به نام خدا

گزارش پروژه درس سیستم‌های نهفته

❤️با تشکر ویژه از آقای اوستاد و آقای براتی❤️

نام استاد درس: دکتر انصاری

نام دستیار آموزشی درس: جناب آقای الیاس اوستاد

آنیتا علیخانی - سیدامیرعلی مقدسی - مهدی داودزاده

تیر 1403

فهرست مطالب

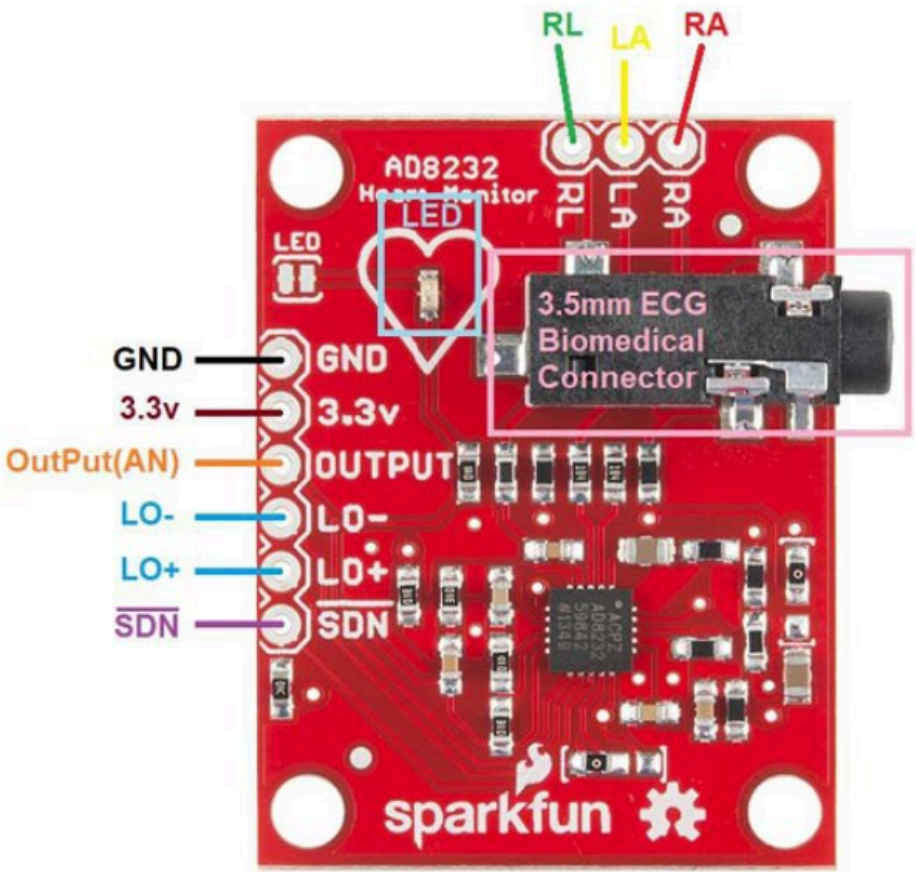
ECG and LCD

Pinout

نمایشگر 3.5 اینچی RPi LCD V3.0 یک نمایشگر چندکاره است. این مدل خاص که با PCB آبی رنگ و به همراه رزولوشن 480x320 شناخته می‌شود، دارای قابلیت صفحه لمسی است و از طریق رابط SPI (رابط سریال محیطی) ارتباط برقرار می‌کند.

از بالای کانکتور در سمت چپ نمایشگر، اولین پین با علامت +5V مشخص شده است. این پین برق مورد نیاز 5 ولت را برای کارکرد نمایشگر فراهم می‌کند. پین بعدی که در زیر آن قرار دارد با علامت 0V یا GND مشخص شده است که اتصال زمین مورد نیاز برای تکمیل مدار و اطمینان از تحویل صحیح برق به ماژول نمایشگر است.

در ادامه، پین‌های مرتبط با رابط ارتباطی SPI قرار دارند. پینی که با DC (داده/فرمان) مشخص شده است برای تمایز بین سیگنال‌های داده و فرمان در حین ارتباط با نمایشگر استفاده می‌شود. پین RST برای بازنشانی نمایشگر استفاده می‌شود و آن را به حالت اولیه بازمی‌گرداند. پین CS (انتخاب چیپ) برای انتخاب ماژول نمایشگر برای ارتباط استفاده می‌شود و اطمینان می‌دهد که دستگاه صحیح در باس SPI آدرس‌دهی شده است.
پایین‌تر، پین MOSI (خروجی مستر، ورودی اسلیو) برای ارتباط SPI است و اجازه می‌دهد داده‌ها از برد مورد نظر به نمایشگر ارسال شوند. پین MISO (ورودی مستر، خروجی اسلیو) توسط قابلیت صفحه لمسی نمایشگر برای ارسال داده‌ها به برد مورد نظر استفاده می‌شود، هرچند اگر قابلیت صفحه لمسی مورد استفاده نباشد نیازی به اتصال آن نیست.
پین بعدی SCK (ساعت سریال) است که سیگنال ساعت لازم برای همگام‌سازی انتقال داده در باس SPI را فراهم می‌کند. ماژول نمایشگر دارای یک پین زمین دیگر با علامت 0V یا GND است تا از اتصال صحیح زمین و یکپارچگی سیگنال اطمینان حاصل کند.
در پایین کانکتور، پین T_CS قرار دارد که مخفف انتخاب چیپ صفحه لمسی است. این پین به طور خاص برای مدیریت قابلیت صفحه لمسی استفاده می‌شود. اگر صفحه لمسی مورد نیاز نباشد، می‌توان آن را به 3.3 ولت وصل کرد تا غیر فعال شود.



ماژول AD8232 یک ماژول مانیتورینگ قلب است که برای اندازه‌گیری سیگنال‌های الکتروکاردیوگرام (ECG) طراحی شده است. در تصویر بالا، پین‌ها و بخش‌های مختلف این ماژول با برچسب‌های مختلف مشخص شده‌اند که در ادامه به توضیح هر یک از آن‌ها می‌پردازیم.
در سمت چپ ماژول، پین‌های مربوط به تغذیه و خروجی‌ها قرار دارند. پین GND که با رنگ مشکی مشخص شده است، پین زمین است که برای تکمیل مدار و اطمینان از عملکرد صحیح ماژول به کار می‌رود. پین 3.3V که با رنگ قرمز مشخص شده است، برای تأمین ولتاژ 3.3 ولت به ماژول استفاده می‌شود.
پین OutPut یا AN که با رنگ نارنجی مشخص شده است، خروجی آنالوگ سیگنال ECG را فراهم می‌کند که می‌توان آن را به یک میکروکنترلر یا دستگاه‌های دیگر برای تحلیل بیشتر متصل کرد. پین LO- و LO+ که به ترتیب با رنگ‌های آبی و فیروزه‌ای مشخص شده‌اند، پین‌های مربوط به تشخیص لید آف هستند که در صورت قطع شدن الکترودها سیگنال خاصی را ارسال می‌کنند.

پین LO خروجی مشترک برای تشخیص لید آف است. پین SDN که با رنگ بنفش مشخص شده است، پین شات‌دان ماژول است که می‌توان با اعمال ولتاژ مناسب ماژول را به حالت کم مصرف منتقل کرد.
در قسمت بالای ماژول، سه پین RA، LA و RL قرار دارند که به ترتیب با رنگ‌های قرمز، زرد و سبز مشخص شده‌اند. این پین‌ها برای اتصال الکترودهای ECG به نقاط مختلف بدن بیمار به کار می‌روند. پین RA به الکترود سمت راست بازو، پین LA به الکترود سمت چپ بازو و پین RL به الکترود روی پای راست بیمار متصل می‌شود.
در بخش میانی ماژول، یک کانکتور 3.5 میلی‌متری برای اتصال کابل‌های الکترودهای ECG قرار دارد که با برچسب "3.5mm ECG Biomedical Connector" مشخص شده است. همچنین در نزدیکی این کانکتور، یک LED با برچسب LED قرار دارد که وضعیت عملکرد ماژول را نشان می‌دهد.

توضیحات کد

LCD_ECG.ino

```ino
#include <SPI.h>
#include <TFT_eSPI.h>

TFT_eSPI tft = TFT_eSPI();

#define LO_PLUS 26 // Leads off detection pin LO+
#define LO_MINUS 27 // Leads off detection pin LO-
#define OUTPUT_PIN 35 // Analog output from AD8232

const int ecgBufferSize = 120;
int ecgBuffer[ecgBufferSize];
int ecgIndex = 0;

void setup() {
  Serial.begin(9600);

  pinMode(LO_PLUS, INPUT_PULLUP);
  pinMode(LO_MINUS, INPUT_PULLUP);
  pinMode(OUTPUT_PIN, INPUT);

  tft.init();
  tft.setRotation(1);
  tft.fillScreen(TFT_BLACK);

  int screenWidth = tft.width();
  int screenHeight = tft.height();

  tft.drawLine(0, screenHeight / 1.7, screenWidth, screenHeight / 1.7, TFT_WHITE);

  int lowerPartY = screenHeight / 1.7 + 10;
```

در این بخش، به توضیح قسمت‌های مختلف کد بر اساس شماره خطوط می‌پردازیم.

در خطوط 1 و 2، کتابخانه‌های مورد نیاز برای استفاده از SPI و TFT eSPI وارد شده‌اند. کتابخانه SPI.h برای ارتباط SPI و TFT_eSPI.h برای کنترل نمایشگر TFT مورد استفاده قرار می‌گیرند.

در خط 4، یک شیء از کلاس TFT_eSPI به نام tft ایجاد می‌شود. این شیء برای کنترل نمایشگر TFT به کار می‌رود.

در خطوط 6 تا 8، پین‌های مربوط به ماژول AD8232 تعریف می‌شوند. پین 26 برای تشخیص لید آف مثبت (LO+)، پین 27 برای تشخیص لید آف منفی (LO-) و پین 35 برای خروجی آنالوگ سیگنال ECG از ماژول AD8232 استفاده می‌شوند.

در خطوط 10 تا 12، اندازه بافر سیگنال ECG تعریف شده و یک آرایه به اندازه 120 برای ذخیره‌سازی داده‌های ECG و یک ایندکس برای مدیریت این داده‌ها ایجاد می‌شود.

در خطوط 14 تا 28، تابع setup() قرار دارد. این تابع شامل تنظیمات اولیه برای ارتباط سریال با سرعت 9600 (خط 15)، پیکربندی پین‌های LO+ و LO- به عنوان ورودی با مقاومت کششی داخلی (خطوط 17 و 18) و پین خروجی آنالوگ به عنوان ورودی (خط 19) است.

در ادامه، نمایشگر TFT مقداردهی اولیه شده و جهت نمایش آن تنظیم می‌شود (خطوط 21 و 22). سپس صفحه نمایش با رنگ مشکی پر می‌شود (خط 23).

در خطوط 25 و 26، عرض و ارتفاع صفحه نمایش TFT به متغیرهای screenWidth و screenHeight اختصاص داده می‌شود.

در خط 28، یک خط افقی سفید رنگ روی صفحه نمایش کشیده می‌شود تا به عنوان یک مرجع بصری برای سیگنال ECG عمل کند.

در خط 30، مختصات Y بخش پایین‌تر صفحه نمایش محاسبه و به متغیر lowerPartY اختصاص داده می‌شود تا بتوان از آن برای نمایش سیگنال‌های ECG استفاده کرد.

```
31
32    tft.setTextColor(TFT_SKYBLUE, TFT_BLACK);
33    tft.setTextSize(2);
34    tft.setCursor(10, lowerPartY);
35    tft.println("SpO2(%)");
36    tft.setTextSize(4);
37    tft.setCursor(10, lowerPartY + 30);
38    tft.println("99");
39
40    tft.setTextColor(TFT_SKYBLUE, TFT_BLACK);
41    tft.setTextSize(2);
42    tft.setCursor(screenWidth / 3 + 10, lowerPartY);
43    tft.println("Pr(bpm)");
44    tft.setTextSize(4);
45    tft.setCursor(screenWidth / 3 + 10, lowerPartY + 30);
46    tft.println("80");
47
48    tft.setTextColor(TFT_YELLOW, TFT_BLACK);
49    tft.setTextSize(2);
50    tft.setCursor(2 * screenWidth / 3 + 10, lowerPartY);
51    tft.println("TEMP(°C)");
52    tft.setTextSize(4);
53    tft.setCursor(2 * screenWidth / 3 + 10, lowerPartY + 30);
54    tft.println("36.8");
55
56    for (int i = 0; i < ecgBufferSize; i++) {
57      ecgBuffer[i] = screenHeight / 2;
58    }
59  }
60
```

در این بخش از کد، تنظیمات اولیه و نمایش اطلاعات بر روی صفحه نمایش TFT انجام می‌شود. در ادامه هر بخش را بر اساس شماره خطوط توضیح می‌دهم.

در خطوط 31 و 32، رنگ متن و پس‌زمینه برای نمایش مقدار SpO2 تنظیم می‌شود. متن با رنگ آبی آسمانی و پس‌زمینه با رنگ سیاه تنظیم می‌شود.

در خطوط 33 و 34، اندازه متن برای نمایش مقدار SpO2 به 2 تنظیم شده و موقعیت متن با استفاده از setCursor در مختصات (10, lowerPartY) قرار می‌گیرد. سپس در خط 35، عبارت "Spo2(%)" بر روی صفحه نمایش چاپ می‌شود.

در خطوط 36 تا 38، اندازه متن به 4 تغییر می‌کند و مقدار SpO2 که برابر با 99 است در مختصات (10, lowerPartY + 30) چاپ می‌شود.

در خطوط 40 و 41، رنگ متن و پس‌زمینه برای نمایش مقدار ضربان قلب (Pr) دوباره به رنگ آبی آسمانی و پس‌زمینه سیاه تنظیم می‌شود. اندازه متن به 2 تغییر می‌کند و موقعیت متن در مختصات (screenWidth / 3 + 10, lowerPartY) تنظیم می‌شود. سپس در خط 44، عبارت مورد نظر چاپ می‌شود.

در خطوط 45 تا 47، اندازه متن به 4 تغییر می‌کند و مقدار ضربان قلب که برابر با 80 است در مختصات (screenWidth / 3 + 10, lowerPartY + 30) چاپ می‌شود.

