Protótipo de monitor cardíaco utilizando o módulo ESP32 e o sensor AD8232

Pedro Linhares Tristão Moreira*

10 de dezembro de 2020

Resumo

O eletrocardiograma é um dos exames cardiológicos mais importantes e simples para o diagnóstico de doenças cardiovasculares. Fora do ambiente hospitalar, dispositivos inteligentes trazem esta tecnologia como forma de monitoramento durante exercícios e até monitoramento do ritmo cardíaco. Este projeto mostra o desenvolvimento de um monitor cardíaco utilizando um módulo de desenvolvimento do microcontrolador ESP32 e o sensor de monitoramento cardíaco AD8232.
Palavras-chave: Engenharia biomédica. Monitor cardíaco. Eletrocardiografia. Detecção de ondas R.

Abstract

The electrocardiogram is one of the most important and simple procedures when it comes to diagnosing cardiovascular diseases. Outside the hospital environment, smart devices bring this technology to patients as a way to monitor their heart rate during exercise sessions or even as a way to monitor the heart rhythm. This project focus on the development of a heart rate monitor using the ESP32 microcontroller development module and the heart rate sensor AD8232.
Keywords: Biomedical Engineering. Heart monitor. Electrocardiography. R-wave detection.

1 Introdução

Em 2016, 17,9 milhões de pessoas morreram de doenças cardiovasculares, o que representa, aproximadamente, 31% de todas as mortes no ano (OMS, 2017), e muito mais pessoas são portadoras destas doenças que requerem constante cuidado e atendimento. O eletrocardiograma (ECG) é um dos exames cardiológicos não-invasivos mais realizados para registro das atividades elétricas do coração, capaz de detectar anormalidades no ritmo cardíaco e muitos outros problemas cardíacos comuns, como arritmia e cardiomiopatia, por exemplo (NHS-UK, 2018).

Recentemente, com a disseminação de dispositivos inteligentes, como os *smartwatches* (relógios inteligentes), esta tecnologia se aproximou mais do paciente, permitindo que ele monitore sua frequência cardíaca durante sessões de exercícios físicos ou mesmo seja avisado em seu dispositivo celular sobre problemas mais sérios, como pulso irregular (PEREZ et al., 2019). Estes dispositivos, porém, ainda podem ser muito caros para o contexto econômico do Brasil.

A utilização do eletrocardiograma para a determinação da frequência cardíaca, diferentemente do uso para detecção de algumas anormalidades, não necessita de tanto tratamento e processamento do sinal. Desta forma, este trabalho propõe o desenvolvimento de um protótipo de

monitor cardíaco capaz de calcular a frequência cardíaca de pacientes e a sua comparação com outras soluções encontradas no mercado.

2 Referencial Teórico

Para a realização deste projeto foram necessárias as informações e conceitos teóricos apresentados abaixo.

2.1 Microcontrolador ESP32

Fabricado pela *Espressif Systems*, a família de microcontroladores ESP32 é muito utilizada em aplicações de internet das coisas (IoT) por causa de seu baixo consumo de energia, alta performance e custo reduzido. O *kit* de desenvolvimento utilizado neste projeto integra um microprocessador de dois núcleos de 32 bits com WiFi e Bluetooth, além de todo o circuito analógico necessário para a regulação da tensão de alimentação e do circuito responsável pela comunicação USB. Por ser compatível com *protoboards* padrões, a prototipagem de circuitos com este módulo é simples e não requer soldagem.

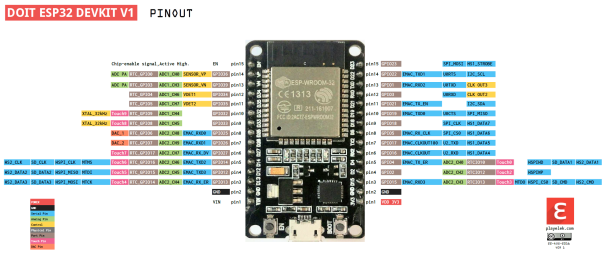

Figura 1 – Mapa de pinos do módulo utilizado

Fonte: Microcontrollerslab (2019)

Além das entradas e saídas digitais, esta família de microcontroladores possui diversos periféricos, como:

- Conversor analógico-digital (ADC)
- Conversor digital-analógico (DAC)
- I2C
- I2S
- *Timers*
- UART
- SPI *Master* e *Slave*
- etc.

