



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA



Lucas Martins Primo
Raul Nicolini Rodrigues
Renato Souza Santana Filho

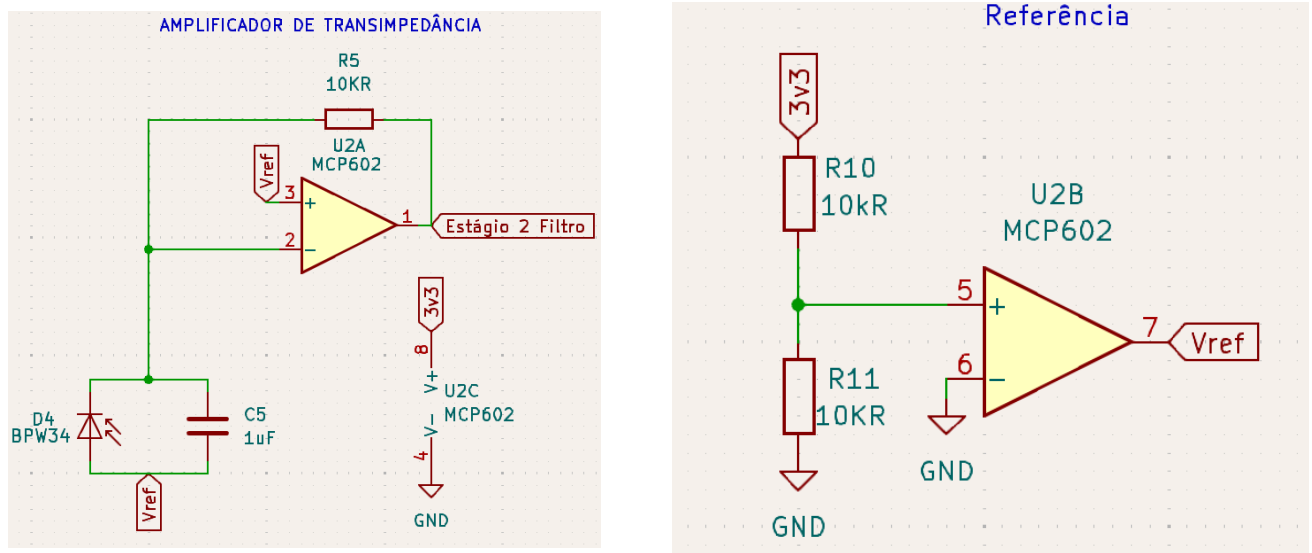
Oxímetro

UBERLÂNDIA
2024

Diagrama eletrônico proposto pelo grupo:

O circuito foi idealizado no editor KiCad com base em >>> e pode ser observado na Figura 1 e 2, o circuito de transimpedância e a divisão de tensão com um amplificador operacional para fazer a referência das demais fases. No primeiro circuito é utilizado em um oxímetro de pulso para converter a corrente elétrica gerada pelo fotodiodo em uma tensão mensurável. No caso de um oxímetro de pulso, LEDs emitem luz através da pele, e o fotodiodo detecta a luz transmitida. A corrente gerada pelo fotodiodo devido à absorção diferencial da luz pelo oxigênio no sangue é muito fraca, e o amplificador de transimpedância amplifica essa corrente, transformando-a em um sinal de tensão que pode ser processado e analisado para determinar a saturação de oxigênio no sangue.

Figuras 1 e 2: Etapa de transimpedância e de geração de Referência para o sistema

