  dataReceived = true;
  lastPacketTime = millis();

  if (!inThresholdInput && thresholdsSet) {
    float temp = incomingData.temperature;
    if (temp < lowerThreshold) Serial.println("Low Temp");
    else if (temp > upperThreshold) Serial.println("High Temp");
    else Serial.println("In Range");

    buzzerActive = (temp < lowerThreshold || temp > upperThreshold);
    if (!buzzerActive) digitalWrite(BUZZER_PIN, LOW);
  }
}

// ================================
// Function to input float via keypad
// ================================
float inputFloatValue(const char* prompt, bool isLower) {
  Serial.print(prompt);
  String inputStr = "";
  bool negative = false;
  bool decimalPointUsed = false;

  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print(isLower ? "Lower Threshold:" : "Upper Threshold:");
  display.display();

  while (true) {
    char key = keypad.getKey();
    if (key) {
      if (key == 'A') { if (!negative && inputStr.length()==0) {
         negative = true; Serial.print("-"); } }
      else if (key == 'B') { if (!decimalPointUsed) {
         decimalPointUsed = true; if(inputStr.length()==0){inputStr
         +="0";} inputStr+="."; Serial.print("."); } }
      else if (key >= '0' && key <= '9') { inputStr += key; Serial.
         print(key); }
      else if (key == 'D') { if(inputStr.length()>0){inputStr.remove
         (inputStr.length()-1);} else if(negative){negative=false;}
         }
      else if (key == 'C') { break; }
    }
    delay(10);
  }

  float value = inputStr.toFloat();
  if (negative) value = -value;
```

```
  return value;
}

// ================================
// Setup
// ================================
void setup() {
  Serial.begin(115200);

  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  pinMode(GREEN_LED, OUTPUT);
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);

  Wire.begin(SDA_PIN, SCL_PIN);
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 not found");
    while(true);
  }

  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(10,20);
  display.print("OLED OK");
  display.display();
  delay(1000);

  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) { Serial.println("Error initializing
      ESP-NOW"); return; }
  esp_now_register_recv_cb(OnDataRecv);

  esp_now_peer_info_t peerInfo = {};
  memcpy(peerInfo.peer_addr, senderMAC, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) { Serial.println("
     Failed to add ESP8266 as peer"); return; }

  Serial.println("ESP32-S3 Receiver Ready");
}

// ================================
// Loop
// ================================
void loop() {
  char key = keypad.getKey();
  if(key == '*') {
    inThresholdInput = true;
```

```
    lowerThreshold = inputFloatValue("Enter LOWER threshold: ", true
       );
    upperThreshold = inputFloatValue("Enter UPPER threshold: ",
       false);
    if(upperThreshold < lowerThreshold){ float tmp = upperThreshold;
        upperThreshold = lowerThreshold; lowerThreshold = tmp; }
    thresholdsSet = true;
    inThresholdInput = false;
  }

  if(buzzerActive && millis() - lastBuzzToggle > 500) {
    buzzState = !buzzState;
    digitalWrite(BUZZER_PIN, buzzState ? HIGH : LOW);
    lastBuzzToggle = millis();
  }

  if(dataReceived && millis() - lastPacketTime > DATA_TIMEOUT)
     dataReceived = false;

  display.clearDisplay();
  if(!dataReceived) { display.setTextSize(2); display.setCursor
     (20,20); display.print("NO DATA"); digitalWrite(YELLOW_LED,HIGH
     ); digitalWrite(GREEN_LED,LOW); digitalWrite(RED_LED,LOW); }
  else {
    digitalWrite(YELLOW_LED,LOW);
    display.setTextSize(1);
    display.setCursor(0,0);
    display.print("Lower:"); display.print(lowerThreshold,1);
    display.print(" Upper:"); display.print(upperThreshold,1);