O conversor analógico digital é um dos periféricos mais importantes para este projeto, pois permite a aquisição de dados de sensores analógicos. Além disso, os *timers* podem ser utilizados para definir a frequência de amostragem dos sensores.

2.2 Sensor AD8232

O circuito integrado AD8232 é um sensor de monitoramento cardíaco produzido pela *Analog Devices*. Apresenta baixo consumo de energia e filtragem interna do sinal do eletrocardiograma. Sua pinagem é apresentada na figura 2.

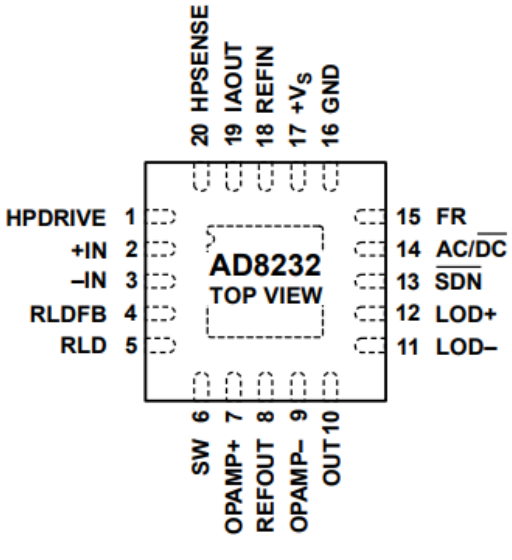

Figura 2 – Pinagem do sensor AD8232

Fonte: *Analog Devices* (2012)

Para facilidade de prototipagem, foi utilizado um módulo da *Sparkfun* que utiliza este sensor e possui todo o circuito analógico necessário para seu funcionamento, além do conector para os eletrodos. Existem 6 pinos de entrada neste módulo, listados abaixo:

1. GND: Terminal negativo da alimentação (terra)
2. 3.3v: Terminal positivo da alimentação (3.3V)

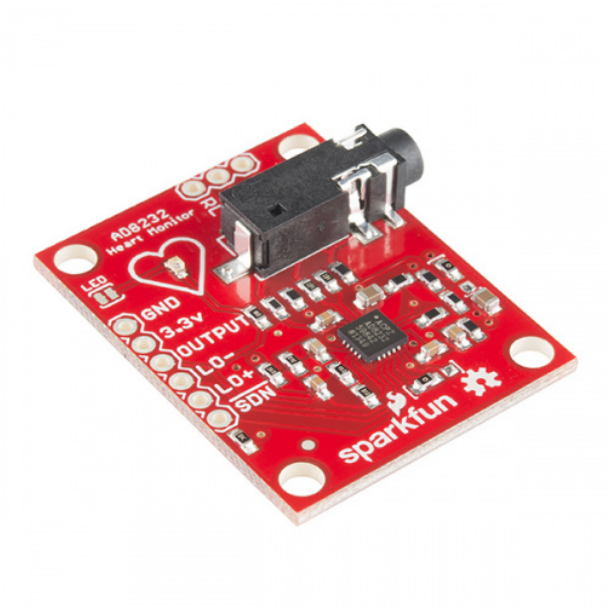

Figura 3 – Módulo de monitoramento cardíaco

Fonte: *Sparkfun* (2020)

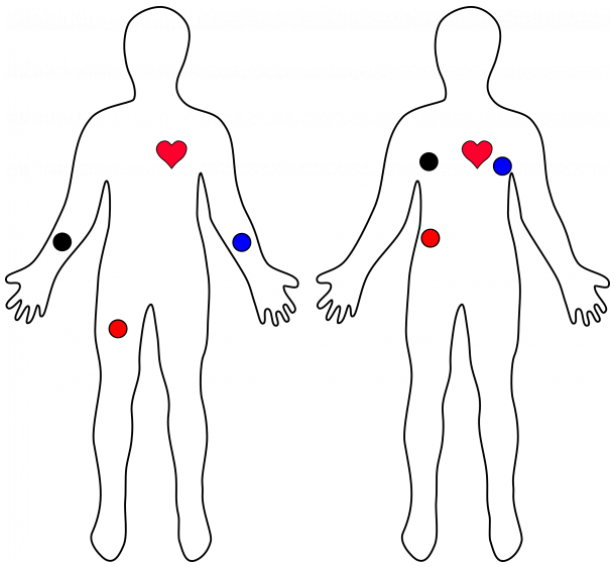

Figura 4 – Posicionamento dos eletrodos

Fonte: *Sparkfun* (2020)