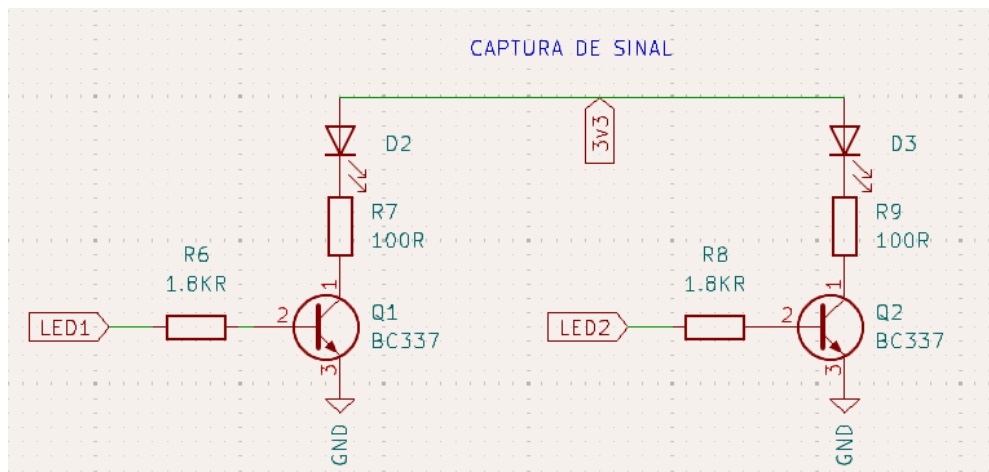


Fonte: Autoria própria

Nesse sentido, foram utilizados dois LEDs de diferentes comprimentos de onda, conforme a Figura 3, um vermelho e um infravermelho para medir a saturação de oxigênio no sangue. Isso ocorre porque a oxihemoglobina (hemoglobina ligada ao oxigênio) e a desoxihemoglobina (hemoglobina sem oxigênio) absorvem luz de maneira diferente em comprimentos de onda distintos. A luz vermelha é mais absorvida pela desoxihemoglobina, que é a forma da hemoglobina sem oxigênio. A luz infravermelha é mais absorvida pela oxihemoglobina, que é a forma da hemoglobina ligada ao oxigênio.

Ao comparar a quantidade de luz absorvida em cada comprimento de onda, o oxímetro pode calcular a proporção de oxihemoglobina em relação à hemoglobina total, o que permite determinar a saturação de oxigênio no sangue.

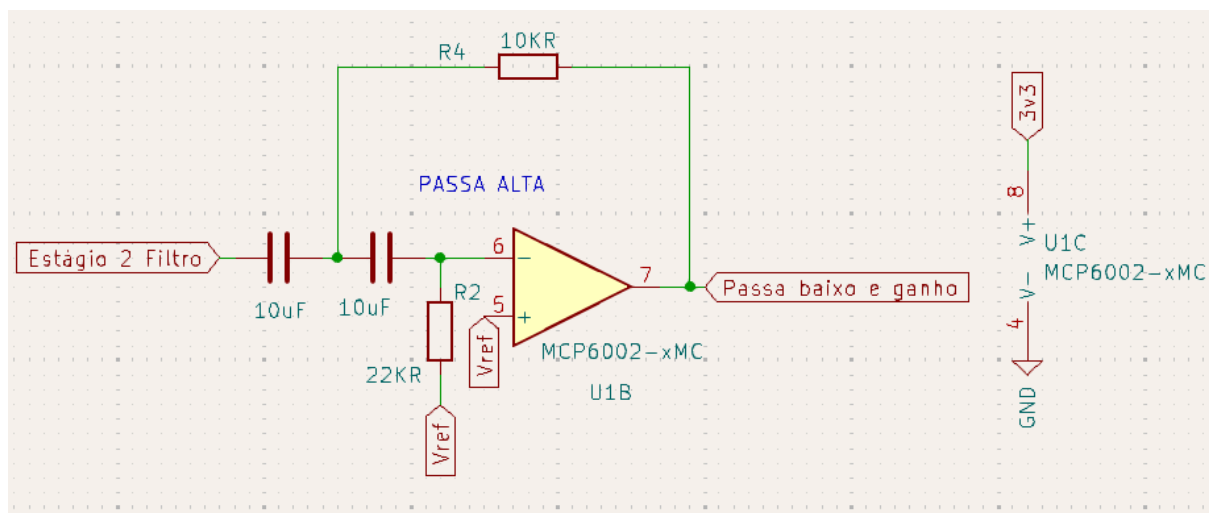
Figura 3: Estágio de captura do sinal para oximetria com um LED vermelho e um LED infravermelho



Fonte: Autoria própria

Um filtro passa-alta de 2ª ordem, ilustrado pela Figura 4, foi utilizado para remover componentes de frequências abaixo de 1Hz que podem interferir na medição precisa do sinal pulsátil de oxigênio no sangue. Essas componentes indesejadas podem incluir: ruído de movimento, variações lentas no ambiente de iluminação e flutuações fisiológicas.

