**UNIVERSIDADE FEDERAL DE SANTA MARIA**
**CENTRO DE TECNOLOGIA**
**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

# IMPLEMENTAÇÃO DE UM SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS DE OXIMETRIA COM COMUNICAÇÃO BLUETOOTH E SOFTWARE DE MONITORAMENTO

**TRABALHO DE CONCLUSÃO DE CURSO**

**Sandra Cossul**

**Santa Maria, RS, Brasil**

**2015**

# IMPLEMENTAÇÃO DE UM SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS DE OXIMETRIA COM COMUNICAÇÃO BLUETOOTH E SOFTWARE DE MONITORAMENTO

**Sandra Cossul**

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
**Graduação em Engenharia de Computação**

**Orientador: Prof. Dr. César Augusto Prior**

**Santa Maria, RS, Brasil**
**2015**

**UNIVERSIDADE FEDERAL DE SANTA MARIA**
**CENTRO DE TECNOLOGIA**
**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

**A Comissão Examinadora, abaixo assinada, aprova o Trabalho de Conclusão de Curso**

# IMPLEMENTAÇÃO DE UM SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS DE OXIMETRIA COM COMUNICAÇÃO BLUETOOTH E SOFTWARE DE MONITORAMENTO

elaborada por
**Sandra Cossul**

como requisito parcial para obtenção do grau de
**Graduação em Engenharia de Computação**

**COMISSÃO EXAMINADORA:**

**César Augusto Prior, Dr.**
(Orientador)

**Carlos Henrique Barriquelo, Dr.** (UFSM)

**Mateus Beck Rutzig, Dr.** (UFSM)

**Santa Maria, RS, Brasil**
**2015**

## AGRADECIMENTOS

Aos meus pais e irmãos, pelo incentivo e apoio durante todo o curso.

Ao meu professor e orientador César Prior, pelo suporte, disponibilidade de tempo e incentivo.

Ao Alex Veloso, pela paciência, ajuda e interesse.

Ao meu namorado João Pedro, pela motivação, compreensão e companheirismo.

Aos amigos e colegas sempre presentes, pela ajuda, sugestões e amizade.

A todos os professores que contribuíram para a minha formação acadêmica.

Obrigado a todos!

# RESUMO

Trabalho de Graduação

Engenharia de Computação

Universidade Federal de Santa Maria

## IMPLEMENTAÇÃO DE UM SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS DE OXIMETRIA COM COMUNICAÇÃO BLUETOOTH E SOFTWARE DE MONITORAMENTO

AUTOR: SANDRA COSSUL

ORIENTADOR: PROF.DR. CÉSAR AUGUSTO PIOR

Data e local da defesa: Santa Maria, 10 de Novembro de 2015.

O presente trabalho apresenta um estudo e implementação de um sistema de aquisição e processamento de sinais de oximetria utilizando-se o oxímetro de pulso. Este dispositivo fornece informações relativas a porcentagem de saturação de oxigênio no sangue arterial, a taxa de batimento cardíaco por minuto e a curva pletismográfica. O sistema também inclui a transmissão dos dados para um dispositivo móvel através do protocolo de comunicação Bluetooth. Este trabalho aborda os conceitos envolvidos na oximetria, o projeto de um modelo de *hardware* utilizando o circuito integrado AFE4490 em conjunto com o microcontrolador Arduino e o módulo de Bluetooth HC06 além do desenvolvimento dos *softwares* envolvidos que incluem o código Arduino, o aplicativo Android e o código Matlab utilizado para visualização dos dados. A análise dos resultados demonstra um funcionamento satisfatório do sistema, já que os dados adquiridos estão próximos dos valores obtidos por um equipamento de oximetria de uso comercial.

Palavras-chave: Oximetria, Processamento de Sinais, Comunicação Bluetooth.

# ABSTRACT

Undergraduate Final Work

Computer Engineering

Federal University of Santa Maria

## IMPLEMENTAÇÃO DE UM SISTEMA DE AQUISIÇÃO E PROCESSAMENTO DE SINAIS DE OXIMETRIA COM COMUNICAÇÃO BLUETOOTH E SOFTWARE DE MONITORAMENTO

AUTHOR: SANDRA COSSUL

TUTOR: PROF.DR. CÉSAR AUGUSTO PIOR

Presentation Date and Local: Santa Maria, November 10th of 2015

The present work introduces a study and implementation of a data acquisition and data processing system of oximetry signals from a pulse oximeter. This device provides information about the percentage of arterial oxygen saturation, heartbeat per minute and the plethysmograph curve. The system also includes data transmission to a mobile device using Bluetooth as the communication protocol. This work approaches oximetry concepts, the project of a hardware model using the integrated circuit AFE4490 with the Bluetooth module HC06 and also, the development of the Arduino code; the Android application and the Matlab software used for visualize data. The analysis of the results demonstrates a satisfactory system operation, since the values of the acquired data are close to the values obtained from a commercially used oximetry.

Keywords: Oximetry, Signal Processing, Bluetooth

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ANEXOS

# LISTA DE ABREVIATURAS E SIGLAS

Sp$0_2$ – nível de saturação de oxigênio

$0_2$ - oxigênio

Hb – hemoglobina

Hb$0_2$ - oxiemoglobina

BPM – batimentos por minuto

CI – circuito integrado

LED – *light emitting diode*

RMS – *root mean square*

SPI – *serial peripheral interface*

AFE – *analog front-end*

ADC – *analog-digital converter*

# SUMÁRIO

# 1. INTRODUÇÃO

O desenvolvimento das tecnologias de comunicação, da capacidade de processamento e integração dos componentes eletrônicos e sistemas promovem a evolução e a melhora dos equipamentos médicos. Também, torna-se possível a implementação de sistemas de telemonitoramento de pacientes, que permitem a observação e registro das informações fisiológicas do indivíduo em tempo real para acompanhamento e avaliação de um especialista. As informações monitoradas incluem sinais biomédicos não invasivos tais como: frequência cardíaca e respiratória, pressão arterial, nível de glicose no sangue, temperatura e nível de saturação do oxigênio (oximetria). O processamento e transmissão desses dados cria um sistema de comunicação constante entre médico e paciente. Este sistema pode ser utilizado em diversas situações trazendo benefícios tais como:

- Monitoramento à distância (*home monitoring*) de pessoas idosas ou pessoas que sofrem de doenças crônicas.
- Monitoramento em hospitais durante processos cirúrgicos, unidades de tratamento intensivo e situações de emergência.
- Acesso médico a áreas de difícil acesso.
- Possibilidade de diagnósticos realizados remotamente.
- Auxílio na realização de diagnósticos com maior rapidez e precisão.
- Facilidade de acesso e armazenamento de informações [1, 2].

O monitoramento em tempo real da oxigenação sanguínea é de extrema importância para a verificação do estado de saúde do paciente, sendo possível a detecção imediata de hemorragias e intoxicações por monóxido de carbono. Além disso, auxilia em diagnósticos prévios de anemia, problemas em recém-nascidos e a verificação de despressurização em aeronaves. Atualmente, o oxímetro de pulso é a técnica mais utilizada para fazer a leitura do nível de oxigenação do paciente.

O oxímetro de pulso é um equipamento médico não invasivo que fornece informações relativas a saturação de oxigênio no sangue arterial além das funções cardiorrespiratórias do paciente como o número de batimentos cardíacos por minuto e a curva de fluxo sanguíneo ou pletismográfica. Por ser um dispositivo versátil e de rápida resposta, é amplamente utilizado durante processos de anestesiologia, em processos cirúrgicos e emergenciais, em unidades de tratamento intensivo, transporte de pacientes, em unidades neonatais, diagnóstico de distúrbios respiratórios e em

outras situações em que o monitoramento se faz necessário para a detecção prévia de problemas associados à baixa taxa de oxigênio presente nos tecidos do corpo, conhecido como hipóxia [3, 4].

Portanto, este trabalho prevê a utilização de um oxímetro de pulso e a interpretação dos dados do mesmo com os seguintes objetivos apresentados abaixo.

## 1.1 Objetivo

O objetivo deste trabalho é a aquisição e interpretação de sinais provenientes de um sensor de oximetria, que compreende uma ponteira com diodos LEDs e um circuito de condicionamento de sinais, a transmissão destes sinais para um microprocessador, deste para um sistema portátil utilizando protocolo de comunicação *Bluetooth* e finalmente um *software* para armazenamento de dados e monitoramento.

## 1.2 Metodologia

A metodologia para o desenvolvimento deste trabalho prevê as seguintes fases:

1. **Revisão da Literatura**
   - ✓ Estudo sobre os fundamentos básicos das técnicas de medidas de oximetria e da implementação de oxímetros.
2. **Planejamento e Especificação do Sistema.**
   - ✓ Especificação do sistema em nível de blocos;
   - ✓ Análise sobre os blocos principais e sua viabilidade técnica para este projeto. Escolha de componentes (blocos do sistema disponíveis comercialmente).
   - ✓ Revisão das especificações e do sistema após busca de dispositivos ou blocos disponíveis.
3. **Projeto de *Hardware***
   - ✓ Projeto de *hardware* para integração física dos blocos componentes do sistema de aquisição de dados de oximetria.
4. **Projeto de *Software***

- ✓ Desenvolvimento de *software* de gerenciamento da comunicação e transferência de dados entre os blocos e as interfaces no microprocessador.
- ✓ Desenvolvimento de um *software* para recepção, armazenamento, interpretação e demonstração dos resultados no dispositivo portátil, (*tablet ou smartphone).*

5. **Construção de um modelo**
   - ✓ Organização e conexão dos dispositivos físicos e lógicos do projeto.
6. **Testes e Verificação dos Resultados**
   - ✓ Testes de leitura e análise dos resultados obtidos.

## 1.3 Estrutura

O trabalho está dividido em 6 seções, onde a primeira apresenta a introdução com os objetivos e a metodologia empregada neste trabalho.

