Computer Interface Course Project Report

4th Year Communication Engineering

Project Title: *Hospital Room Monitoring*

Team ID: *5A*

Team Members:

SN	*Student Name*	*Section No.*
1	Kirollos Nashaat Adeeb	5
2	Mohamed Ahmed Ramadan	5
3	Marwa Gamal Ramadan	5

1. Project Objective:

The aim of this project is to develop a microcontroller-based system to monitor hospital rooms using temperature, humidity, and gas sensors. The system will be able to record temperature and humidity data for a day or several days and display the data in a graph. Additionally, the system will be able to detect smoke using a gas sensor.

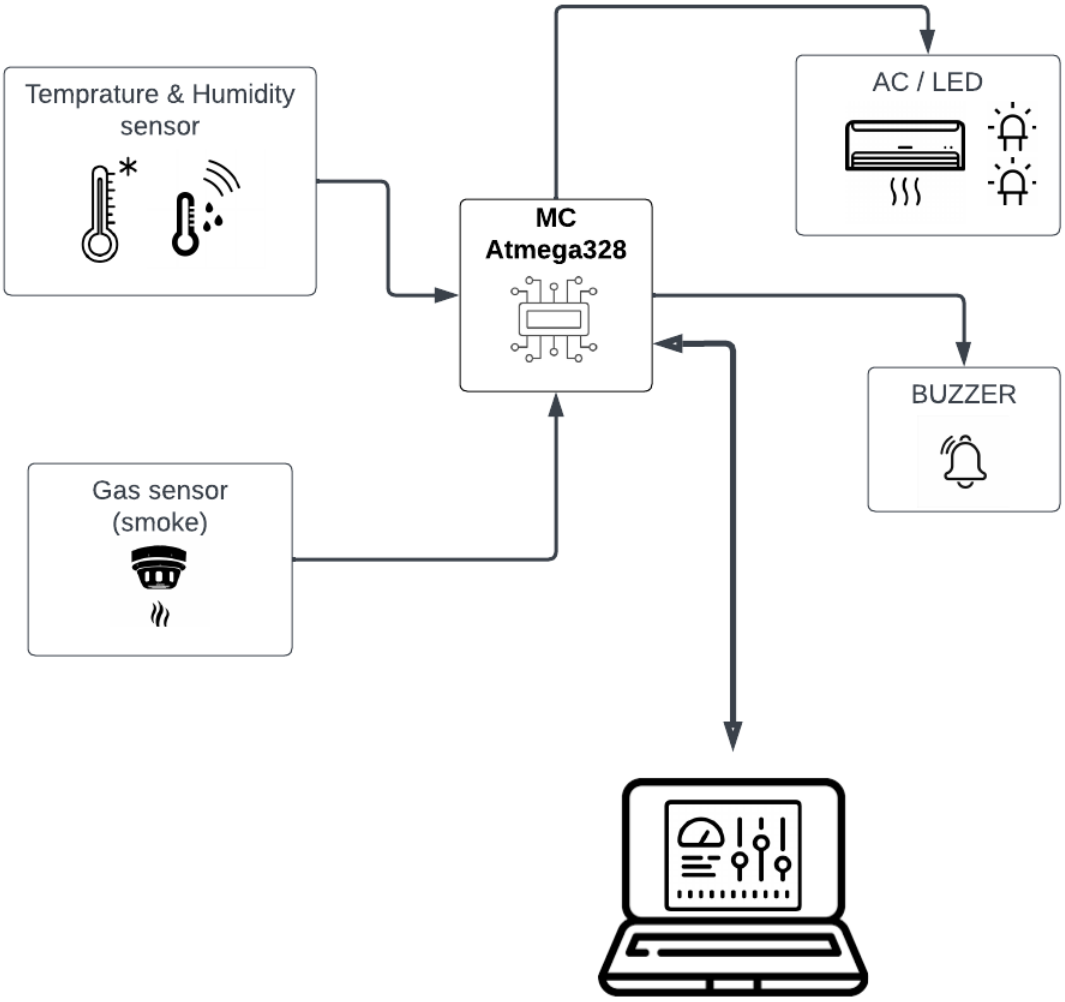
To achieve these objectives, a microcontroller will be chosen and programmed to read data from the temperature and humidity sensors. The data will be stored in memory and displayed in a graph. The control system will adjust the room temperature using an AC or fan based on the sensor readings. The alarm system will detect smoke using a gas sensor. This will help detect smoke and prevent fire incidents, ensuring the safety of everyone in the hospital.

To make the system easy to use, a Graphical User Interface (GUI) will be developed to customize the display of sensor data. This will provide an easy-to-use interface for hospital staff to monitor the room conditions.

The implementation of this system will benefit hospitals by providing a reliable and efficient way to monitor temperature, humidity, and gas levels in hospital rooms. This will help maintain a comfortable and safe environment for patients and staff.