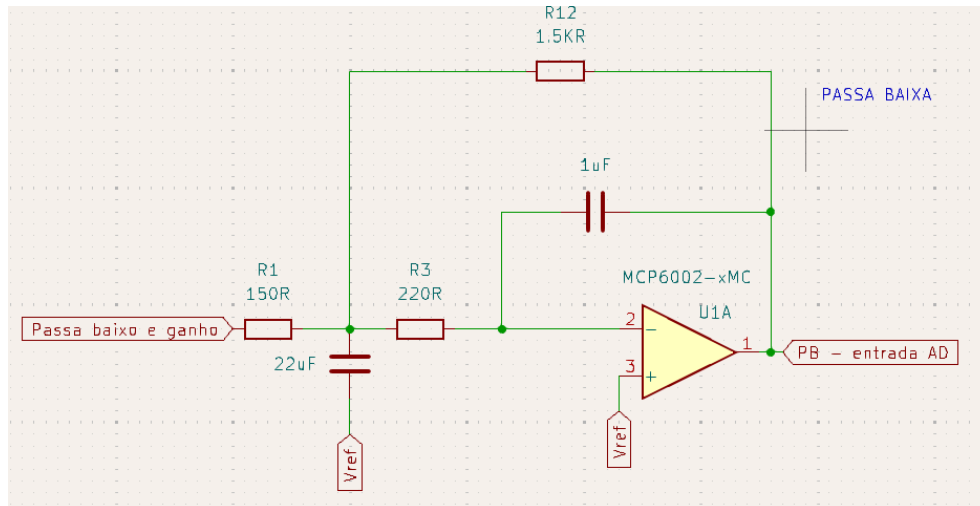
Figura 4: Filtro Passa Alta de 2ª ordem



Fonte: Autoria própria

Um filtro passa-baixa de 60 Hz com um ganho de 10x foi utilizado para melhorar a qualidade do sinal medido pelo dispositivo. O filtro passa-baixa, conforme a Figura 5, é projetado para atenuar ruídos de alta frequência, especialmente a interferência eletromagnética de 60 Hz proveniente de linhas de energia elétrica, que pode distorcer a leitura. Além de rejeitar esse ruído, o filtro também amplifica o sinal útil captado pelo fotodiodo, aumentando sua amplitude em 10 vezes. Isso facilita a detecção e a análise precisa do sinal correspondente ao pulso cardíaco, melhorando a confiabilidade da medição da saturação de oxigênio no sangue.

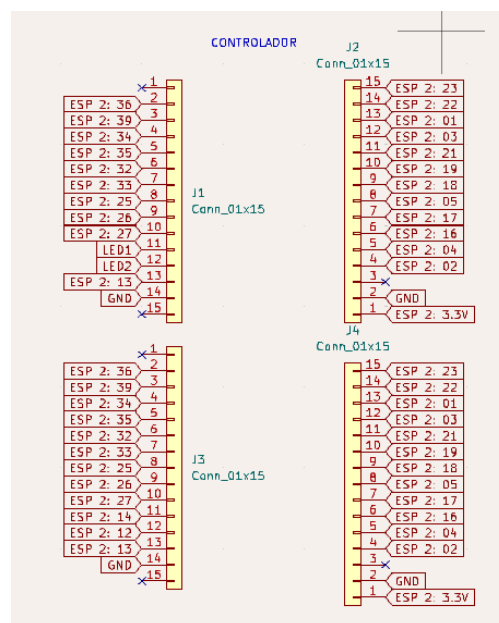
Figura 5: Filtro Passa Baixa de 2ª ordem com ganho



Fonte: Autoria própria

Na Figura 6, ilustra-se o esquemático das portas do microcontrolador, nele são configurados 3 portas GPIO, duas para alternar o acionamento dos LEDs e uma para receber a leitura do sinal captado pelo fotodiodo.