در خطوط 49 و 50، رنگ متن و پس‌زمینه برای نمایش دما (TEMP) به رنگ زرد و پس‌زمینه سیاه تنظیم می‌شود. اندازه متن به 2 تغییر می‌کند و موقعیت متن در مختصات (screenWidth / 3 + 10, lowerPartY * 2) تنظیم می‌شود. سپس در خط 53، عبارت مورد نظر چاپ می‌شود.

در خطوط 54 تا 56، اندازه متن به 4 تغییر می‌کند و مقدار دما که برابر با 36.8 است در مختصات (screenWidth / 3 + 10, lowerPartY + 30 * 2) چاپ می‌شود.

در خطوط 58 تا 60، یک حلقه for تعریف شده است که بافر سیگنال ECG را با مقدار screenHeight / 2 مقداردهی اولیه می‌کند.

```
60
61  void loop() {
62    float sensorValue = 0;
63    if ((digitalRead(LO_PLUS) == HIGH) || (digitalRead(LO_MINUS) == HIGH)) {
64      Serial.println("Leads Off!");
65      sensorValue = 0;
66    } else {
67      sensorValue = analogRead(OUTPUT_PIN);
68      Serial.println(sensorValue);
69    }
70
71    int ecgY = map(sensorValue, 0, 4095, tft.height() / 1.7, 0);
72
73    ecgBuffer[ecgIndex] = ecgY;
74    ecgIndex = (ecgIndex + 1) % ecgBufferSize;
75
76    tft.fillRect(0, 0, tft.width(), tft.height() / 1.7, TFT_BLACK);
77
78    for (int i = 1; i < ecgBufferSize; i++) {
79      int x0 = (i - 1) * (tft.width() / ecgBufferSize);
80      int y0 = ecgBuffer[(ecgIndex + i - 1) % ecgBufferSize];
81      int x1 = i * (tft.width() / ecgBufferSize);
82      int y1 = ecgBuffer[(ecgIndex + i) % ecgBufferSize];
83      tft.drawLine(x0, y0, x1, y1, TFT_GREEN);
84    }
85
86    delay(10);
87  }
```

در این بخش از کد، عملکرد اصلی ماژول AD8232 و نمایش داده‌های ECG بر روی نمایشگر TFT پیاده‌سازی شده است. در ادامه، هر بخش از کد را بر اساس شماره خطوط توضیح می‌دهم.

در خطوط 61 و 62، متغیری برای ذخیره مقدار سنسور به نام sensorValue با مقدار اولیه 0 تعریف می‌شود.

در خطوط 63 تا 66، بررسی می‌شود که آیا لیدهای ECG قطع شده‌اند یا خیر. اگر یکی از پین‌های LO_PLUS یا LO_MINUS مقدار HIGH داشته باشد، این به معنی قطع شدن لیدها است و عبارت "Leads Off!" بر روی سریال مانیتور چاپ می‌شود و مقدار sensorValue برابر با 0 قرار می‌گیرد.

در خطوط 67 تا 70، اگر لیدها متصل باشند، مقدار سنسور از پین آنالوگ OUTPUT_PIN خوانده می‌شود و بر روی سریال مانیتور چاپ می‌شود.

در خط 71، مقدار خوانده شده از سنسور (که بین 0 و 4095 است) با استفاده از تابع map به مقدار ecgY تبدیل می‌شود که در محدوده ارتفاع نمایشگر قرار دارد.

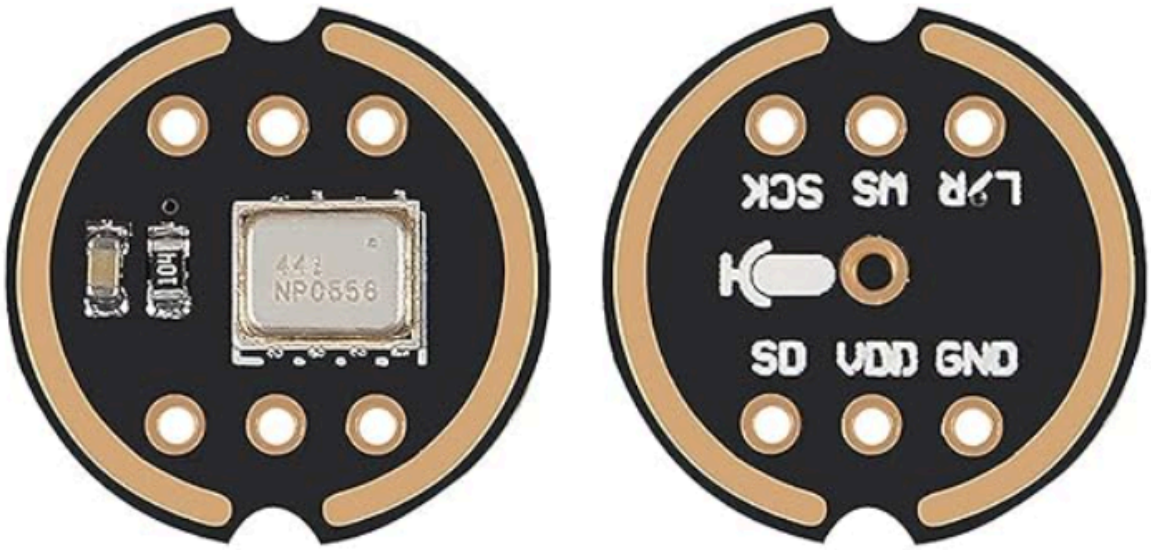
در خطوط 73 و 74، مقدار ecgY در آرایه ecgBuffer ذخیره می‌شود و ایندکس ecgIndex به روز رسانی می‌شود تا به مقدار بعدی در آرایه اشاره کند.

در خط 76، یک مستطیل پر شده به رنگ سیاه بر روی نمایشگر کشیده می‌شود تا قسمت نمایش سیگنال ECG پاک شود.

در خطوط 78 تا 85، یک حلقه for تعریف شده است که از 1 تا اندازه بافر ecgBufferSize تکرار می‌شود. در داخل حلقه، مقادیر x و y برای دو نقطه پیاپی از سیگنال ECG محاسبه می‌شوند و خطی بین این دو نقطه با استفاده از تابع drawLine کشیده می‌شود. رنگ خط، سبز تعیین شده است.
در خط 87، یک تاخیر 10 میلی‌ثانیه‌ای قرار داده شده است تا سرعت به روز رسانی نمایشگر کنترل شود.
این بخش از کد به شما کمک می‌کند که داده‌های ECG را از ماژول AD8232 خوانده و بر روی نمایشگر TFT نمایش دهید. این نمایش به صورت یک سیگنال پیوسته ECG بر روی نمایشگر نشان داده می‌شود.

Microphone

Pinout



ماژول میکروفون INMP441 یک میکروفون دیجیتال MEMS با قابلیت خروجی I2S است که برای ضبط صدا طراحی شده است. در تصویر بالا، دو نمای مختلف از ماژول نشان داده شده است که پین‌های مختلف آن را نمایش می‌دهد.
در سمت راست تصویر، پین‌های ماژول مشخص شده‌اند. پین LR برای انتخاب کانال چپ یا راست میکروفون استفاده می‌شود. اگر این پین به زمین (GND) متصل شود، میکروفون به عنوان کانال چپ عمل می‌کند و اگر به

ولتاژ تغذیه (VDD) متصل شود، به عنوان کانال راست عمل می‌کند. پین WS، پین Word Select یا Frame Sync است که برای همگام‌سازی داده‌های I2S استفاده می‌شود. پین SCK، پین Serial Clock است که ساعت سریال برای پروتکل I2S را فراهم می‌کند. پین SD، پین Serial Data است که داده‌های صوتی دیجیتال از طریق آن ارسال می‌شوند. پین VDD برای تغذیه ولتاژ ماژول استفاده می‌شود و معمولاً ولتاژ تغذیه بین 1.8 ولت تا 3.3 ولت است. پین GND زمین (GND) ماژول است که برای تکمیل مدار و اطمینان از عملکرد صحیح ماژول استفاده می‌شود.

توضیحات کد

Microphone.ino

```
#include <driver/i2s.h>
#include <WebServer.h>
#include <WiFi.h>
#include <SPIFFS.h>

#define I2S_WS 25
#define I2S_SD 32
#define I2S_SCK 14
#define I2S_PORT I2S_NUM_0
#define I2S_SAMPLE_RATE   (16000)
#define I2S_SAMPLE_BITS   (16)
#define I2S_READ_LEN      (16 * 1024)
#define RECORD_TIME       (20) //Seconds
#define I2S_CHANNEL_NUM   (1)
#define FLASH_RECORD_SIZE (I2S_CHANNEL_NUM * I2S_SAMPLE_RATE * I2S_SAMPLE_BITS / 8 * RECORD_TIME)

const char* ssid = "";
const char* password = "";

WebServer server(80);

File file;
const char filename[] = "/recording.wav";
const int headerSize = 44;

void setup() {
  Serial.begin(115200);
  SPIFFSInit();
  i2sInit();
  // xTaskCreate(i2s_adc, "i2s_adc", 1024 * 2, NULL, 1, NULL);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
```

این کد برای ضبط صدای میکروفون دیجیتال INMP441 و ذخیره آن به صورت فایل WAV استفاده می‌شود. در ادامه، بخش‌های مختلف کد را بر اساس شماره خطوط توضیح می‌دهم.

در خطوط 1 تا 4، کتابخانه‌های مورد نیاز برای اجرای برنامه وارد شده‌اند. driver/i2s.h برای ارتباط I2S، WebServer.h برای راه‌اندازی سرور وب، WiFi.h برای اتصال به شبکه Wi-Fi و SPIFFS.h برای مدیریت فایل‌های سیستم استفاده می‌شوند.

در خطوط 6 تا 15، تعدادی ماکرو تعریف شده‌اند که تنظیمات مختلف I2S و ضبط صدا را تعیین می‌کنند. I2S_WS، I2S_SD و I2S_SCK به ترتیب پین‌های Word Select، Serial Data و Serial Clock را مشخص می‌کنند. I2S_PORT پورت I2S را تنظیم می‌کند. I2S_SAMPLE_RATE نرخ نمونه‌برداری (16000 هرتز) را تعیین می‌کند. I2S_SAMPLE_BITS عمق بیت نمونه (16 بیت) را مشخص می‌کند. I2S_READ_LEN طول خواندن داده‌ها (16 کیلوبایت) را تعیین می‌کند. RECORD_TIME مدت زمان ضبط (20 ثانیه) را مشخص می‌کند. I2S_CHANNEL_NUM تعداد کانال‌ها (1 کانال) را تنظیم می‌کند. FLASH_RECORD_SIZE اندازه فایل ضبط شده را محاسبه می‌کند.

در خطوط 16 تا 18، اطلاعات مربوط به شبکه Wi-Fi شامل SSID و رمز عبور تعیین شده‌اند. سپس، یک سرور وب با پورت 80 ایجاد می‌شود. همچنین، متغیرهای file برای مدیریت فایل ضبط شده، filename برای نام فایل ضبط شده و headerSize برای اندازه هدر فایل WAV تعریف شده‌اند.

تابع setup() که از خط 26 شروع میشود، شامل تنظیمات اولیه برای راه‌اندازی سریال، SPIFFS و I2S است. در خط 27، ارتباط سریال با سرعت 115200 بیت بر ثانیه شروع می‌شود. در خط 28، SPIFFS راه‌اندازی می‌شود. در خط 29، I2S مقداردهی اولیه می‌شود. سپس در خطوط 31 تا 37، ارتباط Wi-Fi با استفاده از SSID و رمز عبور مشخص شده آغاز می‌شود و تا زمانی که اتصال برقرار نشود، پیام "Connecting to WiFi..." به همراه نقاط متوالی بر روی سریال مانیتور چاپ می‌شود. پس از اتصال، پیام "Connected to WiFi" چاپ می‌شود.

```
38    Serial.println();
39    Serial.print("Connected! IP address: ");
40    Serial.println(WiFi.localIP());
41
42    server.on("/", HTTP_GET, []() {
43      Serial.println("Serving home page");
44      server.send(200, "text/html", "<h1>ESP32 File Server</h1><a href=\"/download\">Download recording.wav</a>");
45    });
46
47    server.on("/download", HTTP_GET, []() {
48      Serial.println("Download request received");
49      File file = SPIFFS.open("/recording.wav", "r");
50      if (!file) {
51        Serial.println("File not found");
52        server.send(404, "text/plain", "File not found");
53        return;
54      }
55      //server.streamFile(file, "audio/wav");
56      Serial.println("File found, streaming...");
57      server.streamFile(file, "audio/wav");
58      file.close();
59    });
60
61    server.begin();
62    Serial.println("Server started");
63  }
64
65  void loop() {
66    server.handleClient();
67
68    static unsigned long lastRecordingTime = 0;
69    static bool recordingInProgress = false;
70    unsigned long currentMillis = millis();
71
72    if (!recordingInProgress && (currentMillis - lastRecordingTime >= RECORD_TIME * 1000 + 60000)) { // 2 minutes after RECORD_TIME
73      lastRecordingTime = currentMillis;
74      recordAndUpload();
```

در این بخش از کد، سرور وب برای دریافت و ارائه فایل ضبط شده تنظیم می‌شود. همچنین، تابع loop() برای مدیریت درخواست‌های سرور و شروع ضبط جدید در زمان مناسب استفاده می‌شود. در خطوط 36 تا 40، پس از اتصال به شبکه Wi-Fi، آدرس IP محلی ESP32 بر روی سریال مانیتور چاپ می‌شود. این آدرس IP برای دسترسی به سرور وب مورد نیاز است.

در خطوط 42 تا 46، یک مسیر HTTP GET برای صفحه اصلی سرور وب تعریف شده است. وقتی کاربر به آدرس IP دستگاه متصل می‌شود، این مسیر یک صفحه HTML ساده که شامل یک لینک برای دانلود فایل ضبط شده است، به کاربر ارائه می‌دهد.

در خطوط 48 تا 59، مسیر HTTP GET دیگری برای دانلود فایل ضبط شده تعریف شده است. وقتی کاربر روی لینک دانلود کلیک می‌کند، درخواست به این مسیر ارسال می‌شود. در ابتدا پیغام "Download request received" بر روی سریال مانیتور چاپ می‌شود و سپس فایل WAV با استفاده از SPIFFS.open("/recording.wav", "r") باز می‌شود. اگر فایل وجود نداشته باشد، پیغام "File not found" چاپ شده و پاسخ 404 به کاربر ارسال می‌شود. در غیر این صورت، پیغام "File found, streaming..." چاپ شده و فایل با استفاده از server.streamFile(file, "audio/wav"); به کاربر ارسال می‌شود و سپس فایل با دستور file.close(); ` بسته می‌شود.