A segunda seção apresenta a fundamentação teórica subdividida em uma consideração biomédica sobre a oxigenação sanguínea e uma abordagem técnica do oxímetro de pulso demonstrando seu princípio de funcionamento e suas limitações, além de uma implementação simples do sistema.

Na terceira seção é descrito o sistema a ser implementado neste trabalho de forma geral, seguindo então, de forma mais específica com o detalhamento dos módulos de *hardware* e *software* integrantes do trabalho, além dos protocolos de comunicação que serão utilizados.

A quarta seção faz um detalhamento do desenvolvimento do trabalho e a implementação do sistema descrito na seção anterior incluindo o projeto de hardware e software.

Por fim, a quinta seção apresenta os resultados.

# 2. FUNDAMENTAÇÃO TEÓRICA

## 2.1 Oxigenação Sanguínea

O oxigênio ($O_2$) é imprescindível para a sobrevivência dos seres vivos, já que todas as células requerem $O_2$ para a correta realização do seu metabolismo. A circulação sanguínea é responsável pela distribuição de oxigênio para todos os tecidos do corpo através dos sistemas circulatório e respiratório, e a falta do mesmo por alguns minutos pode provocar desmaios além de causar sequelas como perda de habilidades motoras ou levar ao coma.

O transporte de oxigênio no sangue é um processo cíclico em que o $O_2$ é inspirado para os pulmões através da respiração atingindo os alvéolos pulmonares onde passa pelo processo de hematose ou troca gasosa alveolar. O processo de hematose consiste na troca do sangue venoso (rico em gás carbônico) pelo sangue arterial (rico em oxigênio). O sangue arterial sai dos pulmões e é bombeado pelo coração atingindo todos os tecidos e órgãos fornecendo o oxigênio às células e retirando os produtos que foram eliminados do metabolismo celular como água e gás carbônico, tornando-se sangue venoso. O sangue venoso então volta para os pulmões e o ciclo é reiniciado.

Para o transporte efetivo de oxigênio, durante o processo de hematose, o oxigênio se combina à hemoglobina (Hb) presente no sangue, formando a oxiemoglobina (Hb$O_2$), conforme demonstrado na Figura 2.1. Ao passar pelo pulmão, o sangue venoso está com as moléculas de hemoglobina saturadas em 64 % de oxigênio, enquanto que o sangue arterial tem suas moléculas de hemoglobina saturadas em 96 % de moléculas de oxigênio, as quais serão gradualmente liberadas para as células.

Uma importante observação é que a hemoglobina muda de tonalidade de acordo com o seu nível de oxigenação e a partir da diferença de cor entre a Hb e a Hb$O_2$, a medição da oxigenação sanguínea pode ser realizada através do oxímetro de pulso [3, 6].

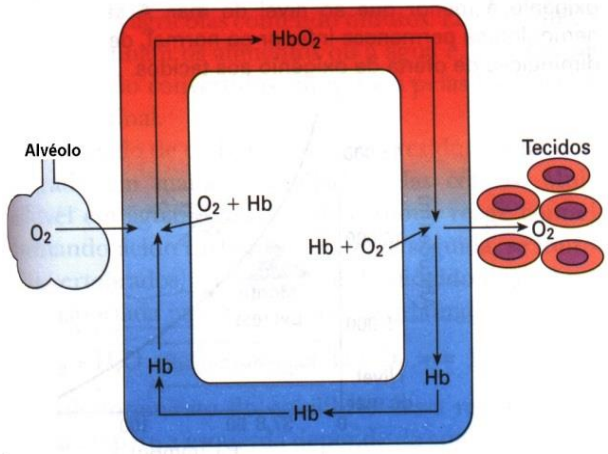



Figura 2.1. Processo de hematose

## 2.2 O Oxímetro de Pulso

O oxímetro de pulso é um aparelho de diagnóstico médico, conforme Figura 2.2, que monitora o nível de saturação de oxigênio presente no sangue arterial ($SpO_2$), além de fornecer informações cardiorrespiratórias do paciente como a frequência cardíaca e a curva de pulsação do sangue, conhecida como curva pletismográfica.


Figura 2.2. Oxímetro de Pulso

A maioria dos oxímetros de pulso consiste em um sensor e um microprocessador que fazem a aquisição e o processamento dos sinais, e um display para mostrar a saturação de oxigênio e os batimentos cardíacos. Em alguns casos, contém um alarme para indicar baixos níveis de $SpO_2$ ou alterações na frequência cardíaca. Usualmente, o oxímetro é posicionado na ponta dos dedos ou lóbulo da orelha por serem regiões com maior concentração de vasos sanguíneos.

A porcentagem de oxigênio no sangue também pode ser medida em laboratório a partir de amostras de sangue, porém é um método invasivo, expõe o paciente a

possíveis contaminações e não traz resultados imediatos. Dessa forma, o oxímetro de pulso apresenta algumas vantagens em relação a outras técnicas, já que define os resultados em um curto período de tempo de forma não invasiva além de ser um dispositivo confiável e seguro. Por isso, é amplamente utilizado em hospitais durante processos de anestesiologia, em processos cirúrgicos e emergenciais, em unidades de tratamento intensivo, durante o transporte de pacientes, para a identificação de anemia, em unidades neonatais e também para o monitoramento à distância [7].

A partir da interpretação dos resultados, o nível de oxigenação $\%\mathrm{Sp}\,\mathrm{O}_2$ considerada normal está entre 97% e 99%. Algumas pessoas, especialmente fumantes podem apresentar leituras entre 93% e 95%. No entanto, leituras de 90% ou menos devem ser tratadas como emergência clínica a partir da detecção de hipóxia, que é um baixo teor de oxigênio nos tecidos. Com relação aos batimentos cardíacos, níveis entre 50 a 100 batimentos por minuto são considerados normais para adultos, podendo ser maiores em crianças [3, 4, 7].

### 2.2.1 Princípio de Funcionamento

Em resumo, o funcionamento do oxímetro de pulso é baseado em dois princípios:

1. A absorção de luz na hemoglobina oxigenada é diferente da absorção da hemoglobina não combinada com oxigênio.
2. Os dois sinais obtidos da leitura do oxímetro contém um componente AC, resultante da movimentação do sangue.

De forma geral, os oxímetros de pulso consistem em um par de diodos emissores de luz, conhecidos como LEDs, operando em diferentes comprimentos de onda; um LED vermelho com comprimento de onda igual a 660 nm e um LED infravermelho com comprimento de onda de 910 nm, além de um fotodiodo que faz a medição da absorção de luz emitida pelos LEDs [5]. Normalmente, o fotodiodo fica localizado de forma oposta ao LEDs, conforme demonstrado na Figura 2.3.

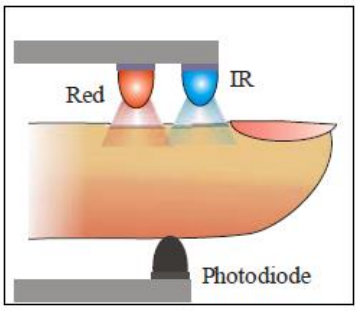



Figura 2.3. Princípio de Funcionamento do Oxímetro: dois LEDs e um fotodiodo.

A hemoglobina (Hb) presente no sangue se apresenta de duas formas: combinada com moléculas de oxigênio ($O_2$) chamada de oxiemoglobina ($HbO_2$) e não combinada com $O_2$ conhecida como desoxiemoglobina. As duas formas de Hb têm diferentes absorções da luz: a hemoglobina oxigenada absorve mais luz infravermelha e deixa passar mais luz vermelha em relação a hemoglobina desoxigenada, como demonstrado na Figura 2.4. A partir desse conceito, o oxímetro de pulso faz a detecção da diferença do nível de absorção de luz pelo fotodiodo nos dois diferentes comprimentos de onda e os utiliza para fazer o cálculo da porcentagem de saturação de oxigênio. Portanto, o nível de saturação total de oxigênio ($SpO_2$) pode ser definido como uma média da oxiemoglobina em relação a concentração total de hemoglobina no sangue, conforme fórmula (1):

$$\boldsymbol{Sp}O_2 = \frac{HbO_2}{HbO_2+Hb} \qquad (1)$$

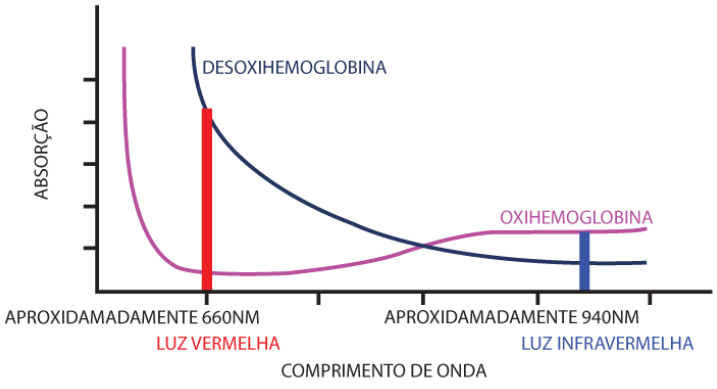



Figura 2.4. Absorção da desoxiemoglobina e oxiemoglobina em relação ao comprimento de onda da luz incidente

O sinal de leitura do fotodiodo apresenta duas componentes: uma componente alternada (AC) e uma componente contínua (DC). A componente DC é o resultado da

absorção de elementos que são constantes como o fluxo de sangue venoso, os tecidos e ossos enquanto que a componente AC é o resultado dos valores oscilatórios do sinal, conforme demonstrado na Figura 2.5. A partir da interpretação da parte AC do sinal, a qual apresenta picos e vales devido a pulsação do sangue, pode-se detectar a taxa de batimentos cardíacos por minuto, traçar a curva pletismográfica que permite verificar a amplitude e frequência do pulso além de realizar o cálculo da oxigenação arterial [3, 5, 9].