    display.setTextSize(3);
    String tempStr = String(incomingData.temperature,1);
    int16_t x1, y1; uint16_t w,h;
    display.getTextBounds(tempStr,0,0,&x1,&y1,&w,&h);
    display.setCursor((SCREEN_WIDTH-w)/2,(SCREEN_HEIGHT-h)/2);
    display.print(tempStr);

    if(incomingData.temperature >= lowerThreshold && incomingData.
       temperature <= upperThreshold) {
      digitalWrite(GREEN_LED,HIGH);
      digitalWrite(RED_LED,LOW);
    } else {
      digitalWrite(GREEN_LED,LOW);
      if(millis()-lastRedBlink>=500){ redState=!redState;
         digitalWrite(RED_LED,redState); lastRedBlink=millis(); }
    }
  }
  display.display();
  delay(50);
}
```

6 Simulation Results

To verify behaviour before hardware testing, a simple simulation was performed showing the expected temperature variation and alert conditions.

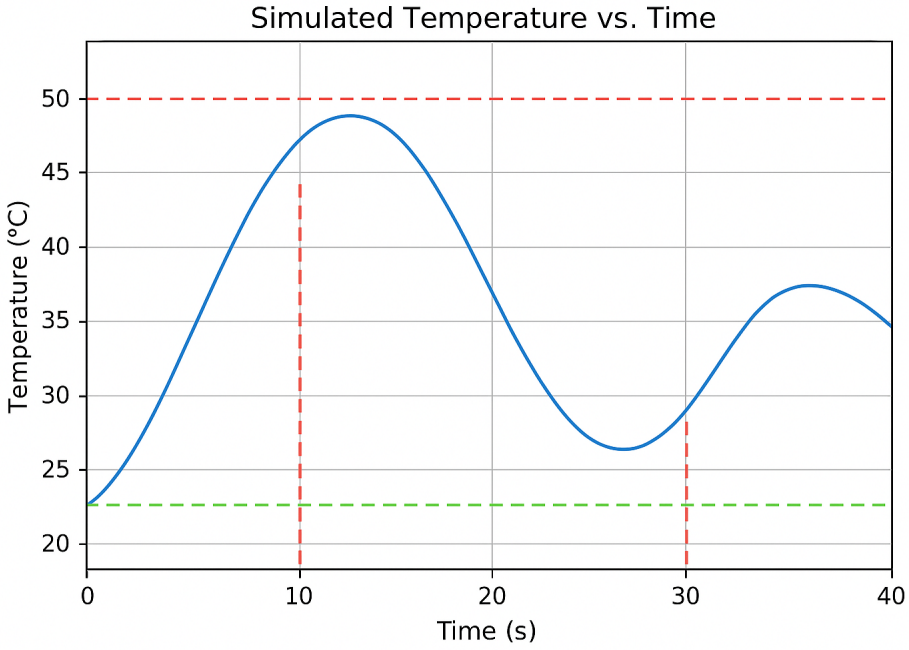

Figure 10: Simulated temperature vs. time graph showing threshold crossings

Simulation confirmed expected transitions and packet transmission rates.

7 Prototype and Testing

7.1 Prototype

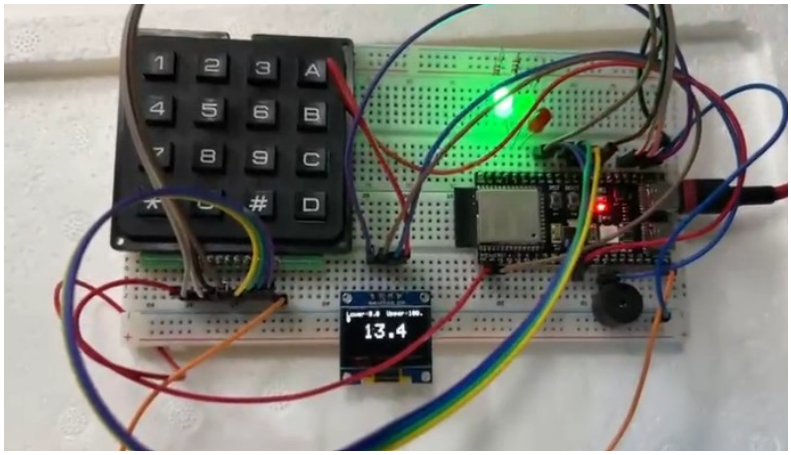

Figure 11: Prototype of the complete system

7.2 Test Results

The prototype was tested under both controlled and outdoor conditions. Key observations include:

- Wireless signal remained stable up to 20–30 m line-of-sight.
- Temperature readings were consistent and stable. A small accuracy deviation of approximately ± 0.5°C was observed when compared to a calibrated thermometer.
- Visual and audio alerts triggered correctly when temperature thresholds were exceeded.
- Battery performance was sufficient for continuous monitoring during field testing.

Overall, the system demonstrated reliable performance, with only minor sensor accuracy limitations that can be addressed in future improvements.

8 Possible Regulatory Pathway

8.1 Device Classification

- The device is likely classified as a **Class I medical device** (low-risk) under FDA regulations, since it only monitors temperature and provides alerts without directly treating or affecting the patient.
- Identify predicate devices with similar intended use and technology, e.g., digital vaccine temperature monitors or data loggers.

8.2 Preclinical Testing

- **Performance Testing:** Validate temperature sensor accuracy, alert response time, wireless communication reliability, and display readability.
- **Electrical Safety and EMC:** Test compliance with IEC 60601-1 and IEC 60601-1-2 for low-voltage medical electrical equipment.