Figura 6: Portas de um microcontrolador ESP32-WROOM-DEVKIT-V1



Fonte: Autoria própria

Lista de componentes:

Nome	“Valor”	Quantidade
C1,C2	10uF	2
C3	22uF	1
C4,C5	1uF	2
D2,D3	MTE8120CP 805nm	2
D4	BPW34	1
J1,J2,J3,J4	Conn_01x15	4
Q1,Q2	BC337	2
R1	150R	1
R2	22KR	1
R3	220R	1
R4,R5,R11	10KR	3

R6,R8	1.8KR	2
R7,R9	100R	2
R10	10kR	1
R12	1.5KR	1
U1	MCP6002-xMC	1
U2	MCP602	1

Algoritmo a ser implementado no ESP32 para gerenciamento do circuito eletrônico:

Os softwares desenvolvidos:

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

// Configurações Globais
#define verm_pin 18
#define infra_pin 19
#define leitura_pin 33

// Variáveis com o estado dos pinos, se estão ou não ativos
bool verm_estado = false;
bool infra_estado = false;

// Variáveis para os valores recebidos
float infra_recebido, verm_recebido;

// Índices para contagem e resultados
float verm_soma, infra_soma; // Soma dos valores dos vetores
float verm_media, infra_media;
float R; // Relação entre os resultados das médias
float spo2; // Saturação de oxigênio no sangue
```

```
const int intervalo = 500; // Intervalo ajustado para quantidade de
dados recebidos (ajuste conforme necessário)

// Vetores para salvar as variáveis
float verm_valores[intervalo], desl_valores[intervalo];
float infra_valores[intervalo];
float verm_subtracao[intervalo], infra_subtracao[intervalo];
float verm_filtrado[intervalo], infra_filtrado[intervalo];

// Variáveis para controle de tempo
unsigned long tempo_anterior_leitura = 0; // Tempo da última leitura
unsigned long tempo_anterior_calculo = 0; // Tempo do último cálculo de
SpO2
const unsigned long intervalo_leitura = 20; // Intervalo em
milissegundos entre leituras (20 ms)
const unsigned long intervalo_calculo = 1000; // Intervalo em
milissegundos para cálculo de SpO2 (1 segundo)

// Configuração Wi-Fi
const char* ssid = "Net do lucas";
const char* password = "12345678";

// Instância do servidor web
WebServer server(80);
void setup() {
    Serial.begin(115200);

    pinMode(verm_pin, OUTPUT);
    pinMode(infra_pin, OUTPUT);

    configurarLEDs(false, false); // Ambos LEDs desligados

    // Conectando à rede Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Conectando ao WiFi...");
    }
    Serial.println("Conectado ao WiFi");
    Serial.println(WiFi.localIP()); // Exibe o IP atribuído

    // Configuração do servidor web
```

```

server.on("/dados", [] () {
    String dados = "SpO2:" + String(spo2);
    server.setHeader("Access-Control-Allow-Origin", "*"); // Adiciona
o cabeçalho CORS
    server.send(200, "text/plain", dados);
});

server.begin();
Serial.println("Servidor iniciado");
}

// Loop principal
void loop() {
    server.handleClient(); // Lida com clientes do servidor

    unsigned long tempo_atual = millis();

    // Verifica se o tempo desde a última leitura é maior que o
intervalo desejado
    if (tempo_atual - tempo_anterior_leitura >= intervalo_leitura) {
        tempo_anterior_leitura = tempo_atual; // Atualiza o tempo da
última leitura

        // Realizar leitura dos LEDs
        realizar_leitura();
        alternar_leds(); // Alterna o estado dos LEDs
    }