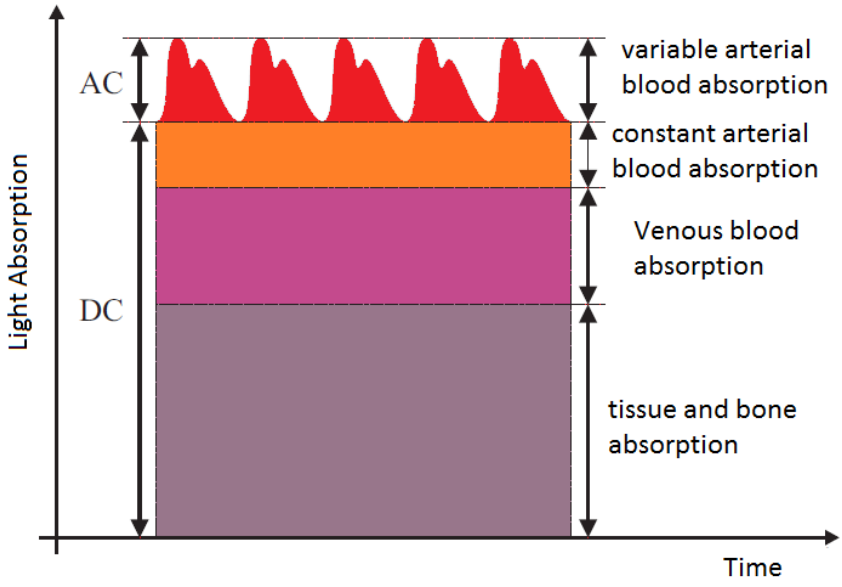



Figura 2.5. Absorção da Luz

Fonte: Abdallah O; Bolz A; Rajendram, Adaptive Filtering by Non-Invasive Vital Signals Monitoring and Diseases Diagnosis.

Com base na lei de *Beer-Lambert* que relaciona a absorção de luz com as propriedades e a concentração do material atravessado, é definida a equação da razão R (2) a partir dos valores AC e DC obtidos da absorção de luz pelos Leds vermelho e infravermelho.

$$R = \frac{\frac{AC_{vermeho}}{DC_{vermelho}}}{\frac{AC_{infra\ vermeho}}{DC_{infra\ vermelho}}} \quad (2)$$

Em que:

$AC_{vermeho}$ e $AC_{infra\ vermelho}$ são os componentes de tensão alternada obtidos a partir dos LEDs vermelho e infravermelho, respectivamente.

$DC_{vermeho}$ e $DC_{infra\ vermelho}$ são os componentes de tensão contínua obtidos a partir dos LEDs vermelho e infravermelho, respectivamente.

A partir do cálculo da razão R demonstrado na equação (2), é possível calcular a saturação de oxigênio $\%SpO_2$. O valor de R é relacionado ao valor mais aproximado de oxigenação sanguínea com base em uma tabela de valores que faz uma relação entre a razão R e o valor esperado de $\%SpO_2$. Esta tabela é determinada a partir do modelo de *Beer-Lambert,* porém este modelo não considera o múltiplo espalhamento da luz pelas células de sangue e, na prática não é linear. Portanto, muitos oxímetros fazem uso de uma tabela de calibração empírica definida com base em dados de oxigenação coletados através de algum método mais preciso, como por exemplo, o método invasivo através da coleta de sangue. A Figura 2.6 faz uma comparação entre a lei de *Beer-Lambert* e um exemplo de calibração empírica, onde podemos ver a diferença dos valores teóricos e os obtidos na prática.

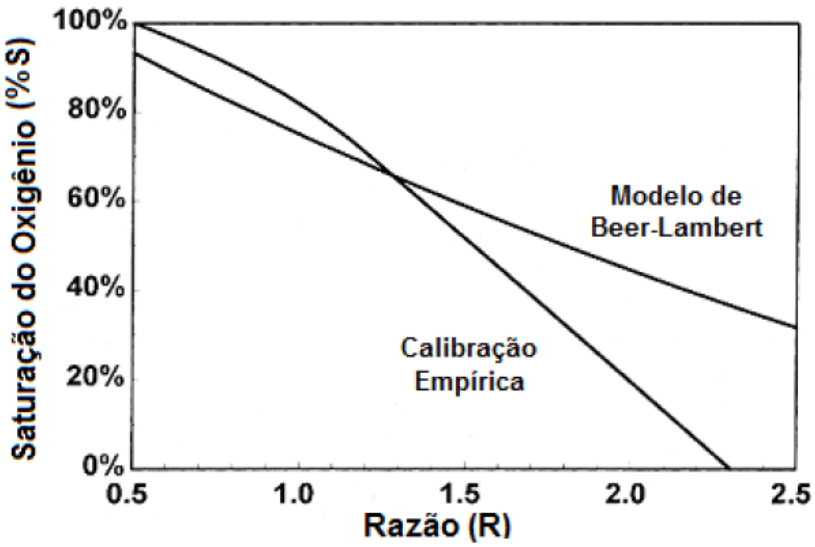



Figura 2.6. Calibração Empírica e Lei de Beer-Lambert.

Em resumo, para o cálculo da $\%SpO_2$ as seguintes etapas são observadas:

- Os valores DC dos sinais são obtidos a partir do cálculo da média dos valores lidos.

- Os valores AC dos sinais são calculados subtraindo-se a parte dc do sinal, e fazendo-se o cálculo do valor RMS ou valor eficaz do sinal, que é a raiz quadrática da média aritmética dos quadrados dos valores.
- Com os valores AC e DC de ambos os sinais, a razão R é calculada.
- Considerando uma relação linear entre os valores de $\%SpO_2$ e a razão R, a seguinte fórmula (3) é utilizada:

$$\mathbf{\%SpO_2} = 100 - R \qquad (3)$$

Para o cálculo da frequência cardíaca é considerado o sinal do LED infravermelho por apresentar valores um pouco maiores do que o sinal do LED vermelho. O sinal então é filtrado através de um filtro passa-baixa para a remoção de frequências altas e indesejadas já que a frequência do batimento cardíaco é baixa. Após o filtro, um algoritmo de detecção de pico é aplicado e a cada 3 picos do sinal, a taxa de batimento cardíaco é calculada com base na fórmula (4). Esta fórmula pode ser adaptada, considerando a detecção de mais picos, pois o valor de 3 picos foi escolhido como teste inicial.

$$\textbf{Taxa Bpm} = \frac{\text{Taxa de amostragem (amostras/s) x 60(segundos) x 3(picos)}}{\text{Número de amostras do intervalo entre 3 picos}} \qquad (4)$$

### 2.2.2 Implementação do Sistema

A implementação do sistema de um oxímetro pode ser descrita pelos seguintes itens:

a) Controle dos LEDs (um vermelho e um infravermelho). Os dois LEDs são ligados de forma alternada, quando o vermelho está ligado o infravermelho se desliga e assim vice versa. Dessa forma é necessária a utilização de um módulo para o controle de tempo do chaveamento dos LEDs. Além disso, o nível de luminosidade dos LEDs também precisa ser regulada através de uma fonte de corrente ou através de um sistema de controle de ganho.
b) Controle do fotodiodo utilizado para absorção da luz emitida dos LEDs. O fotodiodo produz uma corrente relacionada a intensidade da luz absorvida, a qual é convertida em tensão proporcional através de um amplificador de transimpedância e então ajustada por um amplificador.

c) Remoção de ruídos indesejados. O sinal de tensão convertido a partir do fotodiodo passa por um filtro passa baixa para a retirada de frequências desnecessárias.
d) Conversão Analógico-Digital. O sinal é transmitido para o módulo ADC (*Analog to Digital Converter*) que realiza a conversão entre o sinal analógico para o digital.
e) Interface de comunicação com o microcontrolador para interpretação e processamento dos sinais em *software* [5, 8, 9]

### 2.3.3 Limitações

Algumas condições podem afetar o funcionamento do oxímetro e causar leituras falsas tais como:

- Diminuição do volume de sangue devido à hipotermia, hipovolemia ou hipotensão causando dificuldade em obter um sinal satisfatório para a leitura.
- Interferências externas como luminosidades excessivas.
- Tremores podem dificultar a captação do sinal pelo sensor.
- Intoxicação por monóxido de carbono. O monóxido de carbono se liga à hemoglobina formando a carboxiemoglobina sendo mal interpretado pelo oxímetro que obtém leituras altas de saturação do oxigênio, enquanto que a leitura de saturação real é muito mais baixa [4, 9].

# 3. DESCRIÇÃO DO SISTEMA

Este trabalho prevê o desenvolvimento de um sistema capaz de realizar a leitura e interpretação de sinais de um oxímetro de pulso, fazer o processamento dos dados obtidos e a transmissão dos mesmos para um dispositivo móvel utilizando o Bluetooth como protocolo de comunicação. O sistema fará uso dos seguintes módulos:

- Sensor: conjunto dos LEDs, vermelho, infravermelho e fotodiodo, elementos responsáveis pela aplicação dos pulsos luminosos e pela captação da luz irradiada através dos tecidos;
- Circuito integrado AFE4490: geração e aquisição dos sinais de corrente no sensor, filtragem, conversão analógica-digital e a transmissão dos dados através do barramento SPI;
- Microcontrolador Arduino: gerenciamento de recepção e processamento dos dados (%Sp$O_2$) e envio para o módulo de Bluetooth.
- Módulo Bluetooth HC-06: transmissão dos dados via protocolo Bluetooth.
- Dispositivo Móvel - Smartphone: sistema com aplicativo para recepção das informações via Bluetooth, armazenamento, processamento e apresentação dos resultados de oximetria e batimentos cardíacos, e integração em rede com gerenciamento centralizado ou central de monitoramento remoto (home monitoring).

O sistema descrito pode ser visualizado na Figura 3.1, através de um diagrama representativo.