- **Battery and Power Safety:** Evaluate Li-ion battery safety, charging circuits, and protection against overcurrent or short-circuit.
- **Usability:** Conduct human factors testing to ensure healthcare workers can set thresholds, read alerts, and handle the device easily in field conditions.

8.3 Software and Cybersecurity

- Comply with IEC 62304 for medical device software lifecycle and document testing of the ESP-NOW communication, threshold alerts, and mobile app integration.
- Ensure basic cybersecurity measures such as secure wireless communication and data integrity for transmitted temperature data.

8.4 Premarket Submission (510(k) or Exemption)

- Submit device description, intended use, and performance testing results if required.
- Since the device is low-risk, it may be **510(k)-exempt** or require minimal documentation depending on jurisdiction.
- Include labeling specifying proper use, temperature range, and safety precautions.

8.5 Manufacturing and Post-market Compliance

- Implement a Quality Management System (QMS) compliant with **21 CFR Part 820** or ISO 13485.
- Register the device and manufacturing facility with the FDA, EU MDR authorities, and Sri Lanka NMRA as required.
- Monitor post-market performance, collect user feedback, and report any malfunctions or safety issues through the appropriate adverse event reporting system.

9 Final Budget

Table 1: Final Budget for the Vaccine Cooler Box Project

S/N	Product / Component	Price (LKR / USD)
1	ESP8266 NodeMCU	700
2	ESP32-S3	1850
3	DS18B20 Temperature Sensor	240
4	0.96” OLED Display	540
5	4x4 Keypad	680
6	Buzzer	50
7	LED Indicators	30
8	18650 Li-ion Battery	170
9	Boost Converter Module	150
10	Insulated Box / Enclosure Materials	3000
11	Wires and Other connectors	1000
-	**Total Cost**	**8410**

10 Task Allocation

Name	Task
Colombage DM	Circuit Designing, Algorithm Development, PCB soldering
Rahman MFA	Wireless Communication configuration, Enclosure Designing
Abishek L	APP development
Suresh S	Documentation

11 Conclusion and Final Product

The developed system offers a reliable, low-cost method for monitoring vaccine temperature during transport. It provides real-time feedback, configurable thresholds, wireless alerts, and ease-of-use suitable for rural healthcare workflows.

Figure 12: Final product

12 Future Improvements

- Integration of Peltier-based active cooling
- Enhanced insulation materials
- Solar-powered charging module
- Multi-sensor thermal mapping
- Cloud-based data logging