در خطوط 61 و 62، سرور وب با دستور server.begin(); شروع به کار می‌کند و پیغام "Server started" بر روی سریال مانیتور چاپ می‌شود. این کار باعث می‌شود سرور آماده دریافت و پاسخ به درخواست‌ها شود.

در خطوط 65 تا 74، تابع loop() قرار دارد که به صورت مداوم اجرا می‌شود. در این تابع، ابتدا با استفاده از server.handleClient(); درخواست‌های سرور مدیریت می‌شوند. سپس، متغیرهای lastRecordingTime و recordingInProgress به ترتیب برای ذخیره زمان آخرین ضبط و وضعیت ضبط فعلی تعریف شده‌اند. متغیر currentMillis با استفاده از millis() مقدار فعلی زمان اجرا را دریافت می‌کند. اگر ضبط در حال انجام نباشد و مدت زمان کافی از آخرین ضبط گذشته باشد، ضبط جدید شروع می‌شود. این کار با به‌روزرسانی lastRecordingTime و فراخوانی recordAndUpload() انجام می‌شود.

Microphone.ino

```
74      recordAndUpload();
75      recordingInProgress = true;
76    }
77
78    if (recordingInProgress && (currentMillis - lastRecordingTime >= RECORD_TIME * 1000 + 2000)) { // RECORD_TIME + extra time
79      recordingInProgress = false;
80    }
81  }
82
83  void recordAndUpload() {
84    xTaskCreate(i2s_adc, "i2s_adc", 1024 * 2, NULL, 1, NULL);
85
86    // Wait for the recording to finish
87    // delay(RECORD_TIME * 1000 + 2000); // RECORD_TIME in seconds + extra time to ensure recording finishes
88
89
90    // delay(120000); // Add a 10-second delay (adjust this value as needed)
91
92    // Upload the file to the server
93    // uploadFileToServer();
94
95    // Restart the recording process
96    SPIFFS.remove(filename);
97    file = SPIFFS.open(filename, FILE_WRITE);
98    byte header[headerSize];
99    wavHeader(header, FLASH_RECORD_SIZE);
100   file.write(header, headerSize);
101 }
102
103 void SPIFFSInit(){
104   if(!SPIFFS.begin(true)){
105     Serial.println("SPIFFS initialisation failed!");
106     while(1) yield();
107   }
108
109   SPIFFS.remove(filename);
110   file = SPIFFS.open(filename, FILE_WRITE);
111   if(!file){
```

در این بخش از کد، فرآیند ضبط صدا و آپلود آن به سرور وب توضیح داده شده است. همچنین، تنظیمات مربوط به سیستم فایل SPIFFS نیز آمده است. در خطوط 74 و 75، تابع recordAndUpload() فراخوانی می‌شود و متغیر recordingInProgress به true تنظیم می‌شود تا نشان دهد که ضبط در حال انجام است. سپس در خطوط 77 تا 79، اگر ضبط در حال انجام باشد و مدت زمان کافی از شروع ضبط گذشته باشد (بیشتر از RECORD_TIME و زمان اضافی تعیین شده)، متغیر recordingInProgress به false تغییر می‌کند تا نشان دهد که ضبط تمام شده است.

تابع recordAndUpload() که در خطوط 82 تا 101 قرار دارد، مسئولیت ایجاد یک تسک برای ضبط صدا با استفاده از xTaskCreate را دارد. این تسک برای ضبط داده‌ها از میکروفون دیجیتال INMP441 استفاده می‌شود. پس از ایجاد تسک، یک تاخیر برای اطمینان از پایان یافتن ضبط قرار داده شده است. سپس، فایل ضبط شده حذف می‌شود و یک فایل جدید برای ضبط بعدی ایجاد می‌شود. در ادامه، هدر فایل WAV با استفاده از wavHeader نوشته شده و فایل برای ضبط آماده می‌شود.

در خطوط 103 تا 115، تابع SPIFFSInit تعریف شده است که برای مقداردهی اولیه سیستم فایل SPIFFS استفاده می‌شود. این تابع ابتدا تلاش می‌کند تا SPIFFS را راه‌اندازی کند و در صورت شکست، پیغام خطا چاپ می‌شود و برنامه در یک حلقه بی‌نهایت متوقف می‌شود. سپس فایل ضبط شده قبلی حذف می‌شود و یک فایل جدید برای ضبط ایجاد می‌شود.

این بخش از کد به شما کمک می‌کند تا صدا را با استفاده از میکروفون دیجیتال INMP441 ضبط کرده و آن را به یک فایل WAV ذخیره کنید. همچنین، امکان آپلود فایل ضبط شده به سرور وب فراهم می‌شود. تنظیمات

SPIFFS نیز برای مدیریت فایل‌های ضبط شده استفاده می‌شود. این کد به طور کلی فرآیند ضبط، ذخیره و آپلود صدا را مدیریت می‌کند و از طریق سرور وب به فایل‌های ضبط شده دسترسی می‌دهد.

```
125  void i2sInit(){
126    i2s_config_t i2s_config = {
127      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
128      .sample_rate = I2S_SAMPLE_RATE,
129      .bits_per_sample = i2s_bits_per_sample_t(I2S_SAMPLE_BITS),
130      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
131      .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S | I2S_COMM_FORMAT_I2S_MSB),
132      .intr_alloc_flags = 0,
133      .dma_buf_count = 64,
134      .dma_buf_len = 1024,
135      .use_apll = true
136    };
137
138    i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
139
140    const i2s_pin_config_t pin_config = {
141      .bck_io_num = I2S_SCK,
142      .ws_io_num = I2S_WS,
143      .data_out_num = -1,
144      .data_in_num = I2S_SD
145    };
146
147    i2s_set_pin(I2S_PORT, &pin_config);
148  }
149
150
151  void i2s_adc_data_scale(uint8_t * d_buff, uint8_t* s_buff, uint32_t len)
152  {
153      uint32_t j = 0;
154      uint32_t dac_value = 0;
155      for (int i = 0; i < len; i += 2) {
156          dac_value = ((((uint16_t) (s_buff[i + 1] & 0xf) << 8) | ((s_buff[i + 0]))));
157          d_buff[j++] = 0;
158          d_buff[j++] = dac_value * 256 / 2048;
159      }
160      Serial.printf("Scaled %u bytes of data\n", len);
161  }
162
```

در این بخش از کد، تابع i2sInit() برای تنظیمات اولیه پروتکل I2S و تابع i2s_adc_data_scale() برای مقیاس‌بندی داده‌های ADC تعریف شده‌اند. این تنظیمات برای ارتباط با میکروفون دیجیتال INMP441 استفاده می‌شود.

تابع i2sInit() در خطوط 126 تا 148 قرار دارد. در ابتدا، ساختار i2s_config_t تعریف و مقداردهی می‌شود تا تنظیمات اولیه I2S مشخص شود. این تنظیمات شامل حالت کاری (Master و RX)، نرخ نمونه‌برداری، تعداد بیت‌های نمونه، فرمت کانال (تنها کانال چپ)، فرمت ارتباطی (I2S استاندارد و MSB)، تعداد و اندازه بافر DMA و استفاده از APLL است. این تنظیمات با دستور i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL) به درایور I2S اعمال می‌شوند.

سپس، ساختار i2s_pin_config_t تعریف و مقداردهی می‌شود تا پین‌های I2S مشخص شوند. این پین‌ها شامل پین‌های ساعت سریال (SCK)، انتخاب کلمه (WS) و داده ورودی (SD) هستند. با استفاده از دستور i2s_set_pin(I2S_PORT, &pin_config) این تنظیمات به پین‌های I2S اعمال می‌شوند.

تابع i2s_adc_data_scale() در خطوط 150 تا 162 تعریف شده است. این تابع برای مقیاس‌بندی داده‌های ADC استفاده می‌شود. ابتدا دو متغیر t و dac_value از نوع uint32_t تعریف می‌شوند. در داخل یک حلقه for که بر روی طول داده‌ها تکرار می‌شود، داده‌های ADC خوانده شده و مقیاس‌بندی می‌شوند. مقدار dac_value با استفاده از بیت‌شیفت و عملیات بیتی محاسبه می‌شود و سپس در آرایه d_buff ذخیره می‌شود. در نهایت، پیغام "Scaled x bytes of data" با استفاده از Serial.printf چاپ می‌شود.

```
163  void i2s_adc(void *arg)
164  {
165
166      int i2s_read_len = I2S_READ_LEN;
167      int flash_wr_size = 0;
168      size_t bytes_read;
169
170      char* i2s_read_buff = (char*) calloc(i2s_read_len, sizeof(char));
171      uint8_t* flash_write_buff = (uint8_t*) calloc(i2s_read_len, sizeof(char));
172
173      if (i2s_read_buff == NULL || flash_write_buff == NULL) {
174          Serial.println("Memory allocation failed!");
175          if (i2s_read_buff) free(i2s_read_buff);
176          if (flash_write_buff) free(flash_write_buff);
177          vTaskDelete(NULL);
178          return;
179      }
180
181      i2s_read(I2S_PORT, (void*) i2s_read_buff, i2s_read_len, &bytes_read, portMAX_DELAY);
182      i2s_read(I2S_PORT, (void*) i2s_read_buff, i2s_read_len, &bytes_read, portMAX_DELAY);
183
184      Serial.println(" *** Recording Start *** ");
185      while (flash_wr_size < FLASH_RECORD_SIZE) {
186          //read data from I2S bus, in this case, from ADC.
187          i2s_read(I2S_PORT, (void*) i2s_read_buff, i2s_read_len, &bytes_read, portMAX_DELAY);
188          Serial.printf("Bytes read: %d\n", bytes_read);
189          //example_disp_buf((uint8_t*) i2s_read_buff, 64);
190          //save original data from I2S(ADC) into flash.
191          i2s_adc_data_scale(flash_write_buff, (uint8_t*)i2s_read_buff, i2s_read_len);
192          file.write((const byte*) flash_write_buff, i2s_read_len);
193          flash_wr_size += i2s_read_len;
194          Serial.printf("Sound recording %u%%\n", flash_wr_size * 100 / FLASH_RECORD_SIZE);
195          Serial.printf("Never Used Stack Size: %u\n", uxTaskGetStackHighWaterMark(NULL));
196
197          // Add a delay to allow watchdog to reset
198          vTaskDelay(10 / portTICK_PERIOD_MS); // 10 ms delay
```

در این بخش از کد، تابع i2s_adc برای خواندن داده‌های صوتی از میکروفون دیجیتال INMP441 و ذخیره آن‌ها به صورت مقیاس‌بندی شده در حافظه فلش استفاده می‌شود. تابع i2s_adc که در خطوط 164 تا 198 تعریف شده است، برای خواندن و پردازش داده‌های صوتی از پروتکل I2S استفاده می‌شود.

ابتدا، در خطوط 166 تا 169، متغیرهای i2s_read_len و flash_wr_size و bytes_read تعریف می‌شوند تا اندازه داده‌های خوانده شده و نوشته شده در حافظه فلش را مدیریت کنند. سپس، در خطوط 171 و 172، دو بافر برای خواندن داده‌های I2S و نوشتن آن‌ها در حافظه فلش تخصیص داده می‌شود. اگر تخصیص حافظه ناموفق باشد، پیغام "Memory allocation failed!" چاپ می‌شود و حافظه‌های تخصیص داده شده آزاد می‌شوند و تسک حذف می‌شود.

در خطوط 181 و 182، داده‌های اولیه از I2S خوانده می‌شوند تا بافرها برای استفاده آماده شوند. سپس، پیغام "Recording Start" چاپ می‌شود تا نشان دهد که فرآیند ضبط شروع شده است. در داخل حلقه while که در خطوط 185 تا 196 قرار دارد، داده‌های I2S به صورت مداوم خوانده می‌شوند و در بافر i2s_read_buff ذخیره می‌شوند. سپس این داده‌ها با استفاده از تابع i2s_adc_data_scale مقیاس‌بندی می‌شوند و در حافظه فلش ذخیره می‌شوند. مقدار flash_wr_size به اندازه داده‌های خوانده شده افزایش می‌یابد و اطلاعات مربوط به فرآیند ضبط بر روی سریال مانیتور چاپ می‌شود. در نهایت، یک تأخیر 10 میلی‌ثانیه‌ای برای راه‌اندازی قرار داده شده است.

Microphone.ino

```
276  void listSPIFFS(void) {
277    Serial.println(F("\r\nListing SPIFFS files:"));
278    static const char line[] PROGMEM =  "=================================================";
279
280    Serial.println(FPSTR(line));
281    Serial.println(F("  File name                              Size"));
282    Serial.println(FPSTR(line));
283
284    fs::File root = SPIFFS.open("/");
285    if (!root) {
286      Serial.println(F("Failed to open directory"));
287      return;
288    }
289    if (!root.isDirectory()) {
290      Serial.println(F("Not a directory"));
291      return;
292    }
293
294    fs::File file = root.openNextFile();
295    while (file) {
296
297      if (file.isDirectory()) {
298        Serial.print("DIR : ");
299        String fileName = file.name();
300        Serial.print(fileName);
301      } else {
302        String fileName = file.name();
303        Serial.print("  " + fileName);
304        // File path can be 31 characters maximum in SPIFFS
305        int spaces = 33 - fileName.length(); // Tabulate nicely
306        if (spaces < 1) spaces = 1;
307        while (spaces--) Serial.print(" ");
308        String fileSize = (String) file.size();
309        spaces = 10 - fileSize.length(); // Tabulate nicely
310        if (spaces < 1) spaces = 1;
311        while (spaces--) Serial.print(" ");
312        Serial.println(fileSize + " bytes");
313      }
```

در این بخش از کد، تابع listSPIFFS() برای نمایش فایل‌های موجود در سیستم فایل SPIFFS تعریف شده است. این تابع لیستی از فایل‌ها و دایرکتوری‌های موجود در ریشه سیستم فایل SPIFFS را بر روی سریال مانیتور چاپ می‌کند.

در خطوط 276 تا 279، پیغام‌های اولیه برای نمایش لیست فایل‌ها و یک خط افقی برای جدا کردن بخش‌ها چاپ می‌شوند. با استفاده از F() و FPSTR() از ذخیره‌سازی در حافظه فلش به جای RAM استفاده می‌شود تا حافظه بیشتری صرفه‌جویی شود.