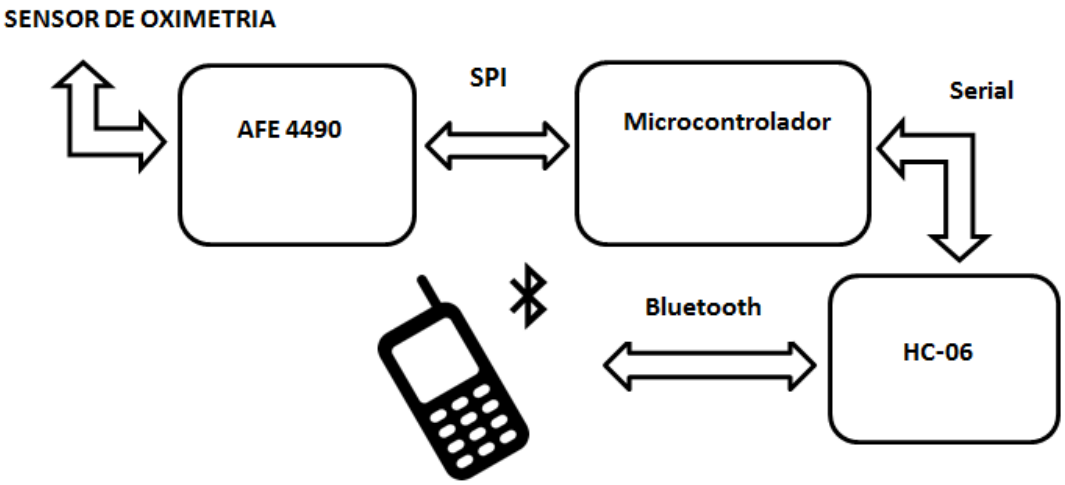



Figura 3.1. Diagrama do Sistema

Nas seções seguintes apresentam-se os módulos de hardware presentes no sistema, descrevendo as suas principais características e a respectiva função no sistema.

É importante ressaltar que, na proposta inicial do trabalho, a interface com o CI AFE4490 seria realizada a partir do projeto de uma placa de circuito impresso (PCI), de acordo com o proposto pelo *datasheet* do componente apresentado no Anexo A. Esta PCI prevê um conector DB-9 para conexão com o sensor de oximetria, além das conexões SPI para interface com um microcontrolador e a interface de comunicação com o módulo de Bluetooth, conforme descrito na Subseção 3.1.1.

Porém, devido as limitações da fresa utilizada e dos equipamentos de soldagem, optou-se pelo uso do próprio kit de desenvolvimento do CI AFE4490, apresentado na Subseção 3.1.2.

### 3.1.1 Placa de Circuito Impresso (PCI)

O projeto de uma placa de circuito impresso envolve três etapas: definição de um esquemático, o desenho do layout e a confecção da placa física. O esquemático consiste em um diagrama representativo de todos os componentes que serão utilizados e suas respectivas conexões. O layout é o desenho da placa com base no esquemático, considerando a espessura das trilhas, o tamanho e a disposição otimizada dos componentes além da definição de *layers* e vias. Por fim, a placa física é confeccionada com base no desenho do layout. O software utilizado nas etapas de esquemático e layout foi o Eagle.

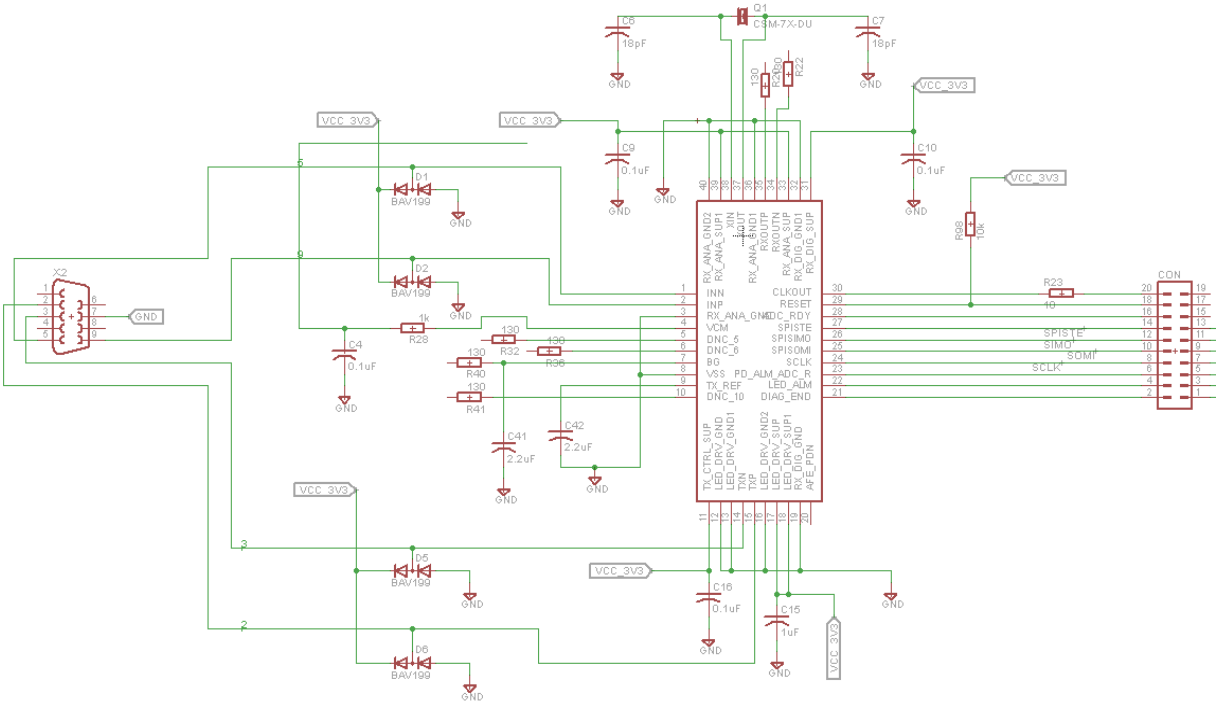



Figura 3.2. Esquemático Parte 1

O esquemático da placa de circuito foi definido com base no esquemático proposto no *datasheet* do CI AFE4490 (Anexo A). Conforme pode ser visto na Figura 3.2, o diagrama interliga o sensor de oximetria através de um conector DB-9 (à esquerda) e deixa os sinais SPI disponíveis em um conector do tipo header (à direita), tendo o CI AFE4490 realizando a interface de recepção e transmissão dos dados. Além disso, de acordo com a Figura 3.3, o projeto do esquemático também prevê a integração com um microcontrolador e já possui pinos disponíveis para a conexão com o módulo de Bluetooth, o qual possui resistores e capacitores de *pull-up*. Considerando que o microcontrolador e o componente HC-06 necessitam de 5V como tensão de alimentação e o chip somente suporta 3.3V, é necessário o uso de um regulador da tensão de entrada, que também foi anexado ao esquemático.

Na Figura 3.4 é apresentado o desenho do layout da placa, em que é possível perceber que a parte crítica do circuito é em torno do chip onde foi necessária a utilização de trilhas de 0.2 mm.

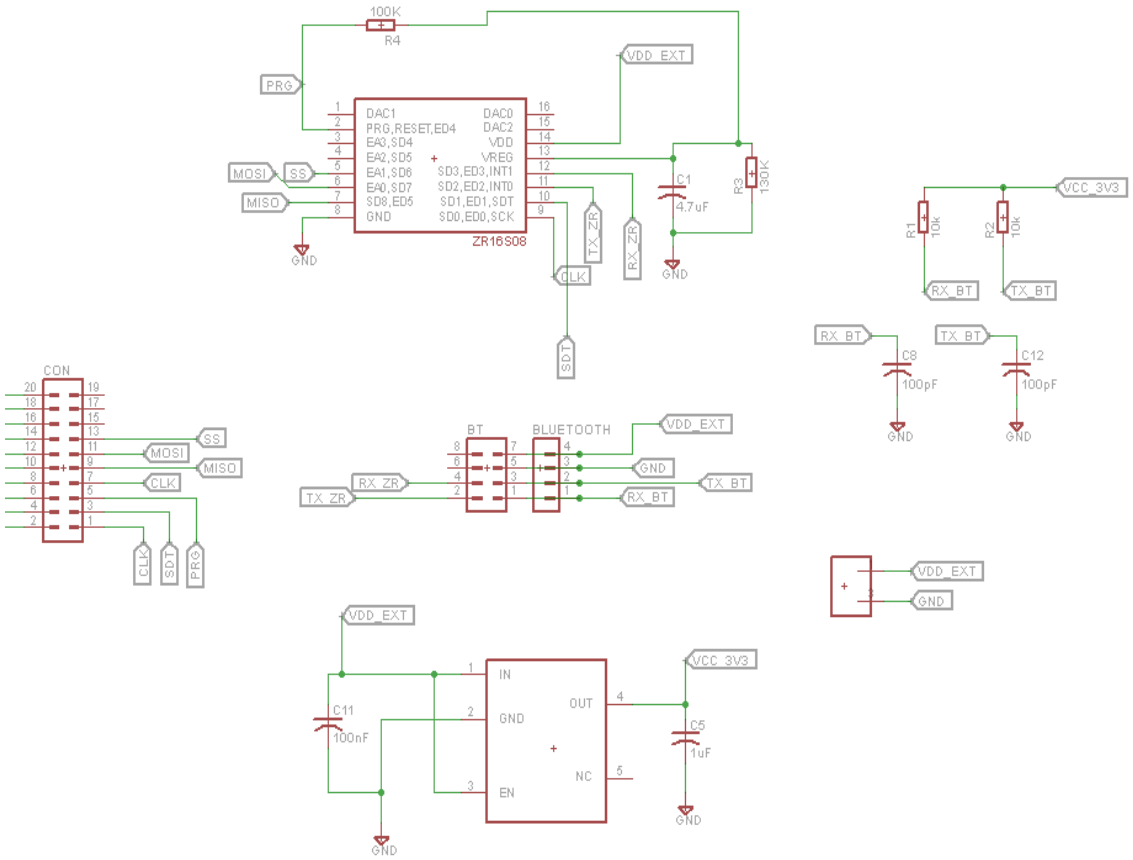



Figura 3.3 Esquemático Parte 2

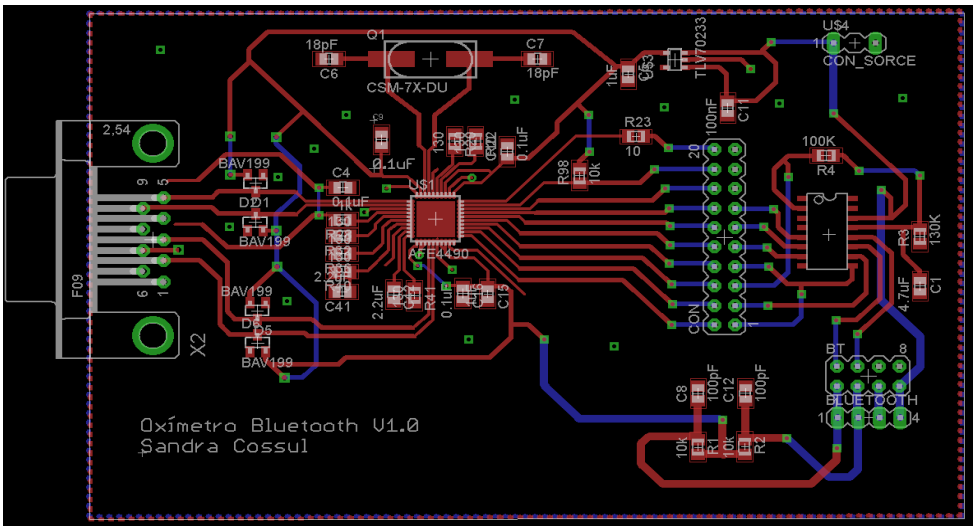



Figura 3.4 Layout

### 3.1.2 Kit de Desenvolvimento: Sensor de Oximetria + AFE4490

Este trabalho utiliza o kit de desenvolvimento da Texas Instruments conhecido como *AFE4490 Evaluation Module,* demonstrado na Figura 3.5, que tem como objetivo facilitar o uso e o acesso ao dispositivo AFE4490. Este kit contém um sensor

de oximetria com conexão DB-9, conforme Figura 3.6, além de um software que permite o acesso a todos os registradores do CI AFE4490 e a coleta e armazenamento de dados de leitura. É importante notar que esta placa de desenvolvimento possui pontos de testes que permitem a medição da tensão e assim a averiguação da funcionalidade do chip. Além disso, muitas das trilhas presentes na placa possuem resistores que atuam como jumpers permitindo a observação do sinal ou até mesmo a aquisição do sinal através da retirada dos mesmos.

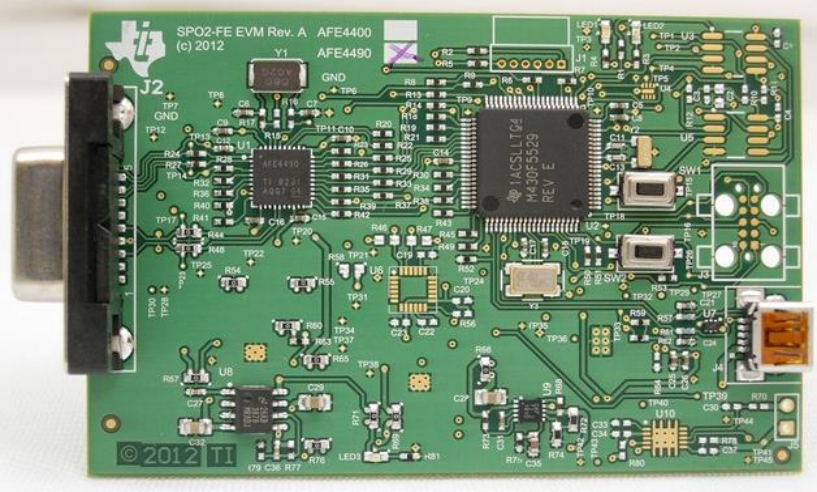



Figura 3.5. AFE4490 - *Evaluation Module*

Figura 3.6. Ponteira dedal - Sensor de Oximetria

O sensor de oximetria é posicionado na ponta do dedo ou no lóbulo da orelha do paciente, onde realiza a aquisição e envio dos sinais de leitura. Este processo envolve a emissão de luz de dois LEDs (vermelho e infravermelho) e a captação da absorção de luz dos dois diferentes comprimentos de onda pelo fotodiodo.

O chip de circuito integrado AFE4490 é conhecido como um *analog front-end (AFE)*, ou seja, um bloco de componentes necessários para realizar a interface de um sensor para um microcontrolador, que nesse caso prevê aplicações relacionadas a oximetria. De forma geral, este dispositivo contém um canal de recebimento de dados,

um conversor analógico digital de 22 bits, um canal de envio de dados de controle para os LEDs e uma seção de detecção de falhas, além de uma interface de comunicação SPI.

A partir da Figura 3.7, que representa o diagrama de blocos do chip AFE4490 que será utilizado neste trabalho, podemos ter uma visão melhor e mais detalhada dos módulos presentes e suas respectivas conexões.

Na seção de recebimento de dados (Rx), os sinais de corrente provenientes do fotodiodo são convertido para sinais de tensão apropriados no amplificador de transimpedância (TIA). Estes sinais passam por um segundo estágio onde são amplificados e então filtrados para a remoção de frequências que não são necessárias. Após esse processo, são transmitidos para o conversor analógico digital que disponibiliza na interface de saída SPI os seguintes sinais: LED1, LED2, luz ambiente LED1, luz ambiente LED2, (LED1 – luz ambiente LED1) e (LED2 – luz ambiente LED2).

O módulo *Timing Controller* realiza o controle e sincronização dos sinais em relação ao tempo considerando o chaveamentos dos LEDs, a aquisição dos sinais e a conversão analógica-digital. O sinal de clock de 4 MHZ é gerado por um oscilador interno a partir de um cristal. Adicionalmente, o módulo LED Current Control + LED Driver realiza o controle da corrente fornecida aos LEDs utilizando valores pré-definidos como referência [10].

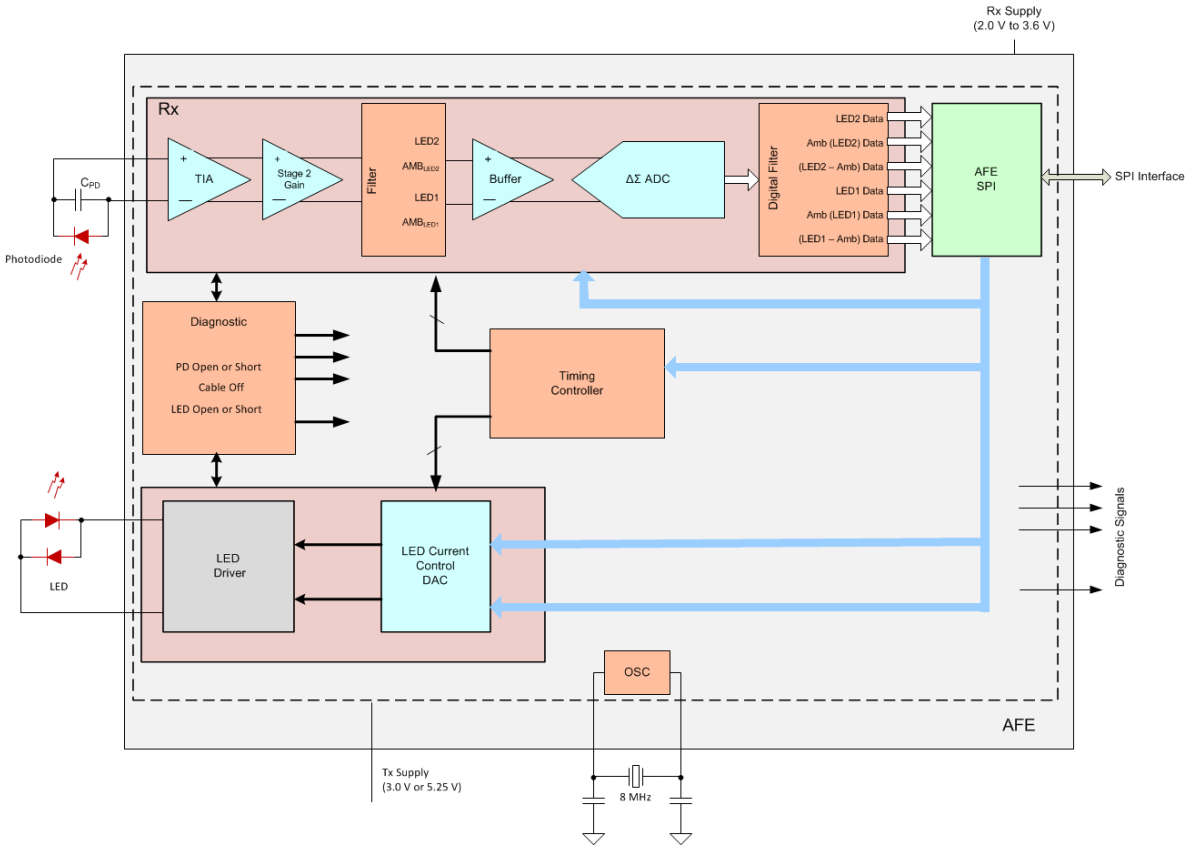



Figura 3.7. Diagrama de Blocos AFE4490

### 3.1.3 Microcontrolador Arduino

O microcontrolador tem como função iniciar a comunicação com o módulo AFE4490 através da interface SPI. Este processo inclui a inicialização dos registradores, a configuração do protocolo de comunicação, a aquisição dos dados além da interpretação e envio dos mesmos para o módulo de Bluetooth. Neste trabalho será utilizado o Arduino modelo Uno, Figura 3.5, por se tratar de um microcontrolador de fácil manuseio, com uma linguagem de programação padrão (C/C++) e bibliotecas testadas, como por exemplo, a biblioteca SPI.

A placa de desenvolvimento Arduino Uno consiste do microcontrolador Atmega328 de 8 bits, 14 pinos de entrada e saída de dados, 6 pinos de entrada analógicos, um cristal de 16MHz, uma conexão USB, um conector para alimentação, conectores ICSP para comunicação SPI ou carregamento de *bootloader* e um botão de reset. O Arduino tem uma tensão de operação de 5V, memória flash de 32KB, memória SRAM de 2KB e memória EEPROM de 1 KB.

A alimentação do Arduino Uno pode ser através da conexão USB ou bateria, desde que respeitados os níveis de tensão de entrada entre 6V e 20V. Dois pinos de saída são deixados disponíveis: um de 5 V e outro de 3.3V, além de pinos de GND, o que pode facilitar a conexão com outros módulos e shields.

O Arduino pode ser programado através do software que é disponibilizado, o qual permite o upload de um novo código sem a utilização de um hardware extra. A programação em C/C++ ocorre de forma simples, sendo necessária a definição de duas funções para um software: setup () que realiza as inicializações e configurações de variáveis, pinos e bibliotecas; e loop () que repete um bloco de comandos [11].

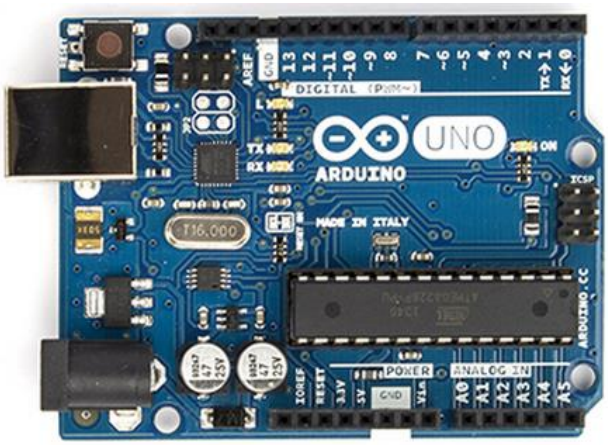



Figura 3.8. Arduino Uno

### 3.1.4 Módulo de Bluetooth HC-06

O componente HC-06 é composto por uma interface de comunicação serial e um adaptador de Bluetooth, podendo, por exemplo, se conectar à porta serial UART de um microcontrolador e fazer a transmissão de dados via Bluetooth.

Este módulo possui Bluetooth na versão 2.0, possui um alcance de até 10 metros, antena embutida na placa, taxa de transmissão de 9600 bps na configuração padrão e sua tensão de alimentação é de 3.3 V. Conforme apresentado na Figura 3.6, o módulo HC-06 possui 6 pinos sendo necessário 4 pinos para sua configuração mínima:

- UART_TXD : UART Data Output
- UART_ RXD : UART Data Input

- GND : Ground
- VCC : Alimentação

Figura 3.9. Módulo Bluetooth HC06.

## 3.2 Protocolos de Comunicação

A seguinte seção descreve o princípio de funcionamento dos dois principais protocolos de comunicação utilizados para a implementação do sistema: Serial Peripheral Interface (SPI) e Bluetooth.

### 3.2.1 SPI (Serial Peripheral Interface)

SPI ou Serial Peripheral Interface é um protocolo de comunicação serial síncrono utilizado para realizar a troca de informações entre microcontroladores e um ou mais dispositivos periféricos ou entre dois microcontroladores. O protocolo faz uso de uma interface Mestre/Escravo (*Master/Slave*), em que, na maioria dos casos, o microcontrolador age como o Mestre e seus periféricos como Escravos. O barramento de comunicação entre os dispositivos é full-duplex, ou seja, o fluxo de dados ocorre em ambas as direções podendo atingir a velocidade máxima de 10Mbps.

O protocolo SPI utiliza 4 linhas de comunicação, entre elas uma para o clock, duas para transmissão de dados e uma para selecionar o dispositivo para troca de informações, conforme Figura 3.7:

- SCLK (Serial Clock) - gerado pelo mestre para sincronizar a comunicação
- MISO (Master In Slave Out) - dados do escravo para o mestre
- MOSI (Master Out Slave In) – dados do mestre para o escravo
- SS (Slave Select) – seleção do escravo/periférico pelo mestre

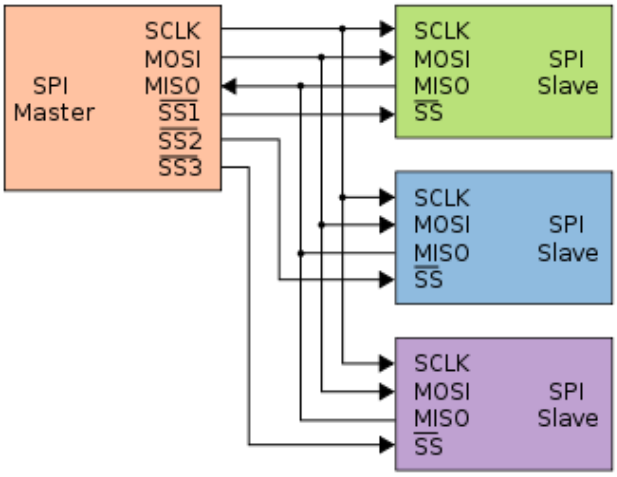



Figura 3.10. Representação Protocolo SPI

A comunicação é iniciada pelo microcontrolador/mestre, que faz a configuração do clock e então, seleciona o periférico/escravo que deseja se comunicar através da linha SS (Slave Select) setando-a com o nível lógico ‘0’. A cada ciclo de clock, o mestre envia dados através da linha MOSI, o escravo lê e ao mesmo tempo envia dados através da linha MISO enquanto que o mestre faz a leitura simultânea ao envio de novos dados.

O protocolo de comunicação SPI é dividido em quatro modos de operação diferenciando-se pela polaridade (CPOL- *clock polarity*) e fase do clock (CPHA - *clock phase*), que define se os dados são lidos e transmitidos na borda de subida ou descida, conforme apresentado na Tabela 1. Outra característica que deve ser observada na comunicação é a posição do bit mais significativo, que pode ser transmitido primeiro ou por último, influenciando diretamente a leitura e escrita do sinal.

Tabela 1 - Modos de Operação SPI

| Modo SPI | CPOL | CPHA | Borda de Troca | Transição |
|---|---|---|---|---|
| 0 | 0 | 0 | Subida | Meio do bit |
| 1 | 0 | 1 | Descida | Começo do bit |
| 2 | 1 | 0 | Descida | Meio do bit |
| 3 | 1 | 1 | Subida | Começo do bit |

O protocolo SPI apresenta algumas vantagens por ser uma comunicação *full-duplex*, possuir uma interface de *hardware* simples além de permitir um alto fluxo de dados e ser flexível em relação ao tamanho e conteúdo das mensagens. Pode-se citar como desvantagens a falta de verificação de erros durante a transmissão dos dados, a necessidade de ser utilizado em curta distância além da configuração somente permitir um único dispositivo mestre [12,13].

### 3.2.2 Bluetooth

Bluetooth é um padrão de comunicação sem fio de curto alcance que permite que os dispositivos tais como smartphones, impressoras, mouses, fones de ouvido se conectem de forma rápida bastando que um esteja próximo do outro. O Bluetooth é caracterizado como um tipo de rede sem fio pessoal conhecida como WPAN *(Wireless personal area networks*), em que a transmissão de dados ocorre utilizando radiofrequência, o que limita o seu alcance, porém tem um baixo consumo de energia.

O alcance de transmissão da tecnologia Bluetooth pode atingir distâncias de até 100 metros na classe 3, sendo possível configurar o alcance para distâncias menores como até 10 metros na classe 2 e até 1 metro na classe 1. A taxa de transmissão é de 3MB/s na versão 2.0 e pode atingir taxas de até 24 MB/s na versão 3.0.

O protocolo Bluetooth utiliza uma faixa de rádio entre 2.4GHz e 2.485 GHz e emprega um esquema chamado frequency hopping, em que o dispositivo que inicia a transmissão muda constantemente de frequência diminuindo as chances de interferência e enfraquecimento do sinal. O processo de comunicação entre dois dispositivos é full-duplex (os dois podem enviar dados) em que o dispositivo que inicia a comunicação é o mestre enquanto que os demais são os escravos, sendo que o mestre é responsável pelo sincronismo e o regulamento da transmissão dos dados.

A conexão entre os dispositivos ocorre através de requisições, em que um dispositivo envia uma solicitação de conexão para um dos dispositivos visíveis e então, fica aguardando a autorização ou o pareamento. Pareamento é o processo necessário para estabelecer uma conexão segura através de uma senha que é gerada por um dos dispositivos e confirmada no outro. Após esse processo de identificação e configuração, a conexão Bluetooth está estabelecida.

A tecnologia Bluetooth apresenta muitas vantagens que facilitaram a sua ampla utilização tais como a sua especificação aberta que permite que todas as fabricantes utilizam mantendo-se um padrão, interface simples de comunicação sem fio que substitui a necessidade de utilizar cabos e fios além da capacidade de transmissão simultânea de dados como voz e dados, facilitando a criação de diferentes aplicações [14,15].

# 4. DESENVOLVIMENTO

Esta seção prevê a implementação do sistema descrito na seção anterior, a partir da implementação do modelo de hardware que define as conexões entre os principais módulos (Sensor de Oximetria, Kit de Desenvolvimento AFE4490, Microcontrolador Arduino e Módulo HC-06), além da descrição do software utilizado no microcontrolador Arduino e no aplicativo Android. Também, é apresentado de forma geral o código utilizado no Matlab para visualização dos sinais.

## 4.1 Projeto Hardware

Para a realização das funções previstas, o circuito integrado AFE4490 deve ser interligado com alguns componentes. Dentre os principais componentes estão os LEDs e o fotodiodo que são conectados ao circuito utilizando o sensor de oximetria através do conector DB-9, um cristal para a geração do clock além de diodos, capacitores e resistores, já presentes na placa de desenvolvimento. A interface de comunicação com o CI AFE4490 é formada pelos sinais de diagnóstico e os sinais da interface SPI, os quais estão interligados com o microcontrolador MSP430F5529 que podem ser acessados através de resistores. Na Tabela 2 são apresentados todos os sinais que estão disponíveis para conexão com um microcontrolador externo e seus respectivos resistores.

Tabela 2 - Sinais Interface AFE4490

| Sinal | Resistor/Jumper |
|---|---|
| STE | R29 |
| SIMO | R31 |
| SOMI | R33 |
| SCLK | R35 |
| ADC_RDY | R26 |
| PD_ALM | R37 |
| LED_ALM | R39 |
| DIAG_END | R38 |
| AFE_PDNZ | R42 |
| AFE_RESETZ | R25 |

Para a comunicação com o CI AFE4490, pode-se somente utilizar os pinos da interface SPI, observando as regras do protocolo SPI. Tem-se a possibilidade de utilização do sinal de interrupção ADC_RDY que indica na borda de subida que o conversor analógico-digital finalizou a conversão e os dados estão disponíveis nos registradores para leitura. Os sinais de diagnóstico incluem o sinal PD_ALM que indica uma falha no sensor de oximetria, o sinal LED_ALM que indica falhas relacionadas a transmissão do sinal e problemas relacionados aos LEDs e também o sinal DIAG_END que aponta que a seção de diagnósticos foi completada. A conexão dos sinais de diagnóstico ao microcontrolador é opcional. Os sinais AFE_PDNZ e AFE_RESETZ são sinais de reset ativos em nível lógico baixo, portanto, devem ser mantidos em nível lógico alto.

O projeto prevê a conexão do CI AFE4490 com outro microcontrolador (Arduino) com o objetivo de uma aplicação própria. Para isso, algumas modificações na placa de desenvolvimento foram necessárias, como a retirada dos quatro resistores referentes a interface SPI (R29, R31, R33 e R35) e a solda de fios no ponto mais à esquerda que então são conectados aos pinos do Arduino. Para esta conexão, é necessário respeitar os níveis de tensão, pois o Arduino possui uma tensão de operação de 5 V e o AFE4490 opera em 3.3 V. Portanto, é necessário a utilização de um divisor resistivo de 5V para 3.3V somente nas conexões do Arduino em direção ao circuito integrado, ou seja, nos sinais SCLK, STE e SIMO.

No Arduino, a interface de conexão do SPI segue um padrão em que os sinais são ligados nos seguintes pinos digitais: MOSI- pino 11, MISO- pino 12, SCLK- pino 13, enquanto que a conexão do sinal STE é indiferente, sendo escolhido o pino digital 10. Ambos os módulos são alimentados através da conexão USB. Para aplicações em que o CI AFE4490 é utilizado sem a placa de desenvolvimento, os sinais AFE_PDN e AFE_RESET devem ser conectados obrigatoriamente ao microcontrolador e mantidos em nível lógico alto que, neste projeto ficou mantido a conexão ao microcontrolador MSP430. Outro ponto importante é que o sinal de ground (GND) da placa de desenvolvimento e do Arduino também deve ser conectado entre si para a manutenção da mesma referência.

Para a verificação da funcionalidade do CI AFE4490 a alimentação deve ser testada e as seguintes tensões devem ser obtidas tendo como referência o ground:

0.75 V no pino 9 (TX_REF), 1 V no pino 7 (BG), 0.9 V no pino 4 (VCM) e 4MHz no pino 30 (CLKOUT).

Para a conexão do microcontrolador com o módulo de Bluetooth HC-06, quatro sinais são considerados em que o sinal de ground é conectado ao GND comum a todos os módulos e o sinal VCC é conectado ao pino de saída do Arduino que fornece uma tensão de 5V. Os sinais para troca de dados TX e RX do HC-06 são conectados respectivamente aos pinos digitais 6 e 7 do Arduino, sendo necessária a utilização de um divisor resistivo de 5 V para 3.3 V para o sinal RX do módulo de Bluetooth, o qual recebe dados do Arduino.

Outra possibilidade para o projeto seria a utilização do microcontrolador MSP430F5529 já presente na placa de desenvolvimento. Porém, devido à falta de familiaridade com este microcontrolador e a possibilidade de adquirir componentes adicionais para o acesso (programação) e para uma interface sem fio, optou se pelo uso do Arduino por ser uma plataforma mais conhecida.

Em resumo, o projeto de hardware inclui a conexão do sensor de oximetria ao conector DB-9 da placa de desenvolvimento, a conexão da interface SPI e do módulo Bluetooth aos respectivos pinos do microcontrolador Arduino, respeitando as diferentes tensões de entrada e saída. Também, deve ser realizada a conexão em comum do ground.

## 4.2 Projeto do Software

### 4.2.1 Software Microcontrolador/Arduino

Para realizar a interface de comunicação SPI, é necessário a utilização de um software que fará a correta inicialização dos parâmetros do chip e a transmissão dos dados conforme os requisitos. De acordo com o *datasheet* do CI AFE4490, o chip contém registradores internos que podem ser acessados através do barramento SPI utilizando os pinos SPISTE, SCLK, SPISOMI e SPISIMO. O pino SCLK é o clock da interface SPI, responsável pelo sincronismo durante a entrada de comandos e saída de dados do dispositivo, controlado pelo pino SPISTE que habilita e desabilita o fluxo de dados. O pino SPISOMI realiza a troca de dados do chip em direção ao

microcontrolador, sendo que o pino SPISIMO realiza o tráfego contrário, do microcontrolador para o chip.

O Arduino possui uma biblioteca padrão utilizada para o protocolo SPI, a qual deve ser incluída como <SPI.h>. Para a utilização dessa biblioteca os pinos da interface SPI são pré-definidos: SCLK – pino 13, MISO – pino 12, MOSI – pino 11, STE – pino 10. As principais funções da biblioteca SPI são:

*SPI.begin()* – Inicializa o protocolo SPI, ajustando os pinos SCLK, MISO e MOSI. O pino STE deve ser configurado separadamente.

*SPI.end()* – Desabilita o protocolo SPI.

*SPI.setBitOrder()* – Seta a ordem de transmissão dos bits, podendo ser MSBFIRST (*Most Significant Bit First*) ou LSBFIRST (*Least Significant Bit First*).

*SPI.setDataMode()* – Seta o modo SPI que será utilizado.

*SPI.setClockDivider()* - Seta o clock do SPI relativo ao clock do sistema.

*SPI.transfer()* – Transfere um byte através do barramento SPI podendo ser uma operação de leitura ou escrita.

Para uma operação de escrita no dispositivo, o “registrador” SPI_READ deve ser setado para ‘0’ e o pino SPISTE ser mantido em nível lógico baixo. Os dados são transferidos em uma palavra de 24 bits de 8 em 8 bits, em que os 8 primeiros bits formam o endereço do registrador a ser escrito e os 24 bits restantes os dados. Para uma operação de leitura, o “registrador” SPI_READ deve ser ‘1’ e o pino SPISTE também deve estar em nível lógico baixo. O processo de leitura ocorre através de um comando de escrita que define o endereço do registrador a ser lido utilizando 8 bits, e então, os dados são transferidos em 3 sequências de 8 bits totalizando 24 bits. Tanto para a operação de leitura quanto para a de escrita os bits mais significativos são enviados primeiro, sempre na borda de subida do clock, portanto a ordem de transmissão é MSBFIRST através do modo 0 ou SPI_MODE0.

A inicialização do circuito integrado AFE4490 acontece a partir da configuração dos registradores setando o instante inicial e final de amostragem dos sinais, o instante inicial e final de conversão analógico digital dos sinais além da configuração do próprio ADC (Analog Digital Converter) e dos sinais de controle com base na frequencia ou PRF (Pulse Repetition Frequency). Neste trabalho, a configuração do dispositivo AFE4490 considera PRF = 500 Hz e *duty cycle* = 25%, em que LED1 =

Led Infravermelho e LED2 = Led Vermelho, com base no software disponibilizado com a placa de desenvolvimento.

Os registradores foram inicializados com os valores (numeração hexadecimal) apresentados na Tabela 3 através de uma operação de escrita utilizando o barramento SPI, seguindo uma ordem de amostragem: LED2Ambient, LED1, LED1Ambient e LED2 e uma ordem de conversão: LED2, LED2Ambient, LED1 e LED1Ambient. A Tabela 3 apresenta também o endereço dos registradores além de uma descrição sucinta de cada, com base no datasheet do dispositivo.

Tabela 3 - Configuração dos Registradores AFE4490

| Registrador | Endereço | Valor Inicial | Descrição |
|---|---|---|---|
| CONTROL0 | 0x00 | 0x00 | Habilita e Desabilita SPI_READ, Software Reset e Diagnósticos |
| LED2STC | 0x01 | 0x17C0 | Início amostra LED2 |
| LED2ENDC | 0x02 | 0x1F3E | Final amostra LED2 |
| LED2LEDSTC | 0x03 | 0x1770 | Início sinal LED2 |
| LED2LEDENDC | 0x04 | 0x1F3F | Final sinal LED2 |
| ALED2STC | 0x05 | 0x0050 | Início amostra Ambient LED2 |
| ALED2ENDC | 0x06 | 0x07CE | Final amostra Ambient LED2 |
| LED1STC | 0x07 | 0x0820 | Início amostra LED1 |
| LED1ENDC | 0x08 | 0x0F9E | Final amostra LED1 |
| LED1LEDSTC | 0x09 | 0x07D0 | Início sinal LED1 |
| LED1LEDENDC | 0x0A | 0x0F9F | Final sinal LED1 |
| ALED1STC | 0x0B | 0x0FF0 | Início amostra Ambient LED1 |
| ALED1ENDC | 0x0C | 0x176E | Final amostra Ambient LED1 |
| LED2CONVST | 0x0D | 0x0006 | Início conversão LED2 |
| LED2CONVEND | 0x0E | 0x07CF | Final conversão LED2 |
| ALED2CONVST | 0x0F | 0x07D6 | Início conversão Ambient LED2 |
| ALED2CONVEND | 0x10 | 0x0F9F | Final conversão Ambient LED2 |
| LED1CONVST | 0x11 | 0x0FA6 | Início conversão LED1 |