3. OUTPUT: Saída analógica do eletrocardiograma
4. LO-: Saída do comparador interno de *Leads off* para a entrada negativa: Este pino indica se o eletrodo conectado na entrada IN- está desconectado.
5. LO+: Saída do comparador interno de *Leads off* para a entrada positiva: Este pino indica se o eletrodo conectado na entrada IN+ está desconectado.
6. SDN: Controle de desligamento: Ao colocar este pino em nível lógico baixo o sensor entra em modo de baixo consumo de energia. O módulo, por padrão, deixa este pino conectado à alimentação através de um resistor de *pull-up*.

O sensor utiliza 3 eletrodos, identificados pelas cores azul, vermelho e preto, que devem ser conectados ao paciente em uma das formas que mostra a figura 4.

2.3 FreeRTOS

O FreeRTOS é o sistema operacional de tempo real (RTOS) de código aberto mais utilizado no mundo. Ele é pequeno e portátil, e dá suporte a mais de 30 arquiteturas de hardware diferentes.

O uso de um RTOS em um sistema embarcado permite que o programador separe as diferentes funções do programa em tarefas, que podem ser executadas por um *kernel* cooperativo ou preemptivo. No caso cooperativo, o sistema dá controle total do processador à uma tarefa e só retoma o controle quando a tarefa termina de ser executada. Já no preemptivo, o sistema operacional consegue interromper a execução de uma tarefa e dar o controle do processador à outra. Esta troca de contexto requer maior processamento por parte do sistema operacional mas garante muitos benefícios para sistemas que exigem operações em tempo real.

Além da gestão de tempo, o FreeRTOS dispõe de diversas ferramentas que facilitam a comunicação entre tarefas e a gestão do fluxo de dados no programa. As filas são uma das alternativas disponíveis para comunicação entre tarefas ou entre rotinas de interrupção e tarefas.

2.4 Eletrocardiografia

A eletrocardiografia consiste em produzir um diagrama de tensão por tempo da atividade elétrica do coração. Em condições normais de saúde este gráfico das atividades elétricas segue um padrão chamado ritmo sinusal normal.

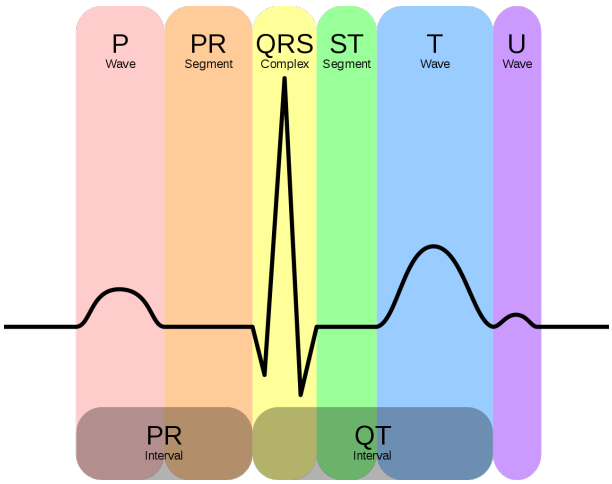

Figura 5 – Ritmo sinusal

Fonte: Wikipedia (2019)

Nota-se que o diagrama é composto por diversos segmnetos. A análise do comportamento de todas as seções do diagrama pode diagnosticar vários problemas cardíacos. Para a determinação da frequência cardíaca, porém, pode ser utilizado apenas o complexo QRS, mostrado na imagem acima em amarelo. Este complexo representa a despolarização dos ventrículos e o impulso elétrico detectado geralmente tem amplitude de pico maior do que as ondas P e T (RAWSHANI, 2020).

2.5 Algoritmo Z-Score

O algoritmo Z-Score é baseado no conceito estatístico de pontuação padrão (*Standard Score*), atribuindo a cada ponto de uma série de dados um valor que indica sua distância da média em termos de desvio-padrão. A pontuação é calculada pela fórmula:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

Onde x é a amostra, μ é a média do sinal e σ seu desvio-padrão. Pontos que estão longe da média, ou seja, picos, apresentam pontuação alta em módulo. Desta forma, a criação de um algoritmo que aplique esta pontuação em uma série temporal pode ser útil para a detecção de picos.

Para séries obtidas em tempo real, pode-se utilizar o algoritmo Z-score suavizado (BRAKEL, 2014). Este algoritmo utiliza média e desvio padrões móveis para comparar a amostra atual com o sinal como um todo e precisa de 3 parâmetros:

- *lag*: Indica quantas amostras anteriores serão suavizadas pelo algoritmo. Um valor alto aumenta a robustez do algoritmo para séries estacionárias, enquanto um valor baixo aumenta a adaptibilidade do algoritmo à séries de média variante no tempo.
- *influence*: Determina o quanto o sinal influencia na detecção do algoritmo. Para sequências com média variante no tempo este valor deve ser escolhido próximo à 1.
- *threshold*: Valor que indica o limite, em termos de desvio-padrão, para a detecção ou não detecção.

3 Metodologia

O desenvolvimento do projeto foi dividido em três etapas: Montagem do protótipo, *Software* embarcado e *Software* de processamento.

3.1 Montagem do protótipo

A montagem do circuito foi realizada em um *protoboard* seguindo o esquemático mostrado abaixo:

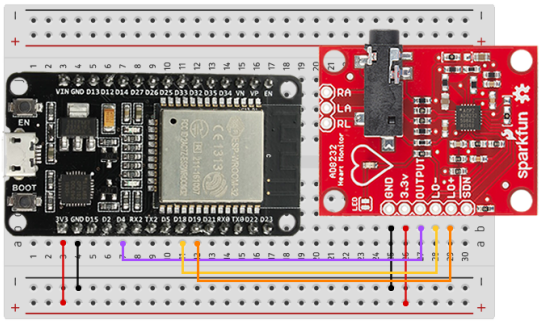

Figura 6 – Esquema do protótipo montado no *protoboard*

Fonte: Autor.

Ambos o microcontrolador e o sensor foram conectados à alimentação de 3.3V. Para o protótipo, a alimentação será feita através de um cabo USB e a tensão será regulada pelo módulo de desenvolvimento ESP32. Desta forma, ele também alimentará o sensor. A saída analógica OUTPUT do sensor foi conectada ao pino D4 do módulo, que corresponde ao canal 0 do conversor analógico digital 2, como mostra a figura 1. Já as saídas LO- e LO+ foram conectadas aos pinos

D18 e D19, respectivamente, que serão configurados como entradas digitais para que possa ser detectado problemas na conexão dos eletrodos.

3.2 *Software* embarcado

O programa embarcado foi desenvolvido em C utilizando o *framework* ESP-IDF no ambiente de desenvolvimento integrado (IDE) *PlatformIO*. O trabalho deste *software* é fazer a aquisição dos dados do sensor e os transmitir para o computador através da interface UART. Para que a frequência cardíaca calculada seja a mais próxima o possível do valor verdadeiro, é importante que as amostras sejam obtidas com uma frequência de amostragem fixa. Para aplicações como esta, em que a análise dos dados do eletrocardiograma no domínio da frequência não é necessária, a amostragem à 100 [Hz] produz resultados aceitáveis (KWON et al., 2018).

Para evitar *overheads* do RTOS, a amostragem do sensor é feita em uma rotina de interrupção do *timer*, configurado para estourar a cada 10 [ms]. Nela, é verificado se os eletrodos estão devidamente conectados checando se ambos os pinos D18 e D19 estão em nível lógico baixo e é realizada a leitura do sensor através do conversor analógico-digital. A amostra lida é enviada para uma fila para que possa ser lida por uma tarefa, responsável por transmití-la ao computador. Desta forma, é possível inserir, se necessário, algoritmos de processamento embarcado mais custosos na tarefa sem gastar muito tempo na rotina de interrupção. O código completo é mostrado abaixo.

```c
/**
 *
 *  ECG sampling
 *  Author: Pedro Linhares <pedrolinhares@unifei.edu.br>
 *
 */
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "driver/gpio.h"
#include "esp_spi_flash.h"
#include "driver/adc.h"
#include "driver/timer.h"
#include "freertos/queue.h"

#define TIMER_INTR_SEL TIMER_INTR_LEVEL
#define TIMER_GROUP TIMER_GROUP_0
#define TIMER_DIVIDER 80
#define TIMER_SCALE (TIMER_BASE_CLK / TIMER_DIVIDER)
#define TIMER_FINE_ADJ (0*(TIMER_BASE_CLK / TIMER_DIVIDER)/1000000)
#define TIMER_INTERVAL0_SEC (0.01)

xQueueHandle queue;

void IRAM_ATTR timer_group0_isr(void *p)
{
    int timer_idx = (int) p;
    int value;
    esp_err_t r = ESP_OK;
    uint32_t intr_status = TIMERG0.int_st_timers.val;
    BaseType_t priorityFlag = pdFALSE;
    if((intr_status & BIT(timer_idx)) && timer_idx == TIMER_0)
    {
        TIMERG0.hw_timer[timer_idx].update = 1;
        TIMERG0.int_clr_timers.t0 = 1;
        TIMERG0.hw_timer[timer_idx].config.alarm_en = 1;
        if(gpio_get_level(19) == 1 || gpio_get_level(18) == 1)
        {
            value = 0;
        }else {
            r = adc2_get_raw(ADC2_CHANNEL_0, ADC_WIDTH_BIT_12, &value);
        }
        if(r == ESP_OK)
            xQueueSendFromISR(queue, &value, &priorityFlag);
    }
}

void timer0_init()
{
    int timer_group = TIMER_GROUP_0;
    int timer_idx = TIMER_0;
    timer_config_t config;
    config.alarm_en = 1;
    config.auto_reload = 1;
    config.counter_dir = TIMER_COUNT_UP;
    config.divider = TIMER_DIVIDER;
    config.intr_type = TIMER_INTR_SEL;
    config.counter_en = TIMER_PAUSE;
    timer_init(timer_group, timer_idx, &config);
    timer_pause(timer_group, timer_idx);
    timer_set_counter_value(timer_group, timer_idx, 0x00000000ULL);
    timer_set_alarm_value(timer_group, timer_idx, (TIMER_INTERVAL0_SEC * TIMER_SCALE) - TIMER_FINE_ADJ);
    timer_enable_intr(timer_group, timer_idx);
    timer_isr_register(timer_group, timer_idx, timer_group0_isr, (void *) timer_idx, ESP_INTR_FLAG_IRAM, NULL);
    timer_start(timer_group, timer_idx);
}

void task_heart(void *pvParameter)
{
    int value = 0;
    for(;;)
    {
        if(xQueueReceive(queue, &value, (TickType_t)(1000/portTICK_RATE_MS)) == pdTRUE)
```

```
        {
            printf("%d\n", value);
            //Processamento dos dados se necess rio
        }
    }
}

void app_main(void)
{
    queue = xQueueCreate(10, sizeof(int));
    gpio_pad_select_gpio(2);
    gpio_set_direction(2, GPIO_MODE_OUTPUT);
    gpio_pad_select_gpio(19);
    gpio_set_direction(19, GPIO_MODE_INPUT);
    gpio_pad_select_gpio(18);
    gpio_set_direction(18, GPIO_MODE_INPUT);
    gpio_set_level(2, 0);
    adc2_config_channel_atten(ADC2_CHANNEL_0, ADC_ATTEN_11db);
    timer0_init();
    xTaskCreate(&task_heart, "transmitter", 2048, NULL, 5, NULL);
}
```

3.3 *Software* de processamento

Para processar os dados amostrados pelo microcontrolador, foi projetado um programa para a plataforma *Windows* em C#. O objetivo deste *software* é mostrar o gráfico dos dados e, utilizando o algoritmo z-score, calcular a frequência cardíaca do paciente. Abaixo é mostrada a implementação deste algoritmo:

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Markup;
using static System.Math;

namespace MonitorCardiaco
{
    class zscore
    {
        const int BUFFER_SIZE = 1500;
        private double threshold, influence;
        private int lag, samples, k;

        double[] y, filteredY, signals, avgFilter, stdFilter;

        private double mean(List<double> lista)
        {
            return lista.Average();
        }

        private double std(List<double> lista)
        {
            double ret = 0;
            if(lista.Count() > 0)
            {
                double avg = lista.Average();
                double sum = lista.Sum(d => Math.Pow(d-avg, 2));
                ret = Math.Sqrt((sum) / (lista.Count()-1));
            }
            return ret;
        }


        private void resetVectors()
        {
            for(int i = 0; i < BUFFER_SIZE; i++)
            {
                this.signals[i] = 0.0;
                this.avgFilter[i] = 0.0;
                this.stdFilter[i] = 0.0;
                this.filteredY[i] = 0;
            }
            var janela = new List<double>(this.filteredY).Skip(0).Take(this.lag).ToList();
            this.avgFilter[lag - 1] = this.mean(janela);
            this.stdFilter[lag - 1] = this.std(janela);
        }

        public zscore(int lag, double threshold, double influence)
        {
            this.lag = lag;
            this.threshold = threshold;
            this.influence = influence;
            this.samples = 0;
            this.y = new double[BUFFER_SIZE];
            this.filteredY = new double[BUFFER_SIZE];
            this.signals = new double[BUFFER_SIZE];
            this.stdFilter = new double[BUFFER_SIZE];
            this.avgFilter = new double[BUFFER_SIZE];
        }
        /***
         * Processa novo ponto recebido
         * @params newData: dado amostrado
         * @returns batida detectada
        ***/
        public bool processar(double newData)
        {
            this.y[this.samples] = (double)newData;
            this.k = this.samples++;
            if(this.k < this.lag)
            {
                return false;
            }
            else if(this.k == this.lag)
            {
                this.resetVectors();
                return false;
            }

            this.signals[this.k] = 0;
            this.filteredY[this.k] = 0;
            this.avgFilter[this.k] = 0;
```

```
85                  this.stdFilter[this.k] = 0;
86
87                  if(Math.Abs(this.y[this.k] -
                        this.avgFilter[this.k-1])
                        > this.threshold * this.
                        stdFilter[this.k-1])
88                  {
89                      if(this.y[this.k] > this.
                            avgFilter[this.k-1])
90                      {
91                          this.signals[k] = 1;
92                      }
93                      else
94                      {
95                          this.signals[k] = -1;
96                      }
97                      this.filteredY[this.k] =
                            this.influence * this.
                            y[this.k] + (1 - this.
                            influence) * this.
                            filteredY[this.k - 1];
98                      var janela = new List<
                            double>(filteredY).
                            Skip(this.k - this.lag
                            ).Take(this.lag + 1).
                            ToList();
99                      this.avgFilter[this.k] =
                            this.mean(janela);
100                     this.stdFilter[this.k] =
                            this.std(janela);
101
102
103                 }
104                 else
105                 {
106                     this.signals[this.k] = 0;
107                     this.filteredY[this.k] =
                            this.y[this.k];
108                     var janela = new List<
                            double>(filteredY).
                            Skip(this.k - this.lag
                            ).Take(this.lag + 1).
                            ToList();
109                     this.avgFilter[this.k] =
                            this.mean(janela);
110                     this.stdFilter[this.k] =
                            this.std(janela);
111                 }
112
113                 if (this.samples >=
                        BUFFER_SIZE)
114                 {
115                     resetVectors();
116                     this.samples = 0;
117                 }
118                 return (this.signals[this.k]
                        == 1);
119             }
120         }
121 }
```

A interface gráfica do programa foi feita no Visual Studio e é mostrada na figura abaixo.

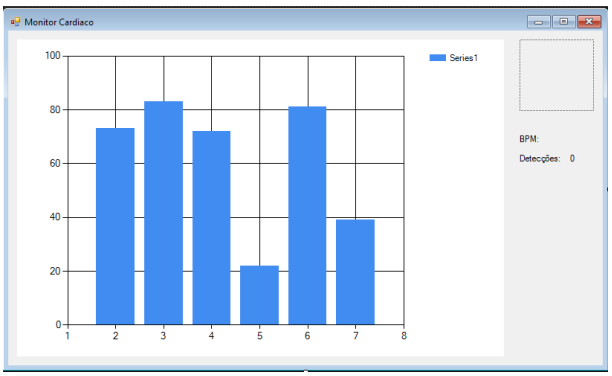

Figura 7 – Interface do monitor cardiáco

Fonte: Autor.

4 Resultados

Foram realizados testes em dois pacientes e a frequência cardíaca calculada pelo protótipo foi comparada à frequência cardíaca calculada pelo aplicativo *Samsung Health* em um celular Galaxy S5, que possui um sensor de frequência cardíaca. Os resultados são apresentados abaixo:

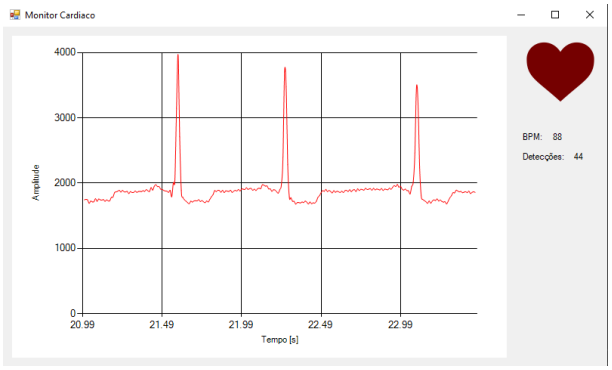

Figura 8 – Monitor cardíaco durante os testes

Fonte: Autor.

Paciente	BPM (Celular)	BPM (Protótipo)
Paciente 1	90	88
Paciente 2	67	67

Tabela 1 – Resultados dos testes

Percebe-se que os valores de frequência cardíaca calculados pelo protótipo foram bem próximos dos valores obtidos com o uso do celular. Logo, concluiu-se que o objetivo do projeto foi alcançado: Foi possível, utilizando os componentes

escolhidos, a criação de um monitor cardíaco que estima a frequência cardíaca do paciente. Na tabela 2 são exibidos os preços dos componentes utilizados para a montagem do protótipo.

Componente	Preço (R$)
Módulo ESP32	79,90
Módulo AD823	128,16
Total	208,06

Tabela 2 – Preços dos componentes

Um oxímetro de pulso da marca Multilaser que, além de calcular a frequência cardíaca também indica a saturação de oxigênio no sangue (SaO2) do paciente, foi encontrado para compra por R$ 101,37 na loja Submarino.

5 Considerações Finais

Neste trabalho foi desenvolvido, em 3 etapas, um protótipo de monitor de frequência cardíaca capaz de obter os dados diretamente do paciente, gerar o gráfico do eletrocardiograma em tempo real e estimar a sua frequência cardíaca. Este protótipo, quando comparado à uma solução existente em um aparelho celular, apresentou resultados semelhantes e possui a vantagem de mostrar o gráfico do sinal. Porém, a solução presente no celular é muito mais portátil e de fácil uso do que a apresentada neste projeto. Além disso, para que o processamento dos dados ocorra de forma correta, é necessário que o paciente esteja completamente parado. Durante os testes, foi notado que movimentos do paciente causavam um desvio na linha de base do sinal, gerando erros na detecção de picos e no cálculo da frequência cardíaca. Embora o aplicativo de celular também tenha problemas com a movimentação do paciente, o protótipo criado se mostrou muito mais sensível à estas variações. Este e outros problemas relacionados ao processamento de sinais poderiam ser atenuados com uma filtragem mais cuidadosa do sinal feita em *software*.

Em relação à custo, o protótipo final foi mais de 2 vezes mais caro do que um oxímetro de pulso comercial. Uma das razões para isto, porém, é a utilização de módulos. A criação de uma placa de circuito impresso e a utilização de componentes discretos deve diminuir consideravelmente o preço do aparelho, se produzido em escala maior.

Por fim, percebe-se que este trabalho pode ser modificado de diversas formas para melhorias. Pode-se, por exemplo, fazer o processamento do sinal no próprio microcontrolador: O ESP32 possui elevado poder computacional e é capaz de realizar a filtragem e detecção de picos de maneira embarcada. Pode-se também utilizar as tecnologias de comunicação sem fio que estão disponíveis no sistema (WiFi e Bluetooth) para, por exemplo, se comunicar com aplicativos de saúde e telemedicina em um aparelho celular.

Referências

BRAKEL, J. v. *Robust peak detection algorithm using z-scores*. 2014. Disponível em: <https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362>. Citado na página 4.

KWON, O. et al. Electrocardiogram sampling frequency range acceptable for heart rate variability analysis. *Healthcare Informatics Research*, v. 24, p. 198, 07 2018. Citado na página 5.

NHS-UK. *Electrocardiogram (ECG)*. 2018. Disponível em: <https://www.nhs.uk/conditions/electrocardiogram/>. Citado na página 1.

OMS. *Cardiovascular diseases (CVDs)*. 2017. Disponível em: <https://www.who.int/en/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)>. Citado na página 1.

PEREZ, M. V. et al. Large-scale assessment of a smartwatch to identify atrial fibrillation. *New England Journal of Medicine*, v. 381, n. 20, p. 1909–1917, 2019. Disponível em: <https://doi.org/10.1056/NEJMoa1901183>. Citado na página 1.

RAWSHANI, A. *Clinical ECG interpretation*. 2020. Disponível em: <https://ecgwaves.com/course/the-ecg-book/>. Citado na página 4.