2. System Block Diagram:

2.1. Block Diagram:



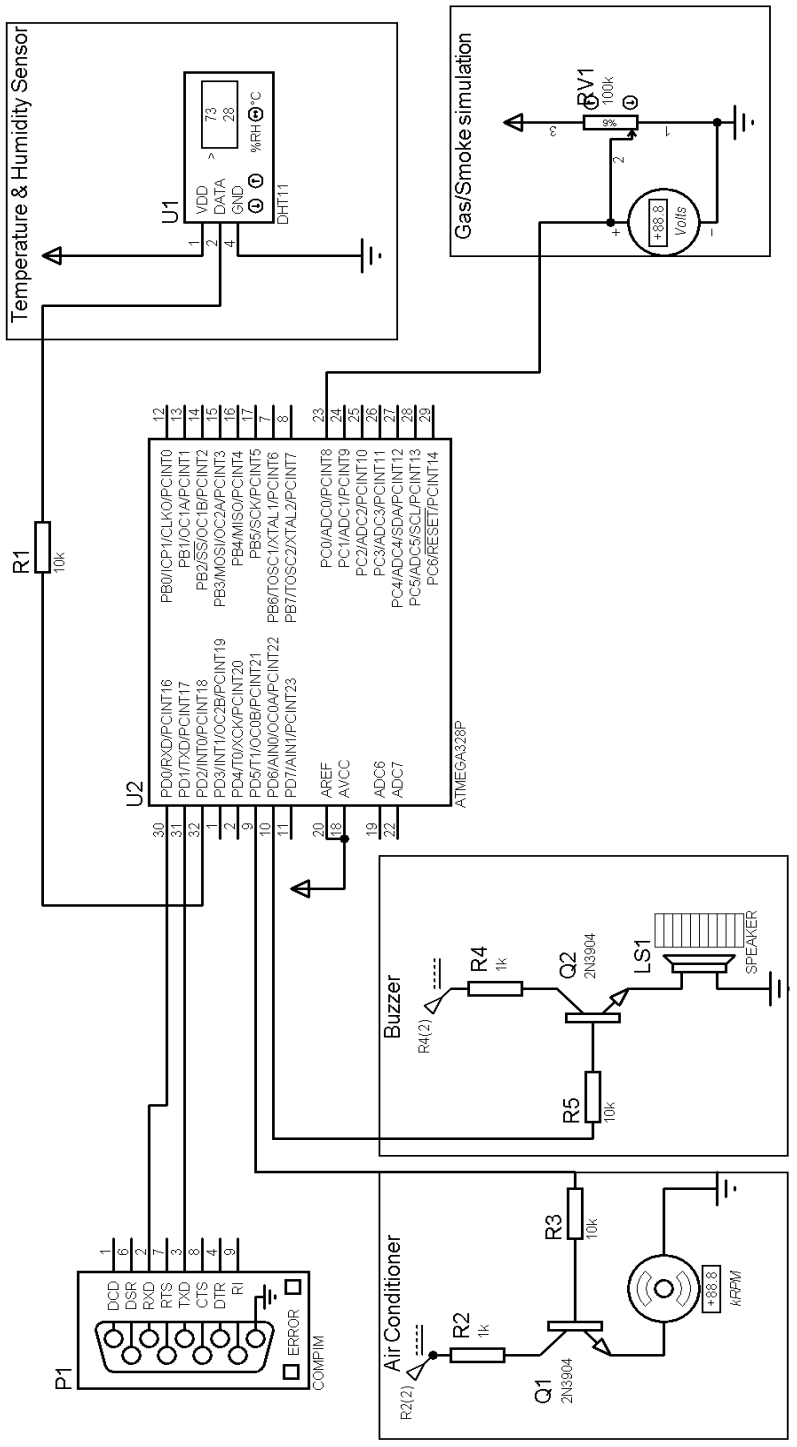
2.2. Block Diagram Description:

This block diagram depicts several sensors that are utilized to monitor the temperature and air quality within a hospital room. A microcontroller, specifically the Atmega 328, is connected to both sensors and actuators. The first sensor used is the DHT11, which is a temperature and humidity sensor. It sends a frame of data containing the temperature and humidity percentage to the microcontroller via asynchronous mode, where the microcontroller sends a start signal consisting of a low pulse followed by a high pulse to initiate communication. The sensor then responds with the data frame, and this process repeats continuously. To ensure proper functioning, a user-defined threshold is set in the GUI application. If the temperature or humidity exceeds this threshold, the microcontroller will automatically send a request for the AC to work, which is indicated by an LED light.

The second sensor utilized is the MQ-2 smoke sensor module, which measures the gas concentration to monitor air quality and detect potential hazards such as gas leaks or smoking near the patient. The sensor sends an analog signal expressing a voltage number equivalent to the gas ratio in the air. To ensure patient safety, a buzzer is used as an alarm to sound if the gas limit exceeds its threshold, which is also user-defined.

The microcontroller is connected to a PC using UART for serial communication to connect with the GUI application. The GUI application displays continuous statistics of the room's air condition and smoke ratio, along with options to turn on/off the AC, buzzer, and adjust the threshold for each parameter measured. Additionally, the GUI displays patient information such as name, room number, and status.

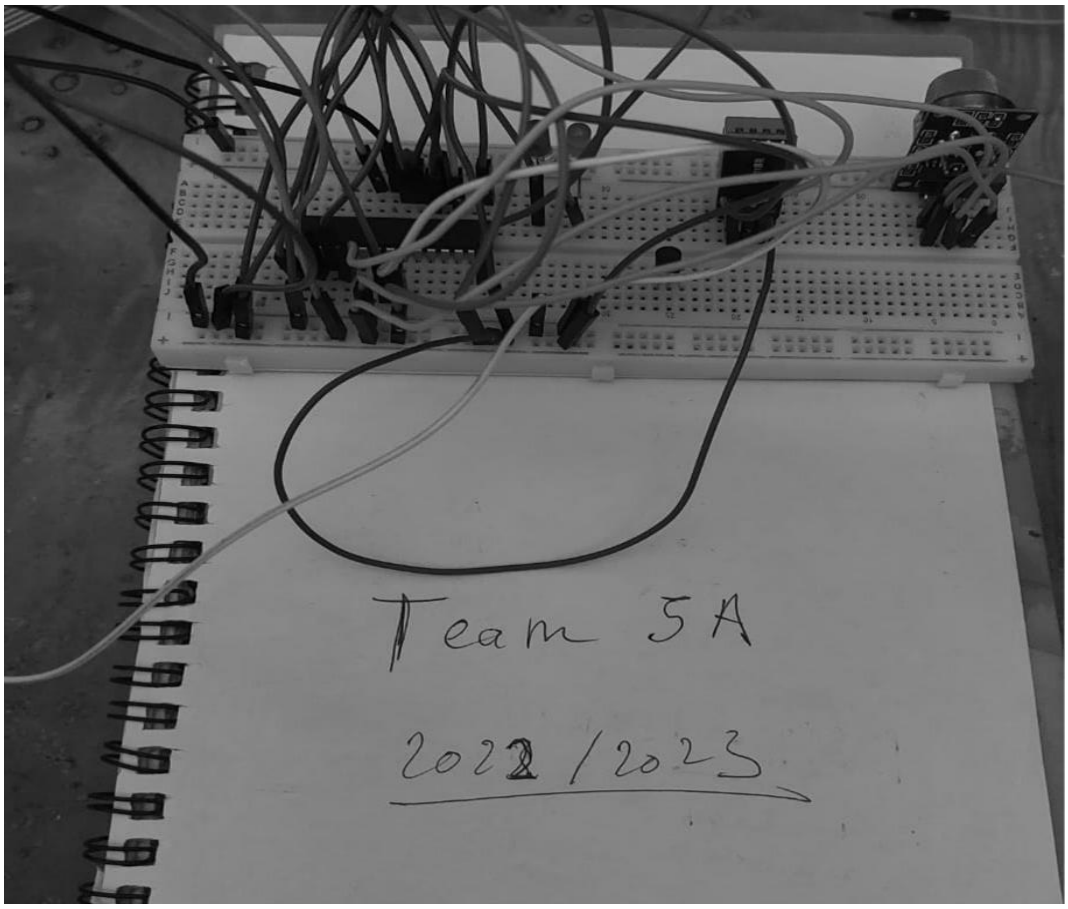
3. Schematic Diagram (Circuit Diagram):

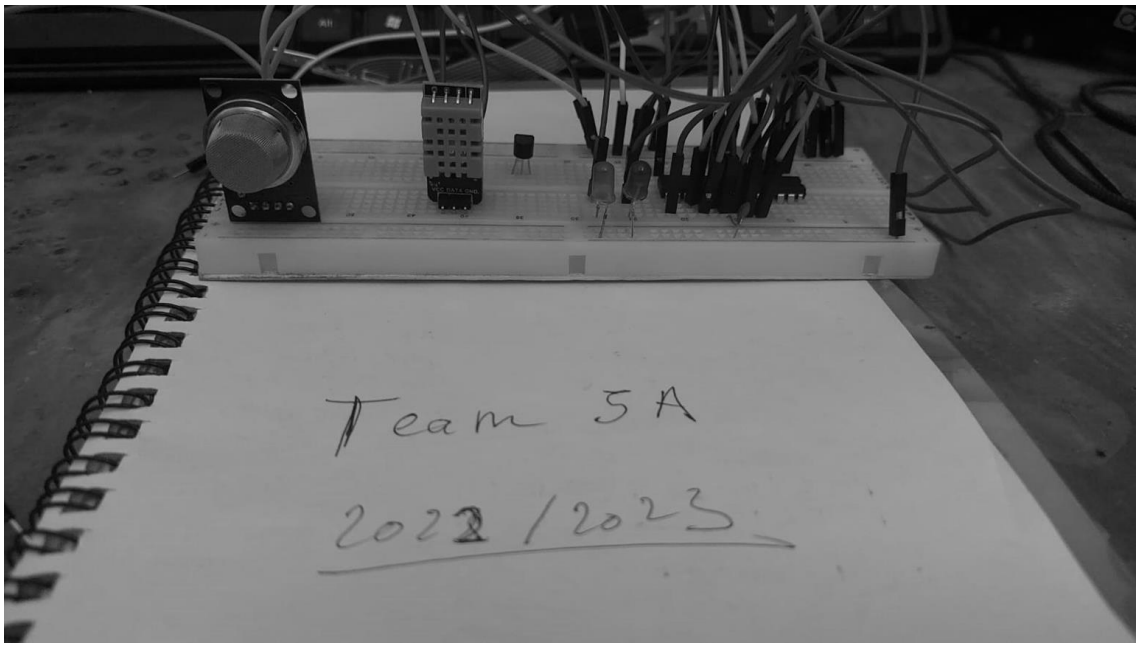


4. List Of Components:

SN	Item Type	Item Code Name	Purpose	Quantity
1	Temperature & Humidity sensor	DHT11	To measure the temperature and humidity for hospital room	1
2	Smoke and LGP gas detector	MQ-2	To detect whether is there any smoke results from fire accidents or cigarettes or any other leakage from various type of gases for patient safety	1
3	Microcontroller	Atmega328-avr controller	This is the main controller	1
4	Buzzer	-	To act as an alarm if there any smoke.	1
5	LED	-	To act as indicator for rise in temperature.	3
6	Bread board	-	To fix our hardware on.	1
7	Jumper wires		For connection.	15

5. Real-Time Hardware Photo:





6. Source Code:

6.1. Hardware-side source code:

main.cpp

```
1.
2.  #define F_CPU 8000000UL
3.  #define BAUD_RATE 9600
4.  #include <avr/io.h>
5.  #include <util/delay.h>
6.  #include <avr/interrupt.h>
7.  #include "UART.h"
8.  #define DHT_PIN 2   // Digital pin connected to the DHT sensor
9.  #define SENSOR_PIN 0 // define the analog input pin for the MQ-2 sensor
10.
11. static volatile uint8_t cmd_buffer[11];
12. volatile uint8_t tx_str[11] = {0};
13.
14. void TX(int temp, int hum, int smoke);
15. void TX_AC(int status);
16. void TX_BZ(int status);
17. void Check_RX();
18. void AC_ON();
19. void AC_OFF();
20. void BUZZER_ON();
21. void BUZZER_OFF();
22. int GetAC();
23. int GetBuzzer();
24.
25. int buzzer_override = 0;
26.
27. void dht_start() {
28.         DDRD |= (1 << DHT_PIN);    // Set the data pin as an output
29.         PORTD &= ~(1 << DHT_PIN); // Set the data pin low
30.         _delay_ms(18);          // Send the start signal
31.         PORTD |= (1 << DHT_PIN);  // Set the data pin high
32.         _delay_us(40);          // Wait for at least 20 microseconds
33.         DDRD &= ~(1 << DHT_PIN);  // Set the data pin as an input
34. }
35.
36. int dht_read(uint8_t* data)
```

```
{
    uint8_t bits[5] = {0};
    uint8_t cnt = 7, idx = 0; // counters // bit and byte. bit is initially set to 7, which means that the first bit read will be the most significant bit of the first byte, and byte is initially set to 0, which means that the first byte read will be stored in buffer[0].
    unsigned int loopCnt = 10000;

    while(!(PIND & (1 << DHT_PIN)))
    {
        if (loopCnt-- == 0)
        {
            return 0;
        }

    }

    loopCnt = 10000;
    while(PIND & (1 << DHT_PIN))
    {
        if (loopCnt-- == 0)
        {
            return 0;
        }

    }

    for (int i=0; i<40; i++)
    {
        int low = 10000;
        while(!(PIND & (1 << DHT_PIN)))
        {
            _delay_us(15);
            low -= 15;
            if(low == 0)
            {
                return 0;
            }

        }
        int high = 10000;
        while(PIND & (1 << DHT_PIN))
        {
```

```
77.                         _delay_us(15);
78.                         high -= 15;
79.                         if(high == 0)
80.                         {
81.                                 return 0;
82.                         }
83.
84.                 }
85.                 if ((10000-high) > 40) bits[idx] |= (1 << cnt);
86.                 if (cnt == 0)   // next byte?
87.                 {
88.                         cnt = 7;    // restart at MSB
89.                         idx++;      // next byte!
90.                 }
91.                 else cnt--;
92.         }
93.         if (bits[4] != (bits[0] + bits[1] + bits[2] + bits[3])) {
94.         return 0; }
95.         data[0] = bits[0]; // decimal part of temperature
96.         data[1] = bits[1]; // integer part of temperature
97.         data[2] = bits[2]; // decimal part of humidity
98.         data[3] = bits[3]; // integer part of humidity
99.         return 1;
100.        }
101.
102.        int main() {
103.                // initialize the analog-to-digital converter (ADC)
104.                ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (0 << ADPS0); // set ADC prescaler to
    64
105.                ADCSRA |= (1 << ADEN); // enable ADC
106.                /* Init UART driver. */
107.                UART_cfg my_uart_cfg;
108.                /* Set USART mode. */
109.                my_uart_cfg.UBRRL_cfg = (BAUD_RATE_VALUE)&0x00FF;
110.                my_uart_cfg.UBRRH_cfg = (((BAUD_RATE_VALUE)&0xFF00)>>8);
111.                my_uart_cfg.UCSRA_cfg = 0;
112.                my_uart_cfg.UCSRB_cfg = (1<<RXEN0) | (1<<TXEN0) | (1<<TXCIE0) |
    (1<<RXCIE0);
113.                my_uart_cfg.UCSRC_cfg = (3<<UCSZ00);
114.                UART_Init(&my_uart_cfg);
115.                sei();
116.                /* Receive the full buffer command. */
```

```
117.                UART_ReceivePayload(cmd_buffer, 10);
118.                DDRD |= (1 << DHT_PIN);   // Set the data pin as an output
119.                PORTD |= (1 << DHT_PIN);  // Set the data pin high
120.                ADMUX &= 0xF0; // clear the analog input pin selection
121.                ADMUX = SENSOR_PIN; // select the sensor pin
122.                DDRB |= 1<<0 | 1<<1;
123.
124.                while (1) {
125.                        uint8_t data[4] = {1,2,3,0};
126.                        // read the analog voltage output of the MQ-2 sensor
127.                        ADCSRA |= (1 << ADSC); // start conversion
128.                        while (ADCSRA & (1 << ADSC)); // wait for conversion to finish
129.                        int sensorValue = ADC; // read the ADC value
130.                        uint8_t temp, hum;
131.                        dht_start();
132.                        if (dht_read(data)) {
133.                                temp = data[2]; //data[1] contains the integer part of the
    temperature and data[0] contains the decimal part of the temperature.
134.                                hum = data[0];
135.                                if(sensorValue >= 1000)
136.                                        sensorValue = 999;
137.                                // Use the temperature and humidity readings
138.                                TX(temp, hum, sensorValue);
139.                        }
140.                        TX_AC(GetAC());
141.                        TX_BZ(GetBuzzer());
142.                        _delay_ms(200);
143.                        Check_RX();
144.                        _delay_ms(2000);
145.                }
146.                return 0;
147.        }
148.
149.        void TX(int temp, int hum, int smoke)
150.        {
151.                //------------------------------------------------------------
152.                sprintf(tx_str, "@%.2i%.3i%.3i;", temp, hum, smoke);
153.                UART_SendPayload(tx_str, 10);
154.                while(0 == UART_IsTxComplete());
155.        }
156.
157.        void TX_AC(int status)
```

```
158.    {
159.        sprintf(tx_str, "@STS:AC%i;", status!=0);
160.        UART_SendPayload(tx_str, 10);
161.        while(0 == UART_IsTxComplete());
162.    }
163.
164.    void TX_BZ(int status)
165.    {
166.        sprintf(tx_str, "@STS:BZ%i;", status!=0);
167.        UART_SendPayload(tx_str, 10);
168.        while(0 == UART_IsTxComplete());
169.    }
170.
171.    void Check_RX()
172.    {
173.        //while(0 == UART_IsRxComplete());
174.        if(UART_IsRxComplete())
175.        {
176.            if(!(cmd_buffer[0] == '@' && cmd_buffer[9] == ';')) return;
177.            if(cmd_buffer[1] == 'A')
178.            {
179.                if(cmd_buffer[2] == '0')
180.                {
181.                    // AC off
182.                    AC_OFF();
183.                }
184.                else
185.                {
186.                    // AC on
187.                    AC_ON();
188.                }
189.            }
190.            else
191.            if(cmd_buffer[1] == 'B')
192.            {
193.
194.                if(cmd_buffer[2] == '0')
195.                {
196.                    // buzzer off
197.                    BUZZER_OFF();
198.                }
199.                else
```

```
                                {
                                        // buzzer on
                                        BUZZER_ON();
                                }
                        }

                        /* Receive the full buffer command. */
                        UART_ReceivePayload(cmd_buffer, 10);
                }

                //address = 0;
        }

        void AC_ON()
        {
                PORTB |= 1<<0;
        }

        void AC_OFF()
        {
                PORTB &= ~(1<<0);
        }

        void BUZZER_ON()
        {
                PORTB |= 1<<1;
        }

        void BUZZER_OFF()
        {
                PORTB &= ~(1<<1);
        }

        int GetAC()
        {
                return PINB & 1<<0;
        }

        int GetBuzzer()
        {
                if(buzzer_override)
                        return 0;
```

```
242.                    return PINB & 1<<1;
243.            }
```

UART.h

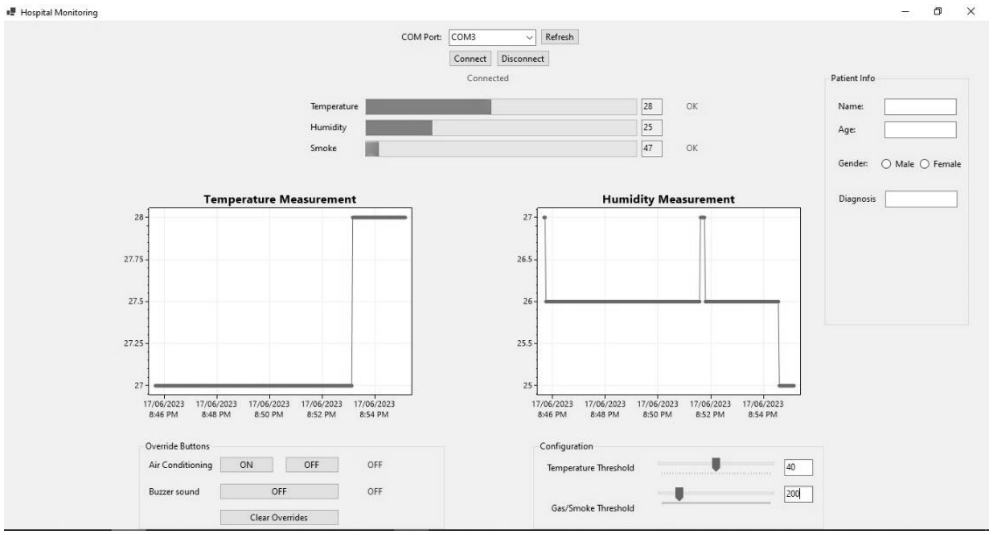
```
1.  #include <stdio.h>
2.
3.  #define BAUD_RATE_VALUE (((F_CPU)/(BAUD_RATE*16UL))-1)
4.  typedef struct {
5.          /* Place here module configuration registers. */
6.          uint8_t UBRRH_cfg;
7.          uint8_t UBRRL_cfg;
8.          uint8_t UCSRA_cfg;
9.          uint8_t UCSRB_cfg;
10.         uint8_t UCSRC_cfg;
11. }UART_cfg;
12. extern void UART_Init(UART_cfg *my_cfg);
13. extern void UART_SendPayload(uint8_t *tx_data, uint16_t len);
14. extern void UART_ReceivePayload(uint8_t *rx_data, uint16_t len);
15. extern uint8_t UART_IsDataAvaiable(void);
16. extern uint8_t UART_IsTxComplete(void);
17. extern uint8_t UART_IsRxComplete(void);
```

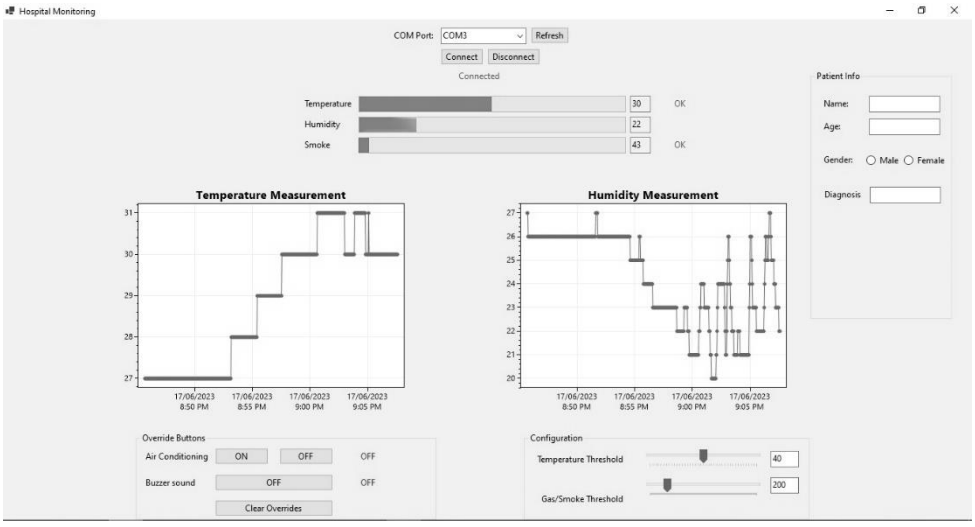
UART.c

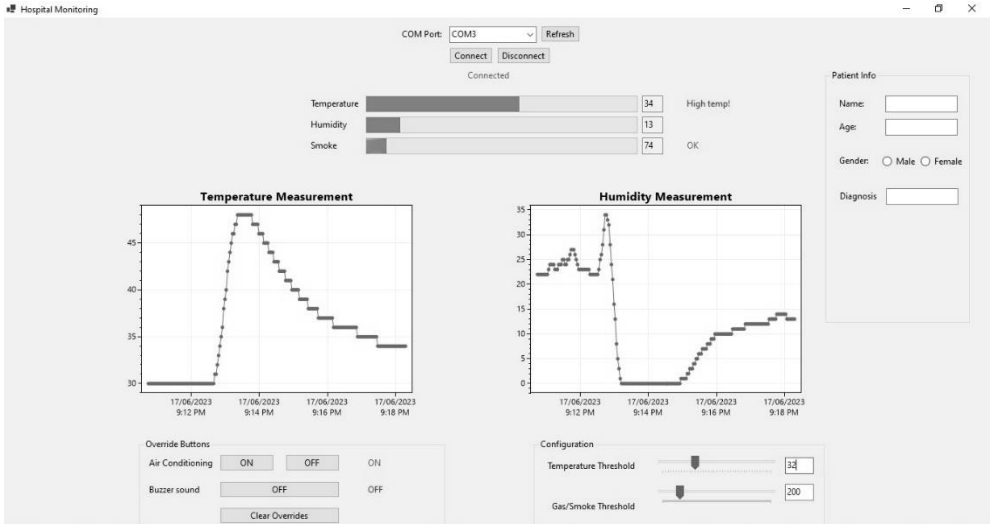
```
1.  #include <avr/io.h>
2.  #include "UART.h"
3.  #include <avr/interrupt.h>
4.  static volatile uint8_t *tx_buffer;
5.  static volatile uint16_t tx_len;
6.  static volatile uint16_t tx_cnt;
7.  static volatile uint8_t *rx_buffer;
8.  static volatile uint16_t rx_len;
9.  static volatile uint16_t rx_cnt;
10. ISR(USART_RX_vect) {
11.         uint8_t rx_data;
12.         cli();
13.         /* Read rx_data. */
14.         rx_data = UDR0;
15.         /* Ignore spaces */
16.         if((rx_cnt < rx_len) && (rx_data != ' ')) {
17.                 rx_buffer[rx_cnt] = rx_data;
18.                 rx_cnt++;
19.         }
```

```
20.         sei();
21. }
22.
23. ISR(USART_TX_vect) {
24.         cli();
25.         tx_cnt++;
26.         if(tx_cnt < tx_len) {
27.                 /* Send next byte. */
28.                 UDR0 = tx_buffer[tx_cnt];
29.         }
30.         sei();
31. }
32. void UART_Init(UART_cfg *my_cfg) {
33.         /* Set baud rate */
34.         UBRR0H = my_cfg->UBRRH_cfg;
35.         UBRR0L = my_cfg->UBRRL_cfg;
36.         UCSR0A = my_cfg->UCSRA_cfg;
37.         UCSR0B = my_cfg->UCSRB_cfg;
38.         UCSR0C = my_cfg->UCSRC_cfg;
39. }
40. void UART_SendPayload(uint8_t *tx_data, uint16_t len) {
41.         tx_buffer = tx_data;
42.         tx_len = len;
43.         tx_cnt = 0;
44.         /* Wait for UDR is empty. */
45.         while(0 == (UCSR0A & (1 << UDRE0)));
46.         /* Send the first byte to trigger the TxC interrupt. */
47.         UDR0 = tx_buffer[0];
48. }
49. void UART_ReceivePayload(uint8_t *rx_data, uint16_t len) {
50.         rx_buffer = rx_data;
51.         rx_len = len;
52.         rx_cnt = 0;
53. }
54. uint8_t UART_IsTxComplete(void) {
55.         return ( (tx_cnt >= tx_len) ? 1 : 0 );
56. }
57. uint8_t UART_IsRxComplete(void) {
58.         return ( (rx_cnt >= rx_len) ? 1 : 0 );
59. }
```

6.2. PC-side source code:







GUI Code:

```
1.  using ScottPlot.Plottable;
2.  using System.IO.Ports;
3.  using System.Media;
4.  using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;
5.
6.  namespace Hospital_Monitoring
7.  {
8.      public partial class Form1 : Form
9.      {
10.         SerialPort port;
11.
12.         Dictionary<DateTime, double> tempValues = new Dictionary<DateTime, double>();
13.         Dictionary<DateTime, double> humValues = new Dictionary<DateTime, double>();
14.         ScatterPlot scatterplot = null;
15.
16.         SoundPlayer Alert = new
        System.Media.SoundPlayer(@"C:\Users\Kirollos\source\repos\Hospital Monitoring\Hospital
        Monitoring\NewMessage.wav");
17.
18.         Queue<string> TXFrameBuffer = new Queue<string>();
19.
20.
21.         int currentTemp, currentHum, currentSmoke;
22.
23.         int tempThreshold;
24.         int smokeThreshold;
25.
26.         bool ACoverride = false, Buzzeroverride = false;
27.
28.         public Form1()
29.         {
30.             InitializeComponent();
31.         }
32.
33.         private void Form1_Load(object sender, EventArgs e)
34.         {
35.             comboBox1.DataSource = SerialPort.GetPortNames();
36.             label2.Text = "Disconnected";
37.             label2.ForeColor = Color.Red;
38.             tempThreshold = int.Parse(textBox4.Text);
```

```
39.         smokeThreshold = int.Parse(textBox5.Text);
40.     }
41.
42.     private void Form1_FormClosing(object sender, FormClosingEventArgs e)
43.     {
44.         if (port != null && port.IsOpen)
45.         {
46.             port.DataReceived -= Port_DataReceived;
47.             port.Close();
48.         }
49.     }
50.
51.     private void button8_Click(object sender, EventArgs e)
52.     {
53.         comboBox1.DataSource = SerialPort.GetPortNames();
54.     }
55.
56.     private void RenderTempGraph()
57.     {
58.         formsPlot.Plot.Clear();
59.
60.         int pointCount = 600;
61.         Random rand = new Random(0);
62.
63.         DateTime[] dates = Enumerable.Range(0, pointCount)
64.                                      .Select(x => DateTime.Now.AddMinutes(x))
65.                                      .ToArray();
66.
67.         // use LINQ and DateTime.ToOADate() to convert DateTime[] to double[]
68.         double[] xs = tempValues.Keys.Select(x => x.ToOADate()).ToArray();
69.
70.         var plt = formsPlot.Plot;
71.         // plot the double arrays using a traditional scatter plot
72.         scatterplot = plt.AddScatter(xs, tempValues.Values.ToArray());
73.
74.         // indicate the horizontal axis tick labels should display DateTime units
75.         plt.XAxis.DateTimeFormat(true);
76.
77.         // add padding to the right to prevent long dates from flowing off the figure
78.         plt.YAxis2.SetSizeLimit(min: 40);
79.
80.         // save the output
```

```
81.         plt.Title("Temperature Measurement");
82.         try { formsPlot.Refresh(); }
83.         catch
84.         {
85.
86.         }
87.     }
88.
89.     private void AddTempData(double temp)
90.     {
91.         tempValues.Add(DateTime.Now, temp);
92.         double[] xs = tempValues.Keys.Select(x => x.ToOADate()).ToArray();
93.     }
94.
95.     private void RenderHumGraph()
96.     {
97.         formsPlot1.Plot.Clear();
98.
99.         int pointCount = 600;
100.                Random rand = new Random(0);
101.
102.                DateTime[] dates = Enumerable.Range(0, pointCount)
103.                                    .Select(x => DateTime.Now.AddMinutes(x))
104.                                    .ToArray();
105.
106.                // use LINQ and DateTime.ToOADate() to convert DateTime[] to double[]
107.                double[] xs = humValues.Keys.Select(x => x.ToOADate()).ToArray();
108.
109.                var plt = formsPlot1.Plot;
110.                // plot the double arrays using a traditional scatter plot
111.                scatterplot = plt.AddScatter(xs, humValues.Values.ToArray());
112.
113.                // indicate the horizontal axis tick labels should display DateTime units
114.                plt.XAxis.DateTimeFormat(true);
115.
116.                // add padding to the right to prevent long dates from flowing off the figure
117.                plt.YAxis2.SetSizeLimit(min: 40);
118.
119.                // save the output
120.                plt.Title("Humidity Measurement");
121.                try { formsPlot1.Refresh(); }
122.                catch
```

```
123.            {
124.
125.            }
126.        }
127.
128.        private void AddHumData(double hum)
129.        {
130.            humValues.Add(DateTime.Now, hum);
131.            double[] xs = humValues.Keys.Select(x => x.ToOADate()).ToArray();
132.        }
133.
134.        private void button10_Click(object sender, EventArgs e)
135.        {
136.            if (port != null && port.IsOpen)
137.            {
138.                port.DataReceived -= Port_DataReceived;
139.                port.Close();
140.                label2.Text = "Disconnected";
141.                label2.ForeColor = Color.Red;
142.            }
143.        }
144.
145.        private void button9_Click(object sender, EventArgs e)
146.        {
147.            if (port != null)
148.                if (port.IsOpen)
149.                    port.Close();
150.            port = new SerialPort(comboBox1.Text, 9600);
151.            port.Open();
152.            if (port.IsOpen)
153.            {
154.                label2.Text = "Connected";
155.                label2.ForeColor = Color.Green;
156.            }
157.            else
158.            {
159.                label2.Text = "Disconnected";
160.                label2.ForeColor = Color.Red;
161.            }
162.            port.DataReceived += Port_DataReceived;
163.        }
164.
```

```
165.        private void Port_DataReceived(object sender, SerialDataReceivedEventArgs e)
166.        {
167.            char[] buffer = new char[10];
168.
169.            int len = 0;
170.            string buff = "";
171.            bool start = false;
172.            bool end = false;
173.            while (true)
174.            {
175.                try
176.                {
177.                    len = port.Read(buffer, 0, 1);
178.                }
179.                catch
180.                {
181.                    start = end = false;
182.                    break;
183.                }
184.                for (int i = 0; i < len; i++)
185.                {
186.                    if (buffer[i] == '@')
187.                        start = true;
188.                    if (start)
189.                        buff += buffer[i];
190.                    if (start && buffer[i] == ';') //end
191.                    {
192.                        end = true;
193.                        break;
194.                    }
195.                }
196.                if (end) break;
197.            }
198.
199.            this.Invoke(() => handleRX(new string(buff)));
200.        }
201.
202.        void port_write(string frame)
203.        {
204.            TXFrameBuffer.Enqueue(frame);
205.        }
206.
```

```
207.        private void handleRX(string frame)
208.        {
209.            if (frame.StartsWith("@STS"))
210.            {
211.                if (frame.StartsWith("@STS:AC"))
212.                {
213.                    int idx = frame.IndexOf("AC") + 2;
214.                    int status = int.Parse(frame.Substring(idx, frame.IndexOf(';') - idx));
215.                    label8.Text = status > 0 ? "ON" : "OFF";
216.                    label8.ForeColor = status > 0 ? Color.Green : Color.Red;
217.                }
218.                if (frame.StartsWith("@STS:BZ"))
219.                {
220.                    int idx = frame.IndexOf("BZ") + 2;
221.                    int status = int.Parse(frame.Substring(idx, frame.IndexOf(';') - idx));
222.                    label9.Text = status > 0 ? "ON" : "OFF";
223.                    label9.ForeColor = status > 0 ? Color.Green : Color.Red;
224.                }
225.                return;
226.            }
227.            int temp = int.Parse(frame.Substring(1, 2));
228.            int humidity = int.Parse(frame.Substring(3, 3));
229.            int smoke = int.Parse(frame.Substring(6, 3));
230.
231.            handleRX(temp, humidity, smoke);
232.        }
233.
234.        private void handleRX(int temp, int humidity, int smoke)
235.        {
236.            AddTempData(temp);
237.            RenderTempGraph();
238.            AddHumData(humidity);
239.            RenderHumGraph();
240.            //double smokevolt = (smoke * 5.0) / 1024;
241.            textBox1.Text = temp.ToString();
242.            textBox2.Text = humidity.ToString();
243.            textBox3.Text = smoke.ToString(); //+ " | " + smokevolt.ToString("0.00") + "V";
244.            progressBar3.Value = temp;
245.            progressBar1.Value = humidity;
246.            progressBar2.Value = smoke * 100 / 900;
247.
248.            currentTemp = temp;
```

```
249.            currentHum = humidity;
250.            currentSmoke = smoke;
251.        }
252.
253.        private void button1_Click(object sender, EventArgs e)
254.        {
255.            port_write("@A1000000;");
256.            ACoverride = true;
257.        }
258.
259.        private void button2_Click(object sender, EventArgs e)
260.        {
261.            port_write("@A0000000;");
262.            ACoverride = true;
263.        }
264.
265.        private void button3_Click(object sender, EventArgs e)
266.        {
267.            port_write("@B0000000;");
268.            Buzzeroverride = true;
269.            alarmcancel = true;
270.            label14.Text = "Smoke detected!";
271.            label14.ForeColor = Color.Red;
272.        }
273.
274.        private void trackBar1_Scroll(object sender, EventArgs e)
275.        {
276.            tempThreshold = trackBar1.Value;
277.            textBox4.Text = trackBar1.Value.ToString();
278.        }
279.
280.        private void textBox4_TextChanged(object sender, EventArgs e)
281.        {
282.            if (textBox4.Text.Length == 0 || int.Parse(textBox4.Text) < 20) return;
283.            if (int.Parse(textBox4.Text) >= 60)
284.                textBox4.Text = "60";
285.            tempThreshold = int.Parse(textBox4.Text);
286.            trackBar1.Value = int.Parse(textBox4.Text);
287.        }
288.
289.        private void trackBar2_Scroll(object sender, EventArgs e)
290.        {
```

```
291.            smokeThreshold = trackBar2.Value;
292.            textBox5.Text = trackBar2.Value.ToString();
293.        }
294.
295.        private void textBox5_TextChanged(object sender, EventArgs e)
296.        {
297.            if (textBox4.Text.Length == 0 || int.Parse(textBox5.Text) < 50) return;
298.            if (int.Parse(textBox5.Text) >= 900)
299.                textBox5.Text = "900";
300.            smokeThreshold = int.Parse(textBox5.Text);
301.            trackBar2.Value = int.Parse(textBox5.Text);
302.        }
303.
304.        private void timer1_Tick(object sender, EventArgs e)
305.        {
306.            if (currentTemp > tempThreshold)
307.            {
308.                label12.Text = "High temp!";
309.                label12.ForeColor = Color.Red;
310.                if (!ACoverride)
311.                    port_write("@A1000000;");
312.            }
313.            else
314.            {
315.                label12.Text = "OK";
316.                label12.ForeColor = Color.Green;
317.                if (!ACoverride)
318.                    port_write("@A0000000;");
319.            }
320.
321.            if (currentSmoke > smokeThreshold)
322.            {
323.                label14.Text = "Smoke detected!";
324.                label14.ForeColor = Color.Red;
325.                if (!Buzzeroverride)
326.                    port_write("@B1000000;");
327.
328.                timer2.Enabled = true;
329.            }
330.            else
331.            {
332.                label14.Text = "OK";
```

```
                label14.ForeColor = Color.Green;
                if (!Buzzeroverride)
                    port_write("@B0000000;");

                timer2.Enabled = false;
                alarmcancel = false;
            }
        }
        private bool alarmcancel = false;
        private void timer2_Tick(object sender, EventArgs e)
        {
            if (alarmcancel) return;
            Alert.Play();
            label14.Text = "Smoke detected!";
            label14.ForeColor = Color.Red;
            label14.Visible = !label14.Visible;

            if (currentSmoke < smokeThreshold)
            {
                label14.Text = "OK";
                label14.ForeColor = Color.Green;
                if (port.IsOpen)
                    port_write("@B0000000;");
                timer2.Enabled = false;
                Buzzeroverride = false;
            }
        }

        private void timer3_Tick(object sender, EventArgs e)
        {
            if (port != null && port.IsOpen && TXFrameBuffer.Count > 0)
            {
                port.Write(TXFrameBuffer.Dequeue());
            }
        }

        private void button4_Click(object sender, EventArgs e)
        {
            ACoverride = false;
            Buzzeroverride = false;
        }
    }}
```