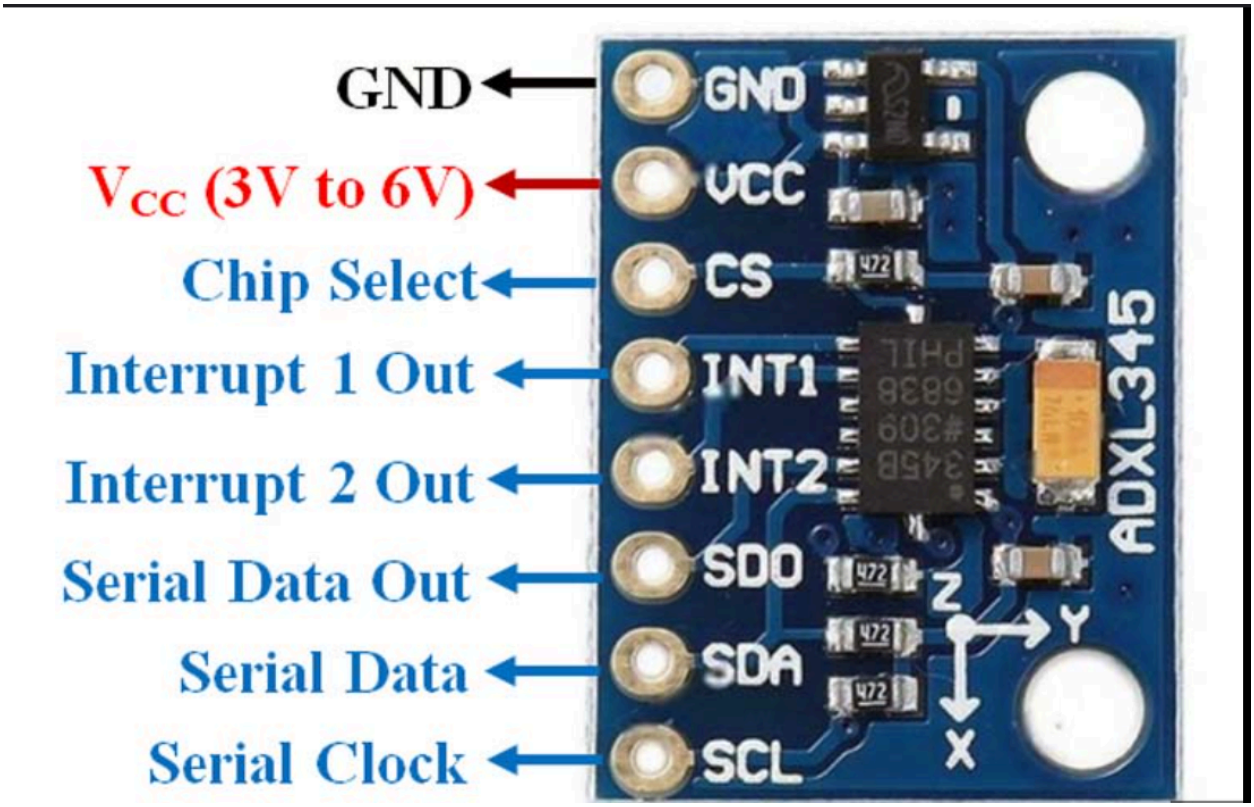
در خطوط 281 تا 283، سر ستون‌ها برای نمایش نام فایل و اندازه آن چاپ می‌شوند تا لیست فایل‌ها به صورت مرتب و قابل فهم نمایش داده شود.

در خطوط 285 تا 288، دایرکتوری ریشه سیستم فایل SPIFFS باز می‌شود. اگر باز کردن دایرکتوری ناموفق باشد، پیغام "Failed to open directory" چاپ می‌شود و تابع با استفاده از دستور return خاتمه می‌یابد. همچنین اگر دایرکتوری باز شده یک دایرکتوری نباشد، پیغام "Not a directory" چاپ می‌شود و تابع خاتمه می‌یابد.

در خطوط 293 تا 313، یک حلقه while برای پیمایش فایل‌ها و دایرکتوری‌های موجود در دایرکتوری ریشه استفاده می‌شود. در داخل حلقه، اگر فایل یک دایرکتوری باشد، پیغام "DIR" به همراه نام دایرکتوری چاپ می‌شود. اگر فایل یک دایرکتوری نباشد، نام فایل و اندازه آن چاپ می‌شود. برای چاپ مرتب و درست، تعداد فضاهای خالی محاسبه و چاپ می‌شود تا هر فایل به درستی در ستون مربوطه قرار گیرد.

ADXL and GPS

Pinout



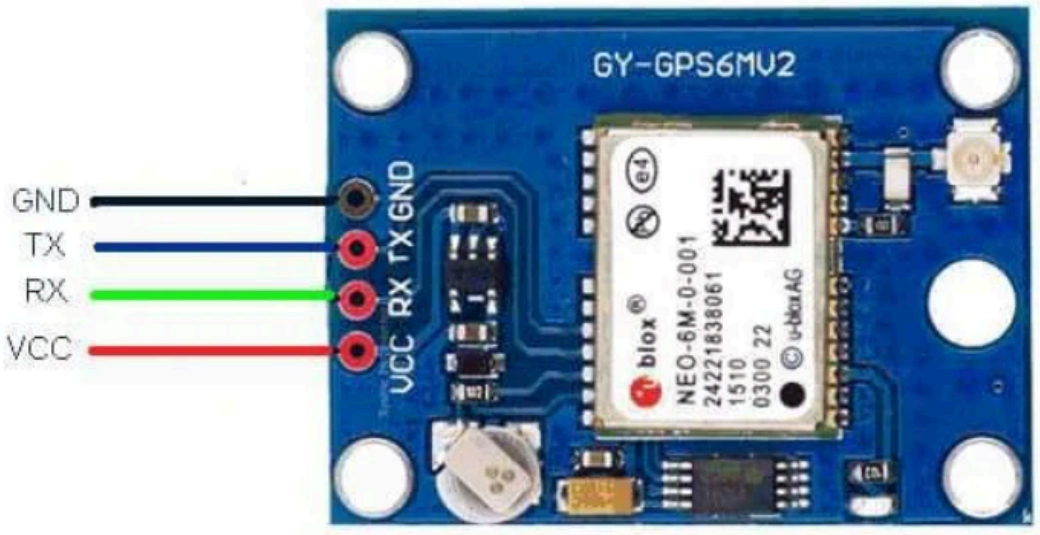
ماژول ADXL345 یک شتاب‌سنج سه‌محوره دیجیتال است که می‌تواند برای اندازه‌گیری شتاب در سه محور X، Y و Z استفاده شود. در تصویر بالا، پین‌های مختلف ماژول ADXL345 نشان داده شده‌اند که در ادامه به توضیح هر یک از آن‌ها می‌پردازم.

پین GND که با رنگ مشکی مشخص شده است، پین زمین است که برای تکمیل مدار و اطمینان از عملکرد صحیح ماژول به کار می‌رود.

پین VCC که با رنگ قرمز مشخص شده است، برای تأمین ولتاژ تغذیه ماژول استفاده می‌شود. این پین می‌تواند ولتاژی بین 3 ولت تا 6 ولت را دریافت کند. (توصیه میشود از 3.3 ولت استفاده کنید.)

پین CS که با رنگ آبی مشخص شده است، پین انتخاب چیپ (Chip Select) است که در ارتباط SPI برای انتخاب ماژول به کار می‌رود.

پین INT1 که با رنگ آبی مشخص شده است، خروجی وقفه 1 (Interrupt 1 Out) است که می‌تواند برای دریافت سیگنال‌های وقفه از ماژول استفاده شود.
پین INT2 که با رنگ آبی مشخص شده است، خروجی وقفه 2 (Interrupt 2 Out) است که مشابه پین INT1 برای دریافت سیگنال‌های وقفه از ماژول استفاده می‌شود.
پین SDO که با رنگ آبی مشخص شده است، خروجی داده‌های سریال (Serial Data Out) است که در ارتباط SPI برای ارسال داده‌ها از ماژول به میکروکنترلر استفاده می‌شود.
پین SDA که با رنگ آبی مشخص شده است، داده سریال (Serial Data) است که در ارتباط I2C و SPI برای انتقال داده‌ها استفاده می‌شود.
پین SCL که با رنگ آبی مشخص شده است، پین ساعت سریال (Serial Clock) است که در ارتباط I2C و SPI برای همگام‌سازی داده‌ها استفاده می‌شود.



NEO-6M Module Pinout

ماژول NEO-6M یک ماژول GPS بسیار معروف و پرکاربرد است که برای ردیابی موقعیت جغرافیایی استفاده می‌شود. در تصویر بالا، پین‌های مختلف ماژول NEO-6M نشان داده شده‌اند که در ادامه به توضیح هر یک از آن‌ها می‌پردازم.
پین GND که با رنگ مشکی مشخص شده است، پین زمین است که برای تکمیل مدار و اطمینان از عملکرد صحیح ماژول به کار می‌رود.

پین TX که با رنگ آبی مشخص شده است، پین ارسال داده (Transmit) است که برای ارسال داده‌های GPS به میکروکنترلر یا سایر تجهیزات استفاده می‌شود. این پین داده‌ها را از ماژول GPS به دستگاه متصل ارسال می‌کند.

پین RX که با رنگ سبز مشخص شده است، پین دریافت داده (Receive) است که برای دریافت دستورات از میکروکنترلر یا سایر تجهیزات استفاده می‌شود. این پین دستورات را از دستگاه متصل به ماژول GPS دریافت می‌کند.
پین VCC که با رنگ قرمز مشخص شده است، برای تأمین ولتاژ تغذیه ماژول استفاده می‌شود. این پین معمولاً به یک منبع تغذیه 3.3V تا 5V متصل می‌شود تا ماژول را روشن کند.

توضیحات کد

ADXL_GPS.ino

```ino
#include <Wire.h>
#include "ADXL345.h"
#include <WiFi.h>
#include <HTTPClient.h>

ADXL345 accelerometer;

const char* ssid = ""; // Replace with your WiFi SSID
const char* password = ""; // Replace with your WiFi Password

const char* serverName = ""; // Replace with your server URL


void setup(void)
{
    Serial.begin(115200);

    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi...");
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
    Serial.println("Connected to WiFi");
    delay(1000);

    Serial.println("Initialize ADXL345");

    if (!accelerometer.begin()) {
        Serial.println("Could not find a valid ADXL345 sensor, check wiring!");
        while (1);
    }

    accelerometer.setFreeFallThreshold(0.3); // Lower threshold for higher sensitivity
    accelerometer.setFreeFallDuration(0.01);   // Longer duration for more reliable detection

```

این کد برای تنظیم و ارتباط با سنسور ADXL345 و اتصال به شبکه Wi-Fi نوشته شده است. در ادامه به توضیح هر بخش از کد بر اساس شماره خطوط می‌پردازم.

در خطوط 1 تا 4، کتابخانه‌های مورد نیاز وارد شده‌اند. کتابخانه Wire.h برای ارتباط I2C، کتابخانه ADXL345.h برای ارتباط با سنسور ADXL345، کتابخانه WiFi.h برای اتصال به شبکه Wi-Fi و کتابخانه HTTPClient.h برای ارسال درخواست‌های HTTP استفاده می‌شوند.

در خط 6، یک شیء از کلاس ADXL345 به نام accelerometer ایجاد می‌شود که برای کنترل سنسور ADXL345 به کار می‌رود.

در خطوط 8 تا 11، مقادیر SSID و رمز عبور Wi-Fi و همچنین آدرس سرور تعریف شده‌اند که باید با مقادیر مربوط به شبکه Wi-Fi و سرور شما جایگزین شوند.

در خطوط 13 تا 37، تابع setup قرار دارد که شامل تنظیمات اولیه برای ارتباط سریال، اتصال به شبکه Wi-Fi و مقداردهی اولیه سنسور ADXL345 است.

در خط 15، ارتباط سریال با سرعت 115200 بیت بر ثانیه شروع می‌شود.

در خطوط 17 و 18، ارتباط Wi-Fi با استفاده از SSID و رمز عبور تعریف شده شروع می‌شود و پیام "Connecting to WiFi..." بر روی سریال مانیتور چاپ می‌شود.

در خطوط 19 تا 23، یک حلقه while تعریف شده است که تا زمانی که وضعیت اتصال Wi-Fi برابر با WL_CONNECTED نشود، هر ثانیه یک نقطه روی سریال مانیتور چاپ می‌کند و تاخیر 1000 میلی‌ثانیه‌ای دارد.

در خط 24، پس از اتصال موفقیت‌آمیز به Wi-Fi، پیام "Connected to WiFi" بر روی سریال مانیتور چاپ می‌شود و یک تاخیر 1000 میلی‌ثانیه‌ای اعمال می‌شود.

در خط 27، پیام "Initialize ADXL345" بر روی سریال مانیتور چاپ می‌شود تا فرآیند مقداردهی اولیه سنسور ADXL345 شروع شود.

در خطوط 28 تا 31، بررسی می‌شود که آیا سنسور ADXL345 به درستی مقداردهی اولیه شده است یا خیر. اگر مقداردهی اولیه ناموفق باشد، پیام "Could not find a valid ADXL345 sensor, check wiring!" چاپ می‌شود و برنامه در یک حلقه بی‌نهایت قرار می‌گیرد.

در خطوط 33 و 34، مقادیر آستانه و مدت زمان تشخیص سقوط آزاد برای سنسور ADXL345 تنظیم می‌شوند تا حساسیت و دقت تشخیص سقوط بهینه شود.

ADXL_GPS.ino

```
    // Select INT 1 for getting activities
    accelerometer.useInterrupt(ADXL345_INT1);
    checkSetup();
}

void checkSetup() {
    Serial.print("Free Fall Threshold = "); Serial.println(accelerometer.getFreeFallThreshold());
    Serial.print("Free Fall Duration = "); Serial.println(accelerometer.getFreeFallDuration());
}

void loop(void) {
    delay(50);
    Vector norm = accelerometer.readNormalize();
    Serial.print("Xnorm = "); Serial.print(norm.XAxis);
    Serial.print(" Ynorm = "); Serial.print(norm.YAxis);
    Serial.print(" Znorm = "); Serial.println(norm.ZAxis);
    Activites activ = accelerometer.readActivites();
    Serial.print("Free fall status: "); Serial.println(activ.isFreeFall);
    Serial.print("Overrun: "); Serial.println(activ.isOverrun);
    Serial.print("Watermark: "); Serial.println(activ.isWatermark);
    Serial.print("Inactivity: "); Serial.println(activ.isInactivity);
    Serial.print("Activity: "); Serial.println(activ.isActivity);
    Serial.print("Double Tap: "); Serial.println(activ.isDoubleTap);
    Serial.print("Tap: "); Serial.println(activ.isTap);
    Serial.print("Data Ready: "); Serial.println(activ.isDataReady);
    Serial.print("Activity on X: "); Serial.println(activ.isActivityOnX);
    Serial.print("Activity on Y: "); Serial.println(activ.isActivityOnY);
    Serial.print("Activity on Z: "); Serial.println(activ.isActivityOnZ);
    Serial.print("Tap on X: "); Serial.println(activ.isTapOnX);
    Serial.print("Tap on Y: "); Serial.println(activ.isTapOnY);
    Serial.print("Tap on Z: "); Serial.println(activ.isTapOnZ);
    if (activ.isFreeFall) {
        Serial.println("Free Fall Detected!");
        sendFallDetected();
        while (1) {}
    }
}
```

در خط 38، وقفه INT1 برای دریافت فعالیت‌ها از سنسور ADXL345 تنظیم می‌شود که وقفه شماره 1 را برای دریافت اطلاعات فعالیت‌ها از سنسور فعال می‌کند. سپس در خط 39، تابع checkSetup() فراخوانی می‌شود تا تنظیمات اولیه سنسور بررسی شود و اطلاعات مربوط به آستانه سقوط آزاد و مدت زمان آن چاپ شود.

تابع checkSetup() که در خطوط 41 تا 45 قرار دارد، وظیفه چاپ مقادیر آستانه و مدت زمان تشخیص سقوط آزاد را بر عهده دارد. در این تابع، ابتدا مقدار آستانه سقوط آزاد با استفاده از accelerometer.getFreeFallThreshold() خوانده شده و سپس با استفاده از Serial.print و Serial.println این مقدار بر روی سریال مانیتور چاپ می‌شود. به طور مشابه، مقدار مدت زمان سقوط آزاد با استفاده از accelerometer.getFreeFallDuration() خوانده و چاپ می‌شود. این اطلاعات برای اطمینان از تنظیمات صحیح سنسور مهم هستند.

تابع loop() که در خطوط 47 تا 73 قرار دارد، حلقه اصلی برنامه است که به صورت مداوم اجرا می‌شود و داده‌های سنسور ADXL345 را خوانده و پردازش می‌کند. در خط 48، یک تأخیر 50 میلی‌ثانیه‌ای قرار داده شده است تا سرعت نمونه‌برداری تنظیم شود. سپس در خطوط 49 تا 52، داده‌های نرمال شده از سنسور خوانده می‌شوند و مقادیر شتاب در محورهای X، Y و Z بر روی سریال مانیتور چاپ می‌شوند. این کار با استفاده از

accelerometer.readNormalize() انجام می‌شود که داده‌های شتاب را نرمال کرده و در قالب یک ساختار Vector باز می‌گرداند.

در خطوط 53 تا 70، وضعیت‌های مختلف فعالیت‌ها از سنسور خوانده شده و بر روی سریال مانیتور چاپ می‌شوند. این وضعیت‌ها شامل سقوط آزاد، اضافه‌بار، نشانگر سطح آب، عدم فعالیت، فعالیت، ضربه دوگانه و تک ضربه هستند. برای این کار، ابتدا با استفاده از accelerometer.readActivites() داده‌های فعالیت‌ها خوانده می‌شوند و سپس هر کدام از این وضعیت‌ها با استفاده از دستورات Serial.print و Serial.println چاپ می‌شوند. به عنوان مثال، وضعیت سقوط آزاد با استفاده از activ.isFreeFall و وضعیت عدم فعالیت با استفاده از activ.isInactivity خوانده و چاپ می‌شوند.

در خطوط 71 تا 73، اگر سقوط آزاد تشخیص داده شود، پیام "Free Fall Detected!" بر روی سریال مانیتور چاپ شده و تابع sendFallDetected() فراخوانی می‌شود. سپس برنامه در یک حلقه بی‌نهایت قرار می‌گیرد تا از ادامه اجرای برنامه جلوگیری شود. این کار برای اطلاع رسانی در مورد تشخیص سقوط آزاد و جلوگیری از هرگونه خطای احتمالی پس از تشخیص سقوط استفاده می‌شود.

از توضیح تابع sendFallDetected خودداری میکنیم زیرا این تابع متناسب با فرمت درخواست‌های سرور شما باید نوشته شود که برای هر سروری متفاوت میتواند باشد.

SPO2

O2HR.ino

```ino
#include <Wire.h>
#include "MAX30105.h"
#include "spo2_algorithm.h"

MAX30105 particleSensor;

// تعداد نمونه‌های مورد نیاز برای محاسبه‌ی SpO2 و ضربان قلب
const int bufferSize = 100;
uint32_t irBuffer[bufferSize]; // آرایه‌ای برای ذخیره مقادیر IR
uint32_t redBuffer[bufferSize]; // آرایه‌ای برای ذخیره مقادیر قرمز

// تعریف متغیرهای مورد نیاز
const int rateSize = 10;
float rates[rateSize]; // آرایه‌ای برای ذخیره نرخ‌های ضربان قلب
int rateSpot = 0; // موقعیت فعلی در آرایه
float beatSum = 0;
float beatAvg = 0;

const float spo2Scale = 0.98; // ضریب مقیاس‌بندی برای کالیبراسیون SpO2
const float spo2Offset = 0.0; // مقدار افست برای کالیبراسیون SpO2
const float bpmScale = 0.5; // ضریب مقیاس‌بندی برای کالیبراسیون ضربان قلب
const float bpmOffset = 30; // مقدار افست برای کالیبراسیون ضربان قلب

float spo2Buffer[rateSize]; // برای میانگین‌گیری متحرک SpO2
int spo2BufferIndex = 0;
float spo2Sum = 0;


void setup() {
  Serial.begin(115200);
  //Wire.begin(19, 18); // Use GPIO19 as SDA and GPIO18 as SCL
```

```
  //Wire.begin(19, 18); // Use GPIO19 as SDA and GPIO18 as SCL

  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30105 was not found. Please check wiring/power.");
    while (1);
  }

  particleSensor.setup(); // تنظیمات اولیه سنسور
  particleSensor.setPulseAmplitudeRed(0x0A); // روشن کردن LED قرمز
  particleSensor.setPulseAmplitudeIR(0x0A); // روشن کردن LED مادون قرمز

  Serial.println("Place your finger on the sensor with steady pressure.");

  // مقداردهی اولیه بافر SpO2 و ضربان قلب
  for (int i = 0; i < rateSize; i++) {
    spo2Buffer[i] = 0;
    rates[i] = 0;
  }
}

void loop() {
  for (int i = 0; i < bufferSize; i++) {
    while (particleSensor.available() == false) {
      particleSensor.check(); // بررسی داده‌های جدید از سنسور
    }
    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); // آماده شدن برای نمونه‌گیری بعدی
  }

  // متغیرهای مورد نیاز برای محاسبه‌ی SpO2 و BPM
  int32_t spo2;
  int8_t validSPO2;
  int32_t heartRate;
  int8_t validHeartRate;
```

```
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferSize, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

  if (validHeartRate && validSPO2) {
    // کالیبراسیون و فیلتر کردن داده‌های ضربان قلب
    float calibratedHeartRate = heartRate * bpmScale + bpmOffset;
    if (calibratedHeartRate > 30 && calibratedHeartRate < 180) {
      rates[rateSpot] = calibratedHeartRate;
      rateSpot = (rateSpot + 1) % rateSize;

      // محاسبه میانگین ضربان قلب
      beatSum = 0;
      int validBeatCount = 0;
      for (int i = 0; i < rateSize; i++) {
        if (rates[i] > 30 && rates[i] < 180) {
          beatSum += rates[i];
          validBeatCount++;
        }
      }
      beatAvg = validBeatCount > 0 ? beatSum / validBeatCount : 0;
    }

    // کالیبراسیون و فیلتر کردن داده‌های SpO2
    float calibratedSpO2 = spo2 * spo2Scale + spo2Offset;
    if (calibratedSpO2 > 70 && calibratedSpO2 <= 100) {
      spo2Buffer[spo2BufferIndex] = calibratedSpO2;
      spo2BufferIndex = (spo2BufferIndex + 1) % rateSize;

      // محاسبه میانگین SpO2
      spo2Sum = 0;
      int validSpo2Count = 0;
      for (int i = 0; i < rateSize; i++) {
        if (spo2Buffer[i] > 70 && spo2Buffer[i] <= 100) {
          spo2Sum += spo2Buffer[i];
          validSpo2Count++;
        }
```

```
      }
    }
    float spo2Avg = validSpo2Count > 0 ? spo2Sum / validSpo2Count : 0;

    // تبدیل SpO2 به رشته با دو رقم اعشار
    char spo2String[8];
    dtostrf(spo2Avg, 6, 2, spo2String);

    // نمایش نتایج
    Serial.print("Heart Rate (BPM): ");
    Serial.print(calibratedHeartRate);
    Serial.print(" | Avg BPM: ");
    Serial.print(beatAvg);
    Serial.print(" | SpO2: ");
    Serial.print(spo2String);
    Serial.println("%");
  }
 } else {
  Serial.println("Invalid reading");
 }

 delay(1000); // تأخیر یک ثانیه‌ای برای مشاهده تغییرات
}
```

ماژول MAX30102 یک سنسور نوری پیشرفته است که برای تشخیص ضربان قلب و اندازه‌گیری سطح اکسیژن خون (SpO2) طراحی شده است. این سنسور دارای دو LED (قرمز و مادون قرمز) و یک فتودیود است که با اندازه‌گیری تغییرات نور منعکس شده از خون، می‌تواند اطلاعات حیاتی مربوط به سلامت کاربر را فراهم کند.

پین‌های استفاده شده:

- VIN (VCC): این پین باید به ولتاژ تغذیه (معمولاً 3.3 ولت) متصل شود.
- GND: این پین به زمین (GND) متصل می‌شود.
- SCL: خط ساعت I2C است که برای ارتباط با ESP32 استفاده می‌شود. در این پروژه، این پین به GPIO 22 در ESP32 متصل می‌شود.
- SDA: خط داده I2C است که برای ارسال و دریافت داده‌ها استفاده می‌شود. در این پروژه، این پین به GPIO 21 در ESP32 متصل می‌شود.

نحوه اتصال:

1. VIN سنسور به پین 3V3 در ESP32 متصل می‌شود.
2. GND سنسور به پین GND در ESP32 متصل می‌شود.
3. SDA سنسور به GPIO 21 در ESP32 متصل می‌شود.
4. SCL سنسور به GPIO 22 در ESP32 متصل می‌شود.

این اتصالات به سنسور اجازه می‌دهد که با میکروکنترلر ESP32 از طریق پروتکل I2C ارتباط برقرار کند و داده‌های مربوط به ضربان قلب و سطح اکسیژن خون را ارسال و دریافت کند. این تنظیمات اساسی برای شروع کار با سنسور MAX30102 و ارتباط با ESP32 ضروری است

در این پروژه، هدف اصلی ما اتصال سنسور MAX30102 به ماژول ESP32 و استفاده از آن برای اندازه‌گیری ضربان قلب و سطح اکسیژن خون (SpO2) است. برای این منظور، اقدامات زیر را انجام دادیم:

1. راه‌اندازی کتابخانه‌های مورد نیاز:
   - کتابخانه‌های MAX30105 و spo2_algorithm را برای ارتباط و پردازش داده‌های سنسور نصب کردیم.
   - این کتابخانه‌ها شامل توابعی برای تنظیم سنسور، خواندن داده‌های خام و پردازش آنها برای محاسبه ضربان قلب و سطح اکسیژن خون هستند.
2. نوشتن و بارگذاری کد:
   - کدی را نوشتیم که سنسور را راه‌اندازی می‌کند و داده‌های مورد نیاز را از سنسور می‌خواند.
   - داده‌های خام (مقادیر نور قرمز و مادون قرمز) را در آرایه‌هایی ذخیره می‌کند و سپس با استفاده از توابع موجود در کتابخانه‌ها، این داده‌ها را پردازش کرده و ضربان قلب و سطح اکسیژن خون را محاسبه می‌کند.
   - نتایج محاسبه شده را از طریق سریال مانیتور به نمایش می‌گذارد.
3. کالیبراسیون و فیلتر کردن داده‌ها:
   - داده‌های دریافت شده از سنسور را کالیبره کرده و فیلترهای ساده‌ای برای حذف نویز و بهبود دقت اندازه‌گیری اعمال کردیم.
   - از روش‌های میانگین‌گیری برای محاسبه مقادیر دقیق‌تر ضربان قلب و SpO2 استفاده کردیم.

توضیح کد و لایبری‌های استفاده شده

در این بخش، به توضیح جزئیات کد و کتابخانه‌های استفاده شده برای پروژه می‌پردازیم:

کتابخانه‌ها:

1. Wire.h:

- این کتابخانه استاندارد برای ارتباط I2C در آردوینو استفاده می‌شود. با استفاده از این کتابخانه، ESP32 می‌تواند با سنسور MAX30102 از طریق پروتکل I2C ارتباط برقرار کند.
- توابع اصلی این کتابخانه شامل Wire.begin(), Wire.requestFrom(), Wire.write(), Wire.beginTransmission() و Wire.endTransmission() هستند که به ترتیب برای شروع ارتباط، درخواست داده، آغاز ارسال داده، نوشتن داده‌ها و پایان ارتباط استفاده می‌شوند.

2. MAX30105.h:
   - این کتابخانه توسط SparkFun ارائه شده و برای تعامل با سنسور MAX30102 استفاده می‌شود. توابع این کتابخانه شامل راه‌اندازی سنسور، خواندن داده‌های خام از سنسور، و تنظیم پارامترهای سنسور مانند شدت نور LED است.
   - توابع کلیدی شامل sensor.begin(), sensor.setup(), sensor.setPulseAmplitudeRed(), sensor.setPulseAmplitudeIR(), sensor.getRed(), sensor.getIR() و sensor.nextSample() هستند که به ترتیب برای شروع سنسور، تنظیمات سنسور، تنظیم شدت نور LED قرمز و مادون قرمز، خواندن داده‌های قرمز و مادون قرمز، و آماده‌سازی نمونه بعدی استفاده می‌شوند.
3. spo2_algorithm.h:
   - این کتابخانه شامل الگوریتم‌های محاسبه ضربان قلب و سطح اکسیژن خون بر اساس داده‌های خام خوانده شده از سنسور است. این الگوریتم‌ها با استفاده از تکنیک‌های پردازش سیگنال به محاسبه دقیق‌تر این پارامترها کمک می‌کنند.
   - تابع اصلی maxim_heart_rate_and_oxygen_saturation() است که داده‌های خام را دریافت کرده و با استفاده از روش‌های پردازش سیگنال، ضربان قلب و SpO2 را محاسبه می‌کند.