    // Verifica se é hora de realizar os cálculos de SpO2
    if (tempo_atual - tempo_anterior_calculo >= intervalo_calculo) {
        tempo_anterior_calculo = tempo_atual; // Atualiza o tempo do
último cálculo

        // Chamada de funções para cálculo de SpO2
        filtro_modocomum();
        calculo_media();
        calculo_final();

        // Plotagem no serial monitor dos resultados finais
        Serial.print("SpO2: ");
        Serial.println(spo2);
        Serial.println();
    }
}

```



```

    delay(1); // Pequeno delay para aliviar o processamento
}

// Função para realizar leitura, dependendo do estado dos LEDs
void realizar_leitura() {
    static int i = 0; // Índice estático para manter a posição entre chamadas

    if (verm_estado) { // Quando o LED vermelho está ligado
        verm_recebido = analogRead(leitura_pin);
        verm_valores[i] = verm_recebido;
        infra_valores[i] = 0;
    } else if (infra_estado) { // Quando o LED infravermelho está
        ligado
        infra_recebido = analogRead(leitura_pin);
        infra_valores[i] = infra_recebido;
        verm_valores[i] = 0;
    } else { // Quando os dois LEDs estão desligados
        desl_valores[i] = analogRead(leitura_pin);
    }

    i = (i + 1) % intervalo; // Incrementa o índice e reseta quando
    atinge o intervalo
}

// Função de filtro de modo comum com filtragem adicional de suavização
void filtro_modocomum() {
    for (int k = 0; k < intervalo; k++) {
        verm_subtracao[k] = verm_valores[k] - desl_valores[k];
        infra_subtracao[k] = infra_valores[k] - desl_valores[k];

        // Filtro passa-baixa simples para suavização
        verm_filtrado[k] = 0.9 * abs(verm_subtracao[k]) + 0.1 *
verm_filtrado[k];
        infra_filtrado[k] = 0.9 * abs(infra_subtracao[k]) + 0.1 *
infra_filtrado[k];
    }
}

// Função para calcular a média dos vetores
void calculo_media() {
    verm_soma = 0;

```

```

infra_soma = 0;

for (int i = 0; i < intervalo; i++) {
    verm_soma += verm_filtrado[i];
    infra_soma += infra_filtrado[i];
}

verm_media = verm_soma / intervalo;
infra_media = infra_soma / intervalo;

// Verifica as médias calculadas
Serial.print("Vermelho Médio: ");
Serial.println(verm_media);
Serial.print("Infravermelho Médio: ");
Serial.println(infra_media);
}

// Função para fazer o cálculo final usando a relação de Beer-Lambert
void calculo_final() {
    if (infra_media != 0) { // Evitar divisão por zero
        R = verm_media / infra_media;
        spo2 = 110 - 25 * R; // Ajuste conforme necessário
        if (spo2 < 70) spo2 = 70;
        if (spo2 > 100) spo2 = 100;
    } else {
        spo2 = 0; // Caso infra_media seja zero, SpO2 é indefinido
        Serial.println("Aviso: Média infravermelha zero, SpO2
indefinido.");
    }
}

// Função para configurar o estado dos LEDs
void configurarLEDs(bool estadoVermelho, bool estadoInfra) {

    digitalWrite(verm_pin, estadoVermelho ? HIGH : LOW);
    digitalWrite(infra_pin, estadoInfra ? HIGH : LOW);
    verm_estado = estadoVermelho;
    infra_estado = estadoInfra;
}

// Função para alternar os estados de LEDs com intervalo mais longo
para estabilidade
void alternar_leds() {

```

```

static unsigned long ultimo_troca = 0;
if (millis() - ultimo_troca > 100) { // Alterna a cada 100ms para
estabilidade
    if (!verm_estado && !infra_estado) {
        configurarLEDs(true, false); // Liga vermelho
    } else if (verm_estado && !infra_estado) {
        configurarLEDs(false, true); // Liga infravermelho
    } else {
        configurarLEDs(false, false); // Desliga ambos
    }
    ultimo_troca = millis();
}
}

```

Imagens do circuito funcionando:

