Remote Heart Rate and ECG Monitoring System

1. Abstract and keywords

Abstract :

- Health monitoring is very Important and its related technologies are an usefull and attractive research area.
- The electrocardiogram (ECG) has always been a very important measurement scheme to assess the health of heart and bodyand diagnose cardiovascular diseases.
- The number of ECG monitoring systems is expanding exponentially. And thus is very hard for researchers and healthcare experts to compare, and evaluate systems that serve their needs and fulfill the monitoring requirements.
- Understanding ECG monitoring systems’ components, contexts, features, and challenges. Hence, a generic architectural model for ECG monitoring systems is proposed, an extensive analysis of ECG monitoring systems’ value chain is conducted, and is presented, highlighting challenges and current trends.
- We identify key challenges and give importance of smart monitoring systems that leverage new technologies of web 2.0 and http , including deep learning, artificial intelligence (AI), Big Data and Internet of Things (IoT), to provide efficient, real time , cost-aware, and fully connected monitoring systems for patients and doctors and healt enthusiasts.

Keywords :

1. IOT
2. ESP32
3. AD8232
4. Arduino IDE
5. Web Programming
6. Jumper Wires
7. ECG monitoring system
8. smart monitoring,
9. heart diseases,
10. cardiovascular diseases,

3.1 Number of Modules

1. ESP32 Development Board:
   This board is used to process the ECG signal from the AD8232 module and send the data to the internet through Wi-Fi. It is a low cost and low power development board by espressif systems. The board can easily connect to a wifi or create it's own wifi.

2. HTML/CSS/JavaScript:
   These web technologies are used to create a web page that displays the ECG signal in real-time. The web page continously requests new data from the esp32 server for ecg signals.

3. Web Server:
   A web server is used to host the web page and receive the data from the ESP32. The web server is mobile responsive and can be accessed with any device that as a browser on the local network

4. Chart.js:
   Chart.js is a JavaScript library that can be used to create interactive charts and graphs on a web page. It is one of the simplest visualization libraries for JavaScript, and comes with the following built-in chart types:

5. File Download Module:
   The module helps to download the raw Ecg signals file generated from the esp32 and the ecg sensor. The file could be later used to inference the health or use ML to predict the health of the patient.

3.2 Algorithm

The Algorithm Used is for the backend and the esp32 :

```
1. Connect the Esp32 to Wifi
2. Check for the presence of the Sensor module.
3. Create a temporary storage buffer of ten seconds for raw ecg signals
4. Create a timer of 500 Hertz :
           for each timer interupt:
               check for the current time in milli seconds
               create the index to store the sensor data
               read the sensor data from the ECG sensor

5. Create a esp32 web server and do request mappings for url parameters
6. create a mDNS server for browsers to connect to the esp32 server with local domain name.

7. On request for raw data :
     check if prevoius data time is present :
       if present send a data up to date response.
       Else :
           send the resent data present to the client browser.
```

For Browser.

```
1. Load the chart js and request library's
2. check for the presence of a ecg server on the lan network
3. if server is available fetch the raw signals from the server with get requests.
   Store the data in an array for representation of graph and download of data with file.
4. Using the chart js library draw the graph on the browser window.
5. On user download button clicked:
```

Convert the stored raw ecg data array to blob file data
create an anchor tag with the href ointing to the creted data file
click on the link with program and store the data in a csv file

3.3 Working

Following components are used for this project.

1. Esp 32 Development Board
2. Jumper Wires
3. Ad8232 ECG Sensor
4. Buzzer
5. ECG Electrodes
6. Resistors

- The Iot projecct is used for measuring the ECG signals of a patient.
- This is achived by attaching ECG eletrodes to the patient at appropriate locations the attached electrodes are then connected to the sensor modlue named as ad8232.
- The sensor module is based on high sensitivity operationsal apmlifiers or OpAmps and noice cancellation with active network of resistors and capacitors.
- The AD8232 ECG sensor consists of three electrodes that are attached to the chest to measure the electrical signals produced by the heart. The output from the electrodes is then amplified by an instrumentation amplifier and filtered to remove any noise or interference.
- The filtered signal is then fed into the analog-to-digital converter (ADC) of the ESP32 development board, which converts the analog signal into a digital signal that can be processed by the board.
- The ESP32 development board is programmed to read the digital ECG signal and send it to a web server through Wi-Fi. The data can be transmitted using protocols such as HTTP, depending on the specific implementation.
- On the web server, the data can be analyzed and displayed using various web technologies such as HTML, CSS, JavaScript, and Chart.js. Real-time updates can be achieved by using a combination of server-side programming and client-side programming (using JavaScript and AJAX).
- The AD8232 ECG sensor measures the electrical activity of the heart, the ESP32 development board processes the ECG signal and transmits it to a web server, and the web server analyzes and displays the data in real-time using various web technologies. Such as Chart.js graph
- The data is then made available to the patient in a raw csv file for further utilization, analysis and experimentations.

Esp32 Source Code :

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>


#define LED 13
int IRAM_ATTR ECG_RAW[10][500];
// reading from heart sensor.
hw_timer_t *My_timer = NULL;
void IRAM_ATTR onTimer(){
  //Read value.
  int sec = millis() / 1000;
  int arrayInd = sec%10;
  int readingInd = (millis()%1000)/2;
  ECG_RAW[arrayInd][readingInd] = analogRead(34); // ecg pin attached to pin 34.
}

// web server
const char *ssid = "SS";
const char *password = "@12345678";

WebServer server(80);

const int led = 2;

void setup(void) {
  pinMode(led, OUTPUT);
  digitalWrite(led, 0);
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");

  //set up interupts for sensor reading.
  pinMode(LED, OUTPUT);
  My_timer = timerBegin(0, 80, true);
  timerAttachInterrupt(My_timer, &onTimer, true);
  timerAlarmWrite(My_timer, 2000, true);
  timerAlarmEnable(My_timer); //Just Enable

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (MDNS.begin("ecg")) {
    Serial.println("MDNS responder started");
  }
  server.enableCORS(true);
  server.enableCrossOrigin(true);
  server.on("/", handleRoot);
  server.on("/raw", sendECG);
  server.on("/inline", []() {
    server.send(200, "text/plain", "this works as well");
  });
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("HTTP server started");
}

void loop(void) {
  server.handleClient();
  delay(2);//allow the cpu to switch to other tasks
}

void handleRoot() {
  digitalWrite(led, 1);
  char temp[400];
  int sec = millis() / 1000;
  int min = sec / 60;
  int hr = min / 60;

  snprintf(temp, 400,

           "<html>\
  <head>\
    <meta http-equiv='refresh' content='1'/>\
    <title>ECG SYSTEM</title>\
    <style>\
      body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif; Color: #000088; }\
    </style>\
  </head>\
  <body>\
    <h1>ECG SENSOR</h1>\
    <p>Uptime: %02d:%02d:%02d</p>\
  </body>\
</html>",

           hr, min % 60, sec % 60
          );
  server.send(200, "text/html", temp);
  digitalWrite(led, 0);
```

```
}

void handleNotFound() {
  digitalWrite(led, 1);
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";

  for (uint8_t i = 0; i < server.args(); i++) {
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }

  server.send(404, "text/plain", message);
  digitalWrite(led, 0);
}


void sendECG(){
  int sec = millis() / 1000;
  int arrayInd = sec%10;
  int newData = 1;
  if(server.args() > 0){
    int prevSec =  server.arg(0).toInt();
    Serial.println(String(prevSec)+" <-> "+String(arrayInd));
    if(prevSec == arrayInd){
      Serial.println("same data "+String(sec));
      newData = 0;
    }else{
      arrayInd = (prevSec+1)%10;
    }
  }
  else if(arrayInd>0){ arrayInd--;}
  else{arrayInd = 9;}
  String out = "{\"sec\":"+String(arrayInd)+",";
  out+="\"newData\":"+String(newData);
  if(newData==0){
    out+= "}";
  }else{
    out += ",\"data\":[";
    for(int i = 0 ; i < 499 ; i++){
      out += String(ECG_RAW[arrayInd][i])+",";
    }out += String(ECG_RAW[arrayInd][499])+"]}  ";
  }

  server.send(200, "application/json", out);
}
```