توضیح کد:

کد ارائه شده به سه بخش اصلی تقسیم می‌شود: تعریف‌ها و متغیرها، تنظیمات اولیه (setup)، و حلقه اصلی (loop).

1. تعریف‌ها و متغیرها:
   - bufferSize برای تعیین تعداد نمونه‌های مورد نیاز برای محاسبه‌ی SpO2 و ضربان قلب استفاده می‌شود و به طور پیش‌فرض روی 100 تنظیم شده است.
   - irBuffer و redBuffer آرایه‌هایی هستند که داده‌های خام مادون قرمز و قرمز را ذخیره می‌کنند.

- rateSize اندازه آرایه‌ای است که برای ذخیره نرخ‌های ضربان قلب استفاده می‌شود و به طور پیش‌فرض روی 20 تنظیم شده است.
- rates آرایه‌ای برای ذخیره نرخ‌های ضربان قلب است.
- rateSpot موقعیت فعلی در آرایه rates را نگه می‌دارد.
- beatSum و beatAvg برای میانگین‌گیری ضربان قلب استفاده می‌شوند.
- spo2Scale و spo2Offset ضریب مقیاس‌بندی و افست برای کالیبراسیون SpO2 هستند.
- bpmScale و bpmOffset ضریب مقیاس‌بندی و افست برای کالیبراسیون ضربان قلب هستند.
- spo2Buffer آرایه‌ای برای میانگین‌گیری متحرک SpO2 است.
- spo2BufferIndex و spo2Sum برای مدیریت و محاسبه میانگین SpO2 استفاده می‌شوند.

2. تابع setup:
   - Serial.begin(115200): ارتباط سریال را با نرخ 115200 بیت بر ثانیه آغاز می‌کند.
   - particleSensor.begin(Wire, I2C_SPEED_FAST): سنسور MAX30102 را راه‌اندازی می‌کند.
   - particleSensor.setup(): تنظیمات اولیه سنسور را انجام می‌دهد.
   - particleSensor.setPulseAmplitudeRed(0x0A) و particleSensor.setPulseAmplitudeIR(0x0A): شدت نور LED قرمز و مادون قرمز را تنظیم می‌کنند.
   - مقداردهی اولیه بافرهای SpO2 و ضربان قلب: تمام عناصر آرایه‌های spo2Buffer و rates را به صفر تنظیم می‌کند.
3. تابع loop:
   - در این تابع، داده‌های خام از سنسور خوانده می‌شوند و در بافرها ذخیره می‌گردند.
   - حلقه for برای جمع‌آوری داده‌های خام از سنسور:
     - while (particleSensor.available() == false): منتظر می‌ماند تا داده‌های جدید از سنسور در دسترس باشند.
     - redBuffer[i] = particleSensor.getRed(): داده‌های قرمز را از سنسور می‌خواند.
     - irBuffer[i] = particleSensor.getIR(): داده‌های مادون قرمز را از سنسور می‌خواند.
     - particleSensor.nextSample(): آماده‌سازی برای نمونه‌گیری بعدی.
   - محاسبه‌ی SpO2 و BPM با استفاده از تابع maxim_heart_rate_and_oxygen_saturation().
   - کالیبراسیون و فیلتر کردن داده‌های ضربان قلب:
     - اگر مقدار ضربان قلب محاسبه شده معتبر باشد (بین 30 و 180):
     - مقدار ضربان قلب کالیبره شده در آرایه rates ذخیره می‌شود و موقعیت فعلی در آرایه به روز می‌شود.

- میانگین ضربان قلب با استفاده از مقادیر معتبر در آرایه rates محاسبه می‌شود.
    - کالیبراسیون و فیلتر کردن داده‌های SpO2:
        - اگر مقدار SpO2 محاسبه شده معتبر باشد (بین 70 و 100):
            - مقدار SpO2 کالیبره شده در آرایه spo2Buffer ذخیره می‌شود و موقعیت فعلی در آرایه به روز می‌شود.
            - میانگین SpO2 با استفاده از مقادیر معتبر در آرایه spo2Buffer محاسبه می‌شود.
            - مقدار SpO2 میانگین با دو رقم اعشار نمایش داده می‌شود.
    - نمایش نتایج:
        - مقادیر ضربان قلب کالیبره شده، میانگین ضربان قلب، و SpO2 میانگین در سریال مانیتور نمایش داده می‌شوند.
    - delay(1000): تأخیر یک ثانیه‌ای برای مشاهده تغییرات.

این کد به گونه‌ای طراحی شده که داده‌های دقیق و پایدار از سنسور MAX30102 جمع‌آوری و پردازش شوند و نتایج به صورت قابل اعتماد و دقیق به کاربر ارائه شوند.

Body Temperature

ماژول MAX30205 یک حسگر دمای دقیق است که برای اندازه‌گیری دمای بدن طراحی شده است. این حسگر قادر به اندازه‌گیری دما با دقت بسیار بالا تا دقت ±0.1 درجه سانتی‌گراد در محدوده دمای بدن انسان (37 درجه سانتی‌گراد) است. این ماژول به دلیل دقت بالا و مصرف انرژی پایین، انتخاب مناسبی برای کاربردهای پزشکی و دستگاه‌های پوشیدنی است.

هدف از این پروژه اندازه‌گیری دمای بدن با استفاده از حسگر MAX30205 و میکروکنترلر ESP32 و نمایش داده‌های دما روی سریال مانیتور است. این پروژه برای کاربردهای پزشکی و دستگاه‌های پوشیدنی مانند دماسنج‌های دیجیتال و دستگاه‌های پایش سلامت مناسب است.

مراحل انجام پروژه:

1. نصب کتابخانه‌های مورد نیاز: ابتدا کتابخانه‌های Wire و ClosedCube_MAX30205 را نصب کردیم. کتابخانه Wire برای ارتباط I2C و کتابخانه ClosedCube_MAX30205 برای کار با حسگر MAX30205 استفاده می‌شود.
2. اتصال حسگر به ESP32: اتصالات الکتریکی حسگر MAX30205 به ESP32 به صورت زیر انجام شد:
    - VCC حسگر به پین 3.3V ESP32 متصل شد.
    - GND حسگر به پین GND ESP32 متصل شد.
    - SDA حسگر به پین GPIO 21 ESP32 متصل شد.
    - SCL حسگر به پین GPIO 22 ESP32 متصل شد.
3. تنظیمات اولیه در کد: در بخش setup، ارتباط سریال با سرعت 115200 بیت در ثانیه شروع شد و ارتباط I2C نیز با استفاده از تابع Wire.begin() آغاز شد. سپس، حسگر با استفاده از تابع sensor.begin(0x48) فعال شد که در آن 0x48 آدرس I2C پیش‌فرض حسگر است.
4. خواندن دما در حلقه loop: در بخش loop، یک حلقه بی‌نهایت برای خواندن دما و محاسبه میانگین دما در یک بازه زمانی 10 ثانیه‌ای ایجاد شد. این بخش از کد به شرح زیر عمل می‌کند:
    - مقدار دما از حسگر خوانده شده و 2 درجه به آن اضافه می‌شود (برای شبیه‌سازی یا کالیبراسیون).
    - در صورت معتبر بودن مقدار دما و قرار داشتن در محدوده 30 تا 45 درجه سانتی‌گراد، زمان شروع اندازه‌گیری ثبت می‌شود.
    - در طول 10 ثانیه، دماهای معتبر در محدوده 20 تا 40 درجه سانتی‌گراد جمع‌آوری و میانگین آن‌ها محاسبه می‌شود.
    - در نهایت، تعداد خواندن های معتبر و کل خواندن ها و همچنین زمان سپری شده نمایش داده می‌شود.
5. نمایش نتایج: نتایج شامل میانگین دما، تعدادخواندن های معتبر، کل خواندن ها و زمان سپری شده بر روی سریال مانیتور نمایش داده می‌شود. این اطلاعات به کاربر کمک می‌کند تا از صحت عملکرد حسگر و دقت اندازه‌گیری‌ها اطمینان حاصل کند.
6. تاخیر قبل از شروع مجدد: یک تاخیر 2 ثانیه‌ای قبل از شروع مجدد حلقه قرار داده شده است تا حسگر و میکروکنترلر برای دور بعدی اندازه‌گیری آماده شوند.

توضیح کد و لایبری‌های استفاده شده

کتابخانه‌ها:

- Wire: کتابخانه Wire یکی از کتابخانه‌های استاندارد Arduino است که برای برقراری ارتباط با دستگاه‌های I2C استفاده می‌شود. این کتابخانه شامل توابع لازم برای آغاز ارتباط I2C، ارسال و دریافت داده‌ها است.

- ClosedCube_MAX30205: این کتابخانه مخصوص حسگر MAX30205 است که توسط شرکت ClosedCube توسعه یافته است. این کتابخانه توابعی را فراهم می‌کند که به سادگی می‌توان دما را از حسگر MAX30205 خواند و از آن استفاده کرد. توابع این کتابخانه شامل آغاز به کار حسگر، خواندن دما و بررسی وضعیت حسگر است.

شرح کامل کد:

```
#include <Wire.h>

#include "ClosedCube_MAX30205.h"


ساخت یک شی از کلاس حسگر //

ClosedCube_MAX30205 sensor;
```

در این بخش، کتابخانه‌های Wire و ClosedCube_MAX30205 وارد شده و یک شیء از کلاس ClosedCube_MAX30205 با نام sensor ساخته می‌شود.

```
void setup() {

  Serial.begin(115200);

  Wire.begin(); // شروع ارتباط I2C


  sensor.begin(0x48); // آدرس پیش فرض حسگر MAX30205


  Serial.println("MAX30205 Body Temperature Sensor Test");
```

}

در تابع setup، ارتباط سریال با سرعت 115200 بیت بر ثانیه آغاز می‌شود و سپس ارتباط I2C با استفاده از Wire.begin() آغاز می‌شود. بعد از آن، حسگر MAX30205 با استفاده از تابع sensor.begin(0x48) فعال می‌شود که در آن 0x48 آدرس I2C پیش‌فرض حسگر است. یک پیام برای تأیید شروع برنامه در سریال مانیتور چاپ می‌شود.

```
void loop() {
  unsigned long startTime = 0;
  bool started = false;

  float sum = 0;
  int count = 0;
  int totalReadings = 0;

  while (true) {
    float temp = sensor.readTemperature(); // خواندن دما
    temp = temp + 2; // اضافه کردن 2 درجه به دما
```

تعریف شده‌اند تا برای totalReadings و count، sum، started، startTime در این بخش از کد، متغیرهای ذخیره زمان شروع، وضعیت شروع، مجموع دماهای خوانده شده، تعداد دماهای معتبر و تعداد کل خواندن ها استفاده شوند. سپس یک حلقه بی‌نهایت آغاز می‌شود که در آن دما از حسگر خوانده شده و 2 درجه به آن اضافه می‌شود.

```
// بررسی معتبر بودن دما و قرار داشتن در محدوده 30 تا 45 درجه برای شروع
 if (!started && !isnan(temp) && temp >= 30 && temp <= 45) {
   started = true;
   startTime = millis(); // زمان شروع پس از اولین دمای معتبر
   Serial.println("Started measurements after first valid temperature reading.");
                                                                                {


 if (started) {
   // بررسی معتبر بودن دما و قرار داشتن در محدوده 20 تا 40 درجه برای اندازه‌گیری
   if (!isnan(temp) && temp >= 20 && temp <= 40) {
     sum += temp;
     count++;
   }


   totalReadings++;
   unsigned long currentTime = millis(); // زمان فعلی
   unsigned long elapsedTime = currentTime - startTime;
   if (elapsedTime > 10000) {
     break;
```

```
    }
  }
}
```

در این بخش، اگر دمای خوانده شده معتبر باشد و در محدوده 30 تا 45 درجه سانتی‌گراد قرار داشته باشد، اندازه‌گیری‌ها آغاز می‌شود و زمان شروع ثبت می‌شود. در طول 10 ثانیه، اگر دمای خوانده شده معتبر باشد و در محدوده 20 تا 40 درجه سانتی‌گراد قرار داشته باشد، به مجموع دماها اضافه شده و شمارنده خواندن های معتبر افزایش می‌یابد. تعداد کل خواندن ها نیز در هر بار تکرار افزایش می‌یابد. اگر زمان سپری شده از 10 ثانیه بیشتر شود، حلقه خاتمه می‌یابد.

```
unsigned long endTime = millis(); // زمان پایان
```

```
 if (count > 0) {
   float average = sum / count;
   Serial.print("Average Temperature (20-40 C): ");
   Serial.print(average);
   Serial.println(" C");
 } else {
   Serial.println("No valid temperature data in the range 20-40 C.");
 }
```

```
 Serial.print("Number of valid readings: ");
```

```
  Serial.println(count);


  Serial.print("Total readings attempted: ");
  Serial.println(totalReadings);


  unsigned long elapsedTime = endTime - startTime; // زمان سپری شده
  Serial.print("Time taken for ");
  Serial.print(totalReadings);
  Serial.print(" readings: ");
  Serial.print(elapsedTime);
  Serial.println(" milliseconds");


  Serial.println("\n");
  Serial.println("\n");
  Serial.println("\n");


  delay(2000); // تاخیر 2 ثانیه‌ای قبل از شروع مجدد
}
```

در این بخش، زمان پایان ثبت می‌شود. اگر تعداد خواندن های معتبر بیشتر از صفر باشد، میانگین دما محاسبه و چاپ می‌شود. در غیر این صورت، پیامی مبنی بر عدم وجود داده‌های دمای معتبر چاپ می‌شود. سپس تعداد

خواندن های معتبر و کل خواندن ها چاپ می‌شود. زمان سپری شده برای انجام خواندن ها نیز محاسبه و چاپ می‌شود. پس از این، یک تاخیر 2 ثانیه‌ای قبل از شروع مجدد حلقه قرار داده شده است.

این کد با استفاده از حسگر MAX30205 و میکروکنترلر ESP32، دمای بدن را اندازه‌گیری می‌کند و میانگین دما را در یک بازه 10 ثانیه‌ای محاسبه می‌کند. این اطلاعات بر روی سریال مانیتور نمایش داده می‌شوند تا کاربر بتواند از صحت عملکرد حسگر و دقت اندازه‌گیری‌ها اطمینان حاصل کند.

ESP32-Cam

در این پروژه، هدف ما راه‌اندازی ماژول ESP32-CAM، اتصال آن به یک شبکه Wi-Fi و ارسال تصاویر زنده به یک مرورگر وب است. برای این منظور، مراحل زیر را انجام دادیم:

1. اتصالات فیزیکی: اتصال پین‌های ضروری ماژول به مبدل USB به سریال (FTDI) و منبع تغذیه پایدار.
2. ورود به حالت برنامه‌ریزی: قرار دادن ماژول در حالت برنامه‌ریزی با اتصال GPIO 0 به GND و ریست کردن برد.
3. آپلود کد: نوشتن و آپلود کد در Arduino IDE برای اتصال به شبکه Wi-Fi و ارسال تصاویر دوربین به یک سرور HTTP.
4. مشاهده تصاویر زنده: مشاهده تصاویر زنده از دوربین ماژول ESP32-CAM در مرورگر وب با استفاده از آدرس IP محلی.

در این پروژه، ما از پین‌های زیر برای اتصال ماژول ESP32-CAM استفاده کرده‌ایم:

- GPIO 0 (IO0): برای ورود ماژول به حالت برنامه‌ریزی (Boot Mode). این پین باید به GND متصل شود تا ماژول به حالت برنامه‌ریزی برود.

- U0R و U0T: برای ارتباط سریال (RX و TX). این پین‌ها برای آپلود کد و ارتباط با کامپیوتر استفاده می‌شوند.
- GND: زمین یا منفی تغذیه. برای تکمیل مدار الکتریکی و ایجاد مرجع مشترک.
- 5V: تغذیه ماژول.5V برای تغذیه استفاده می‌شود.

Boot کردن ESP32-CAM

برای برنامه‌ریزی و آپلود کد بر روی ماژول ESP32-CAM، باید آن را به حالت Boot Mode یا حالت برنامه‌ریزی ببریم. این فرآیند شامل مراحل زیر است:

1. اتصال GPIO 0 به GND:
   - اولین قدم برای قرار دادن ماژول در حالت برنامه‌ریزی این است که پین GPIO 0 (IO0) را به GND (زمین) متصل کنید. این اتصال باعث می‌شود که ماژول هنگام ریست شدن به حالت برنامه‌ریزی برود.
2. اتصال به منبع تغذیه و مبدل USB به سریال:
   - مطمئن شوید که ماژول به یک منبع تغذیه پایدار متصل است. می‌توانید از پین 3.3V یا 5V برای تغذیه استفاده کنید. همچنین، پین‌های RX و TX ماژول را به مبدل USB به سریال (FTDI) متصل کنید.
3. ریست کردن برد:
   - برای وارد کردن ماژول به حالت برنامه‌ریزی، باید برد را ریست کنید. این کار را می‌توانید با دکمه ریست (RESET) انجام دهید یا با قطع و وصل مجدد تغذیه.
   - در حالی که GPIO 0 به GND متصل است، دکمه ریست را فشار دهید و سپس رها کنید. پس از ریست شدن، ماژول باید در حالت برنامه‌ریزی قرار گیرد.
4. آپلود کد از طریق Arduino IDE:
   - در Arduino IDE، برد AI Thinker ESP32-CAM را از منوی Tools -> Board انتخاب کنید.
   - پورت COM مربوط به مبدل USB به سریال را از منوی Tools -> Port انتخاب کنید.
   - کد خود را بنویسید و روی دکمه آپلود کلیک کنید. Arduino IDE کد را کامپایل کرده و از طریق پورت سریال به ماژول ESP32-CAM آپلود می‌کند.
   - در حین آپلود، اگر ماژول در حالت "Connecting..." گیر کرد، ممکن است نیاز باشد دکمه ریست را دوباره فشار دهید یا مراحل را دوباره انجام دهید.
5. جدا کردن GPIO 0 از GND:
   - پس از اتمام فرآیند آپلود کد، GPIO 0 را از GND جدا کنید تا ماژول به حالت عادی بازگردد.

حال ، به تفصیل به توضیح کد و کتابخانه‌های استفاده شده در پروژه خواهیم پرداخت. این کد به طور کلی برای اتصال ماژول ESP32-CAM به یک شبکه Wi-Fi و راه‌اندازی یک سرور HTTP برای ارسال تصاویر زنده دوربین به مرورگر وب طراحی شده است.

کتابخانه‌های استفاده شده

1. esp_camera.h:
   - این کتابخانه برای راه‌اندازی و استفاده از دوربین داخلی ماژول ESP32-CAM طراحی شده است. این کتابخانه شامل توابع و تعاریفی است که برای تنظیمات اولیه دوربین، گرفتن تصاویر و مدیریت فریم‌های تصویر استفاده می‌شود.
2. WiFi.h:
   - این کتابخانه برای مدیریت اتصالات Wi-Fi استفاده می‌شود. این کتابخانه شامل توابعی برای اتصال به شبکه‌های Wi-Fi، مدیریت اتصالات و دریافت اطلاعات شبکه است. با استفاده از این کتابخانه می‌توانیم ماژول ESP32-CAM را به یک شبکه Wi-Fi متصل کنیم و از آن برای ارسال داده‌ها استفاده کنیم.

توضیح کد

تعریف و تنظیمات اولیه

در ابتدا، کتابخانه‌های مورد نیاز را وارد می‌کنیم و سپس نام شبکه Wi-Fi و رمز عبور آن را تعریف می‌کنیم.

```
#include "esp_camera.h"
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "ali";
const char* password = "13801380";
```

در این بخش، دو کتابخانه اصلی وارد شده‌اند: esp_camera.h برای کنترل دوربین و WiFi.h برای مدیریت اتصالات Wi-Fi. همچنین، نام شبکه Wi-Fi و رمز عبور آن در متغیرهای ssid و password تعریف شده‌اند.

تنظیمات پین‌ها و مدل دوربین

در این بخش، مدل دوربین و پین‌های مرتبط با آن مشخص می‌شوند. برای ماژول ESP32-CAM، معمولاً از مدل AI THINKER استفاده می‌شود.

```
#define CAMERA_MODEL_AI_THINKER

#if defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM     0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
#define Y2_GPIO_NUM       5
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22
#endif
```

این بخش شامل تعاریف پین‌هایی است که برای اتصال دوربین به ماژول ESP32 استفاده می‌شوند. هر پین دارای یک شماره GPIO است که به پین‌های سخت‌افزاری ماژول ESP32 متصل می‌شود.

تابع setup()

تابع setup() برای تنظیمات اولیه ماژول و اتصال به شبکه Wi-Fi استفاده می‌شود. همچنین، در این تابع دوربین راه‌اندازی می‌شود و آدرس IP محلی به دست آمده چاپ می‌شود.

```
void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
```

```
  if(psramFound()){
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_CIF;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // Camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  // Wi-Fi connection
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Print the local IP address
  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}
```

1. تنظیم ارتباط سریال: با Serial.begin(115200) ارتباط سریال با نرخ باود 115200 آغاز می‌شود. این ارتباط برای نمایش پیام‌ها در Serial Monitor استفاده می‌شود.
2. تنظیمات دوربین: با استفاده از ساختار camera_config_t، پین‌های مرتبط با دوربین و تنظیمات مربوط به کیفیت تصویر، فرکانس ساعت و فرمت پیکسل‌ها مشخص می‌شوند. اگر حافظه PSRAM موجود باشد، تنظیمات مربوط به کیفیت تصویر و تعداد فریم‌بافرها بهینه‌تر می‌شود.
3. راه‌اندازی دوربین: با فراخوانی esp_camera_init(&config), دوربین راه‌اندازی می‌شود. اگر راه‌اندازی موفقیت‌آمیز باشد، ماژول آماده گرفتن تصاویر است.
4. اتصال به شبکه Wi-Fi: با استفاده از WiFi.begin(ssid, password)، ماژول به شبکه Wi-Fi متصل می‌شود. با استفاده از حلقه while (WiFi.status() != WL_CONNECTED), وضعیت اتصال بررسی می‌شود تا زمانی که اتصال برقرار شود.
5. چاپ آدرس IP محلی: پس از اتصال موفقیت‌آمیز به شبکه Wi-Fi، آدرس IP محلی که به ماژول اختصاص داده شده است، در Serial Monitor چاپ می‌شود. این آدرس IP برای دسترسی به سرور HTTP مورد استفاده قرار می‌گیرد.

تابع loop()

تابع loop() در این مثال خالی است. این تابع معمولاً برای اجرای کدهایی استفاده می‌شود که باید به صورت مداوم تکرار شوند. در این پروژه، نیازی به کدهای تکراری نیست، زیرا سرور HTTP به طور مستقل اجرا می‌شود.

```
void loop() {
  // put your main code here, to run repeatedly:
}
```

Frontend

1. مقدمه

1.1. معرفی پروژه

سامانه‌ی پایش سلامت یک وب اپلیکیشن است که با استفاده از فناوری‌ها، به منظور بهبود کیفیت مراقبت‌های پزشکی و فراهم کردن امکان برقراری ارتباط مداوم بین بیماران و تیم درمان طراحی شده است. این سامانه به

صورت ویژه برای دستگاه‌های سخت‌افزاری طراحی شده که داده‌های سلامت بیماران را اندازه‌گیری کرده و به اپلیکیشن ارسال می‌کنند.

1.2. هدف پروژه

هدف اصلی این پروژه، ایجاد بستری است که بیماران بتوانند به صورت پیوسته وضعیت سلامت خود را با تیم درمان به اشتراک بگذارند و از راه دور تحت نظارت باشند. این سامانه با ارائه امکاناتی نظیر مشاهده و ثبت داده‌های سلامت، ارسال و دریافت پیام، ثبت شرح حال و دریافت توصیه‌ها و نسخه‌های پزشکی، بهبود ارتباط بین بیمار و تیم درمان را تضمین می‌کند. همچنین با ارسال آلارم‌های حیاتی در مواقع ضروری، کمک می‌کند تا اقدامات فوری برای نجات جان بیماران انجام شود.

1.3. مخاطبان هدف

مخاطبان هدف این سامانه شامل دو گروه اصلی هستند:

1. بیماران: بیمارانی که نیاز به پایش مداوم سلامت دارند و می‌خواهند داده‌های سلامتی خود را به راحتی و در هر زمانی با تیم درمان به اشتراک بگذارند.
2. تیم درمان: پزشکان، پرستاران و دیگر اعضای تیم درمان که نیاز به دسترسی به داده‌های سلامت بیماران دارند تا بتوانند به صورت مستمر وضعیت آن‌ها را بررسی کرده و در مواقع ضروری به آن‌ها توصیه‌ها و نسخه‌های لازم را ارائه دهند.

1.4. مزایای پروژه

این سامانه با هدف بهبود کیفیت زندگی بیماران و افزایش کارایی تیم درمان طراحی شده است و دارای مزایای متعددی است:

- افزایش کیفیت مراقبت‌های پزشکی: با ارائه داده‌های دقیق و به روز به پزشکان، امکان تشخیص به موقع و دقیق‌تر بیماری‌ها فراهم می‌شود.
- کاهش هزینه‌های درمان: با کاهش نیاز به مراجعه حضوری به مراکز درمانی و امکان نظارت از راه دور، هزینه‌های مرتبط با درمان کاهش می‌یابد.
- افزایش راحتی بیماران: بیماران می‌توانند بدون نیاز به خروج از منزل، وضعیت سلامت خود را با پزشکان به اشتراک بگذارند و از توصیه‌ها و نسخه‌های پزشکی بهره‌مند شوند.
- پیشگیری از حوادث ناگوار: با ارسال آلارم‌های حیاتی به تیم درمان در مواقع ضروری، امکان واکنش سریع و جلوگیری از حوادث ناگوار افزایش می‌یابد.

1.5. فناوری‌های مورد استفاده

این سامانه با استفاده از فناوری‌های وب و ابزارهای توسعه پیاده‌سازی شده است:

- Next.js: یک فریمورک قدرتمند برای ساخت برنامه‌های React که قابلیت‌های SSR (Server-Side Rendering) و SSG (Static Site Generation) را فراهم می‌کند.
- React: یک کتابخانه جاوااسکریپت برای ساخت رابط کاربری که به دلیل سادگی و کارایی بالا بسیار محبوب است.
- CSS: برای طراحی و استایل‌دهی به صفحات وب.
- MUI: یک کتابخانه‌ی محبوب برای طراحی رابط کاربری در React که بر اساس اصول طراحی متریال دیزاین گوگل ساخته شده و قابلیت‌های فراوانی برای ساخت رابط‌های کاربری زیبا و واکنش‌گرا فراهم می‌کند.

1.6. چشم‌انداز آینده

در آینده، این سامانه می‌تواند با افزودن قابلیت‌های پیشرفته‌تری مانند تحلیل داده‌های سلامت با استفاده از هوش مصنوعی و یادگیری ماشین، ارائه پیش‌بینی‌های دقیق‌تر و تشخیص‌های زودهنگام بیماری‌ها، به یکی از ابزارهای حیاتی در حوزه سلامت از راه دور تبدیل شود. همچنین با گسترش قابلیت‌های ارتباطی و افزودن امکانات جدید، می‌توان کیفیت خدمات ارائه شده را بهبود بخشید و رضایت کاربران را افزایش داد.

2. نیازمندی‌ها

2.1. نیازمندی‌های عملکردی

امکانات و سطوح دسترسی بیماران:

1. **داشبورد همراه بیمار:**
   - نمایش اطلاعات کلی بیمار و داده‌های سلامت جمع‌آوری‌شده.
   - امکان دسترسی به بخش‌های مختلف مانند مشاهده داده‌های سلامت، ثبت شرح حال، مشاهده توصیه‌ها و نسخه‌ها، و ارتباط با تیم درمان.
2. **ثبت نام و وارد کردن مشخصات بیمار:**
   - فرم ثبت نام با فیلدهای نام، نام خانوادگی، کد ملی، شهر، بیماری‌ها، تاریخ تولد، قد، وزن، شغل و دیگر اطلاعات مهم.
   - امکان انتخاب نوع حساب کاربری (بیمار یا پزشک) در صفحه ثبت نام.
3. **آلارم شرایط حیاتی بیمار:**
   - ارسال آلارم به همراه بیمار در صورت بروز شرایط حیاتی خطرناک بر اساس پارامترهای تعیین‌شده.

4. **مشاهده و ثبت داده‌های سلامت:**
    - امکان مشاهده داده‌های سلامت جمع‌آوری‌شده از دستگاه‌های پایش.
    - امکان ثبت دستی داده‌هایی که به صورت خودکار قابل اندازه‌گیری نیستند مانند قند خون، وزن و غیره.
5. **ثبت و مشاهده شرح حال:**
    - امکان ثبت شرح حال بیمار به صورت متن و تصویر در بازه‌های زمانی مختلف.
    - امکان مشاهده تاریخچه کامل شرح حال بیمار.
6. **یادآورها:**
    - امکان ثبت یادآور برای همراه بیمار و تیم درمان.
    - ارسال یادآور در زمان معین شده برای اطلاعات مهم و حیاتی.
7. **مشاهده توصیه‌ها و نسخه‌های پزشک:**
    - امکان مشاهده توصیه‌ها و نسخه‌های درمانی ارسال‌شده توسط پزشکان به صورت متن و تصویر.
8. **ارتباط با تیم درمان:**
    - امکان ارسال و دریافت پیام بین همراه بیمار و تیم درمان.
    - امکان مشاوره و پیگیری وضعیت بیمار از طریق پیام‌ها.
9. **تاریخچه پزشکی:**
    - مشاهده تاریخچه کامل بیماری‌های مختلف، داروهای مصرفی، نتایج آزمایش‌ها و ...
10. **انتخاب و مدیریت بیماری‌ها:**
    - امکان انتخاب بیماری‌ها از لیست و تعیین سطوح خطرناک بودن علایم بر اساس بیماری‌های مختلف.
    - تنظیم ورودی‌های دستی به صورت دوره‌ای برای پارامترهای مشخص مانند قند خون روزانه، وزن هفتگی و غیره.

امکانات و سطوح دسترسی تیم درمان:

1. **مشاهده داده‌های سلامت بیماران:**
    - تیم درمان می‌تواند دسترسی کامل به داده‌های جمع‌آوری‌شده از دستگاه‌های پایش بیماران برای تحلیل و بررسی وضعیت آنها داشته باشد.
2. **داشبورد تیم درمان:**
    - داشبورد شامل امکانات در دسترس تیم درمان می‌شود که می‌تواند جهت دسترسی به امکانات ارائه شده از آن استفاده کند.
3. **ثبت نام و وارد کردن مشخصات پزشک:**

- فرم ثبت نام با فیلدهای نام، نام خانوادگی، کد ملی، شهر، تخصص و دیگر اطلاعات مهم.

4. **آلارم شرایط حیاتی بیمار به تیم درمان:**
   - ارسال آلارم به تیم درمان در صورت بروز شرایط حیاتی خطرناک برای بیماران بر اساس پارامترهای تعیین‌شده.
5. **ثبت و مشاهده شرح حال بیمار:**
   - امکان ثبت شرح حال بیمار به صورت متن و تصویر در بازه‌های زمانی مختلف.
   - امکان مشاهده تاریخچه کامل شرح حال بیمار.
6. **لیست بیماران:**
   - هر پزشک لیستی از بیماران تحت درمان خود را دارد که می‌تواند برای هر یک از آن‌ها خدمات جداگانه‌ای ارائه کند.
7. **ارسال توصیه‌ها و نسخه‌های پزشک:**
   - امکان ارسال توصیه‌ها و نسخه‌های درمانی به بیماران به صورت متن و تصویر.
8. **دریافت و تحلیل گزارش‌های شرح حال:**
   - بررسی گزارش‌های شرح ارسالی از بیمار و سرپرست برای پیگیری بهتر درمان توسط تیم درمان.
9. **ارتباط با بیماران:**
   - امکان ارسال و دریافت پیام بین پزشک و بیماران.
   - امکان مشاوره و پیگیری وضعیت بیمار از طریق پیام‌ها.
10. **تنظیم و مدیریت یادآورها:**
    - ایجاد و مدیریت یادآوری‌ها برای پیگیری‌های درمانی و ویزیت‌های دوره‌ای.
11. **شخصی‌سازی علایم حیاتی برای بیماران خاص:**
    - تنظیم اعداد مربوط به علایم حیاتی بیماران برای شرایط ضروری به صورت دستی.
12. **اشتراک‌گذاری دستاوردهای علمی جدید:**
    - امکان اشتراک‌گذاری آخرین دستاوردهای علمی توسط پزشکان که ممکن است برای بیماران مفید باشد.
13. **امکان درمان گروهی:**
    - تبادل اطلاعات بیماران بین پزشکان زمینه‌های مختلف برای درمان گروهی.
14. **امکان رزرو نوبت از پزشکان عضو برنامه:**
    - بیماران می‌توانند نوبت ملاقات با پزشکان را از طریق برنامه رزرو کنند.
15. **معرفی نزدیک‌ترین مراکز اورژانس به بیمار:**
    - معرفی نزدیک‌ترین مراکز اورژانس به بیمار بر اساس موقعیت جغرافیایی.

2.2. نیازمندی‌های غیر عملکردی

1. **رابط کاربری و تجربه کاربری (UI/UX):**
   - استفاده از MUI (Material-UI) برای طراحی رابط کاربری زیبا و کاربرپسند.
   - رعایت اصول طراحی واکنش‌گرا (Responsive Design) برای دسترسی آسان از طریق دستگاه‌های مختلف.

3. معماری سیستم

3.1. معماری کلی

سامانه‌ی پایش سلامت از راه دور به عنوان یک وب اپلیکیشن طراحی شده است که شامل چندین بخش و اجزا مختلف می‌باشد. این سامانه از یک معماری مبتنی بر کلاینت-سرور استفاده می‌کند که در آن قسمت کلاینت (فرانت‌اند) با استفاده از فناوری‌های وب پیاده‌سازی شده است. معماری کلی سیستم به شرح زیر است:

- **کلاینت (Frontend):**
  - پیاده‌سازی شده با استفاده از Next.js و React.
  - شامل تمامی اجزای رابط کاربری و منطق سمت کاربر.

3.2. اجزای مختلف سیستم

کلاینت (Frontend)

1. **Next.js و React:**
   - Next.js به عنوان فریمورک اصلی برای پیاده‌سازی پروژه استفاده شده است که قابلیت‌های SSR (Server-Side Rendering) و SSG (Static Site Generation) را فراهم می‌کند.
   - React برای ساخت و مدیریت کامپوننت‌های رابط کاربری استفاده شده است.
2. **MUI (Material-UI):**
   - استفاده از MUI برای طراحی رابط کاربری زیبا و کاربر پسند.
   - پیاده‌سازی اصول طراحی واکنش‌گرا (Responsive Design) برای دسترسی آسان از طریق دستگاه‌های مختلف.
3. **ساختار پوشه‌ها و فایل‌ها:**
   - **`app/`**: شامل تمامی کدهای اصلی برنامه.
     - **`favicon.ico`**: آیکون سایت.
     - **`globals.css`**: استایل‌های عمومی پروژه.
     - **`layout.js`**: کامپوننت اصلی برای ساختار کلی صفحات.
     - **`page.js`**: صفحه اصلی برنامه.

- **`page.module.css`**: استایل‌های مربوط به صفحه اصلی.
- **`theme.js`**: تنظیمات مربوط به تم اپلیکیشن.
- **`/user`**: شامل تمامی فایل‌ها و کامپوننت‌های مربوط به کاربر (پزشک و بیمار).
  - **پوشه `doctor`**: شامل صفحات و کامپوننت‌های مربوط به پزشک.
  - **پوشه `forgot-password`**: صفحه فراموشی رمز عبور.
  - **پوشه `login`**: صفحه ورود.
  - **پوشه `patient`**: شامل صفحات و کامپوننت‌های مربوط به بیمار.
  - **پوشه `signup`**: صفحه ثبت نام.

4. **کامپوننت‌های اصلی:**
   - **صفحات ورود و ثبت نام:**
     - **`login/page.js`**: صفحه ورود.
     - **`signup/page.js`**: صفحه ثبت نام.
   - **داشبورد:**
     - **داشبورد بیمار**: نمایش اطلاعات و داده‌های سلامت بیمار.
     - **داشبورد پزشک**: نمایش اطلاعات و داده‌های بیماران.
   - **صفحات مشاهده و ثبت داده‌های سلامت:**
     - **`user/patient/data`**: مشاهده داده‌های سلامت.
     - **`user/patient/data/edit`**: ویرایش داده‌های سلامت.
   - **صفحات مشاهده و ثبت شرح حال:**
     - **`user/patient/data/status`**: مشاهده وضعیت سلامت.
     - **`user/patient/data/status/history`**: مشاهده تاریخچه وضعیت سلامت.
   - **ارتباط با تیم درمان:**
     - **`user/doctor/notif/page.js`**: مشاهده نوتیفیکیشن‌ها.
     - **`user/doctor/notif/add/page.js`**: افزودن نوتیفیکیشن جدید.

3.3. گردش کار سیستم (Workflow)

1. **ثبت نام و ورود:**
   - کاربران (بیماران و پزشکان) از طریق صفحات ثبت نام و ورود وارد سامانه می‌شوند.
   - اطلاعات ثبت نام کاربران در پایگاه داده ذخیره می‌شود.
2. **دریافت و مشاهده داده‌های سلامت:**
   - داده‌های سلامت از دستگاه‌های سخت‌افزاری به صورت مستقیم به اپلیکیشن ارسال می‌شود.

- کاربران می‌توانند داده‌های سلامت خود را مشاهده و در صورت نیاز ویرایش کنند.

3. **ثبت و مشاهده شرح حال:**
   - کاربران (بیماران و پزشکان) می‌توانند شرح حال خود را ثبت کرده و تاریخچه کامل آن را مشاهده کنند.
4. **ارسال و دریافت نوتیفیکیشن‌ها:**
   - نوتیفیکیشن‌های مربوط به شرایط حیاتی و یادآوری‌ها به کاربران ارسال می‌شود.
   - کاربران می‌توانند نوتیفیکیشن‌های خود را مشاهده و مدیریت کنند.
5. **ارتباط بین بیماران و پزشکان:**
   - بیماران و پزشکان می‌توانند از طریق پیام‌ها با یکدیگر در ارتباط باشند.
   - پزشکان می‌توانند توصیه‌ها و نسخه‌های درمانی را برای بیماران ارسال کنند.
6. **مدیریت داده‌ها و گزارش‌ها:**
   - پزشکان می‌توانند داده‌های سلامت بیماران را مشاهده و تحلیل کنند.
   - گزارش‌های شرح حال بیماران بررسی و تحلیل می‌شود.

4. راهنمای نصب و اجرا

4.1. پیش‌نیازهای نصب

برای نصب و اجرای این پروژه، نیاز به نرم‌افزارها و ابزارهای زیر دارید:

1. **Node.js**:
   - نسخه پیشنهادی: >= x.14
   - لینک دانلود: Node.js
2. **npm (Node Package Manager)**:
   - نسخه پیشنهادی: >= x.6
   - به همراه Node.js نصب می‌شود.
3. **Git**:
   - برای مدیریت نسخه‌ها و کلون کردن ریپازیتوری پروژه.
   - لینک دانلود: Git

4.2. نصب وابستگی‌ها

کلون کردن ریپازیتوری پروژه: ابتدا باید ریپازیتوری پروژه را با استفاده از Git کلون کنید. برای این کار، دستور زیر را در ترمینال اجرا کنید:

```
git clone https://github.com/AnitaAlikhani/HealthMonitoringProject.git
```

نصب وابستگی‌ها: پس از کلون کردن ریپازیتوری، وارد دایرکتوری پروژه شوید و وابستگی‌ها را با استفاده از npm نصب کنید:

```
cd [PROJECT_DIRECTORY]
npm install
```

- جایگزین `[PROJECT_DIRECTORY]` با نام دایرکتوری پروژه.

4.3. اجرای پروژه

اجرای سرور توسعه: برای اجرای پروژه در حالت توسعه، دستور زیر را اجرا کنید:

```
npm run dev
```

- این دستور سرور توسعه را بر روی پورت پیش‌فرض (معمولاً 3000) اجرا می‌کند. می‌توانید با وارد کردن `http://localhost:3000` در مرورگر خود، پروژه را مشاهده کنید.

ساخت پروژه برای محیط تولید: برای ساخت پروژه جهت استفاده در محیط تولید، دستور زیر را اجرا کنید:

```
npm run build
```

- این دستور پروژه را برای استقرار در محیط تولید آماده می‌کند.

اجرای پروژه در محیط تولید: پس از ساخت پروژه، می‌توانید سرور تولید را با دستور زیر اجرا کنید:

```
npm start
```

- این دستور پروژه را در حالت تولید بر روی پورت پیش‌فرض (معمولاً 3000) اجرا می‌کند.

4.4. پیکربندی‌های مورد نیاز

1. **تنظیمات فایل `next.config.js`:**
   - این فایل شامل تنظیمات مربوط به Next.js است. می‌توانید تنظیمات مختلف مانند مسیرها، متغیرهای محیطی و ... را در این فایل تعریف کنید.
2. **تنظیمات فایل `.env`:**

- برای مدیریت متغیرهای محیطی، از فایل `.env` استفاده کنید. می‌توانید متغیرهای مهم مانند آدرس APIها، کلیدهای امنیتی و ... را در این فایل تعریف کنید.

3. **تنظیمات فایل `package.json`:**
   - این فایل شامل اطلاعات مربوط به پروژه، اسکریپت‌های npm و وابستگی‌های پروژه است. می‌توانید اسکریپت‌های جدید اضافه کنید یا وابستگی‌ها را مدیریت کنید.

4.6. ابزارهای توسعه

1. **ویراستار کد (IDE):**
   - Visual Studio Code (پیشنهادی)
   - لینک دانلود: Visual Studio Code
2. **افزونه‌های پیشنهادی برای ویراستار کد:**
   - ESLint: برای بررسی و تصحیح خطاهای کدنویسی
   - Prettier: برای قالب‌بندی کد
   - Material-UI: برای استفاده از کامپوننت‌های MUI
3. **پیکربندی محیط توسعه:**
   - تنظیمات فایل `.eslintrc.json` برای ESLint
   - تنظیمات فایل `jsconfig.json` برای پشتیبانی از ویژگی‌های جاوااسکریپت
   - تنظیمات فایل `tsconfig.json` برای TypeScript (در صورت استفاده)