| | | | |
|---|---|---|---|
| LED1CONVEND | 0x12 | 0x176F | Final conversão LED1 |
| ALED1CONVST | 0x13 | 0x1776 | Início conversão Ambient LED1 |
| ALED1CONVEND | 0x14 | 0x1F3F | Final conversão Ambient LED1 |
| ADCRSTSTCT0 | 0x15 | 0x0000 | Início ADC Reset 0 |
| ADCRSTENDCT0 | 0x16 | 0x0005 | Final ADC Reset 0 |
| ADCRSTSTCT1 | 0x17 | 0x07D0 | Início ADC Reset 1 |
| ADCRSTENDCT1 | 0x18 | 0x07D5 | Final ADC Reset 1 |
| ADCRSTSTCT2 | 0x19 | 0x0FA0 | Início ADC Reset 2 |
| ADCRSTENDCT2 | 0x1A | 0x0FA5 | Final ADC Reset 2 |
| ADCRSTSTCT3 | 0x1B | 0x1770 | Início ADC Reset 3 |
| ADCRSTENDCT3 | 0x1C | 0x1775 | Final ADC Reset 3 |
| PRPCOUNT | 0x1D | 0x1F3F | Pulse Repetition Period Count |
| CONTROL1 | 0x1E | 0x0101 | Número de valores para média, Habilita/Desabilita Timer, Configura clock em pinos de alarme |
| SPARE1 | 0x1F | 0x0000 | Deve ser setado em ‘0’ |
| TIAGAIN | 0x20 | 0x0000 | Configura LED1 |
| TIA_AMB_GAIN | 0x21 | 0x0000 | Configura LED2 |
| LEDCNTRL | 0x22 | 0x11414 | Seta a corrente do LED1 e LED2 |
| CONTROL2 | 0x23 | 0x0000 | Habilita/Desabilita AFE, Transmissão ou Recepção, Diagnósticos e Cristal. |
| SPARE2 | 0x24 | 0x0000 | Deve ser setado em ‘0’ |
| SPARE3 | 0x25 | 0x0000 | Deve ser setado em ‘0’ |
| SPARE4 | 0x26 | 0x0000 | Deve ser setado em ‘0’ |
| RESERVED1 | 0x27 | 0x0000 | Reservado para fabricante |
| RESERVED2 | 0x28 | 0x0000 | Reservado para fabricante |
| ALARM | 0x29 | 0x0000 | Controla funcionalidade do alarme |

Após a inicialização dos registradores com os valores apresentados na Tabela 3, o dispositivo está pronto para a obtenção de dados. Os dados que podem ser lidos

estão nos registradores com seus respectivos endereços apresentados na Tabela 4. Os registradores contém os últimos valores obtidos pelo conversor analógico-digital dos sinais referentes ao LED1, LED2, LED1Ambient, LED2Ambient e esses valores subtraídos entre si. Além disso, é possível fazer a leitura do sinal de diagnóstico que informa falhas no sensor de oximetria, nos LEDs e no fotodiodo.

Tabela 4 - Registradores de Leitura AFE4490

| **Registrador** | **Endereço** |
|---|---|
| LED2VAL | 0x2A |
| ALED2VAL | 0x2B |
| LED1VAL | 0x2C |
| ALED1VAL | 0x2D |
| LED2-ALED2VAL | 0x2E |
| LED1-ALED1VAL | 0x2F |
| DIAG | 0x30 |

O conversor analógico-digital gera uma representação digital a partir de uma grandeza analógica, portanto os registradores de leitura assumem valores discretos com base na precisão do conversor que é de 21 bits. Para a obtenção dos valores em volts é necessário a utilização da seguinte fórmula:

**Valor ADC (volts) = (Valor lido ADC * Valor de referência) / $2^{21}$**

Valor referência = 1.2 V

Obtidos os valores de ambos os LEDs e realizada a conversão dos valores em Volts, é possível realizar o cálculo da porcentagem de oxigenação sanguínea. O cálculo de %SPO2 considera um intervalo de leitura de valores dos LEDs vermelho e infravermelho. Este intervalo é definido de forma que um conjunto de valores médios seja utilizado para os cálculos, e neste trabalho, optou-se por um conjunto de 60 valores. O valor DC dos sinais é definido a partir do valor médio dos sinais obtidos dentro do intervalo e o valor AC dos sinais é calculado a partir da subtração do valor médio e então da extração da raiz quadrada dos quadrados dos valores, o que corresponde ao cálculo do valor rms (*root mean square).* Após o cálculo dos valores

AC e DC, estes são aplicados a fórmula (2) para o cálculo da razão R. A partir do resultado obtido, é possível obter o valor final de %SPO2 a partir da fórmula (3).

Este valor é transferido para o aplicativo Android, através do módulo de Bluetooth HC-06 conectado ao Arduino. Para isso, é utilizada a biblioteca padrão <SoftwareSerial.h>, a qual realiza o suporte de comunicações seriais. Para a utilização desta biblioteca, é necessário a definição de dois pinos para transmissão e recepção (TX e RX) além das seguintes funções:

*Serial.begin()* – Inicializa a comunicação serial setando a velocidade (baud rate).
*Serial.println()* – Envia dados para o pino referente a transmissão da porta serial.

### 4.2.2 Software Android

A implementação do sistema prevê o desenvolvimento de um aplicativo Android, o qual será responsável pela recepção dos dados através de Bluetooth e através de uma interface gráfica a apresentação da porcentagem de saturação de oxigênio (%SpO2).

O Android possui uma API para Bluetooth que realiza quatro tarefas indispensáveis para o processo de comunicação utilizando Bluetooth:

1. Configuração do Bluetooth: verificação e ativação da funcionalidade.
2. Procura de dispositivos com o Bluetooth ativado: procedimento para escanear dispositivos na lista de emparelhados ou que estão disponíveis na área de alcance.
3. Estabelecimento da conexão: a partir do mútuo reconhecimento dos dispositivos a conexão é estabelecida em uma interface servidor/cliente.
4. Transferência de dados entre dois dispositivos: após a conexão ter sido realizada, ocorre o compartilhamento de dados [16].

O aplicativo para Android, apresentado na Figura 4.1, possui os seguintes botões: “Turn On” para ativar o Bluetooth e permitir a comunicação, “Connect” para iniciar a conexão com o módulo HC-06, “Send” para enviar dados via Bluetooth utilizado em testes e “Clear” também para testes. O valor de %SpO2 é mostrado ao lado do seu indicativo.


Figura 4.1. Interface App Android

### 4.2.3 Software Matlab

O Matlab é uma importante ferramenta computacional utilizada para interpretação e visualização de sinais além de permitir a manipulação dos mesmos através da utilização de variadas funções matemáticos. Neste trabalho, o Matlab é utilizado para a visualização de forma gráfica dos sinais obtidos dos LEDs vermelho e infravermelho, tendo em vista testar o algoritmo para o cálculo de batimentos cardíacos por minuto (BPM).

Para o cálculo de BPM, é considerada uma amostra de um certo intervalo do LED infravermelho. Este sinal possui frequências indesejadas causadas por movimentos e ruídos e, portanto é aplicado um filtro passa-baixa com frequencia de corte de 10Hz. O valor de corte é definido a partir da consideração que os valores extremos de batimento cardíaco podem atingir até 200 bpm, que seria em torno de 3.3 Hz, portanto valores muito acima desta frequência são indesejados. Para a implementação do filtro é utilizada a função *[b a ] = butter(n, Wn)* que retorna os coeficientes de um filtro passa-baixas de ordem n com frequencia de corte Wn e a função *filter (b ,a ,x)* que filtra os dados presentes no vetor x com base nos vetores de coeficientes *a* e *b*.

Após a filtragem do sinal, um algoritmo de detecção de pico é aplicado ao sinal. Este algoritmo é baseado no algoritmo de Eli Billauer, que encontra os mínimos e máximos locais de um sinal com ruído, através da definição de um valor de *threshold* (delta) entre um pico ou vale e seus valores anteriores e posteriores. Este algoritmo detecta picos máximos e mínimos em um vetor, onde um ponto é considerado um pico

máximo se ele tem um valor máximo e foi precedido por um valor menor [17]. O algoritmo simplificado está demonstrado abaixo:

```
function [maxtab, mintab]=peakdet(v, delta, x)
maxtab = [];
mintab = [];

v = v(:);
mn = Inf; mx = -Inf;
mnpos = NaN; mxpos = NaN;

lookformax = 1;
for i=1:length(v)
  this = v(i);
  if this > mx, mx = this; mxpos = x(i); end
  if this < mn, mn = this; mnpos = x(i); end

  if lookformax
    if this < mx-delta
      maxtab = [maxtab ; mxpos mx];
      mn = this; mnpos = x(i);
      lookformax = 0;
    end
  else
    if this > mn+delta
      mintab = [mintab ; mnpos mn];
      mx = this; mxpos = x(i);
      lookformax = 1;
    end
  end
end
```

A partir da obtenção dos picos do sinal e seus respectivos valores de amostra é realizado o cálculo da taxa de BPM, observando a fórmula (4).

É importante observar que o algoritmo no microcontrolador responsável pela leitura dos sinais não deve conter atrasos, pois isso afeta a taxa de transmissão e consequentemente o número de amostras obtidas. Portanto, o algoritmo para o cálculo de batimentos cardíacos deve compensar os atrasos de transmissão e ajustar a sua taxa de amostragem e valor de *threshold* de forma a obter resultados corretos de batimentos cardíacos.

# 5. RESULTADOS

## 5.1 Apresentação Gráfica dos Sinais

Com o objetivo de visualização dos dados obtidos referente aos LEDs vermelho e infravermelho é utilizado o software Matlab. Este teste inicial comprova o correto funcionamento do circuito e os resultados podem ser observados nas Figuras 5.1 e 5.2, que apresentam a leitura do LED vermelho e do LED infravermelho, respectivamente.

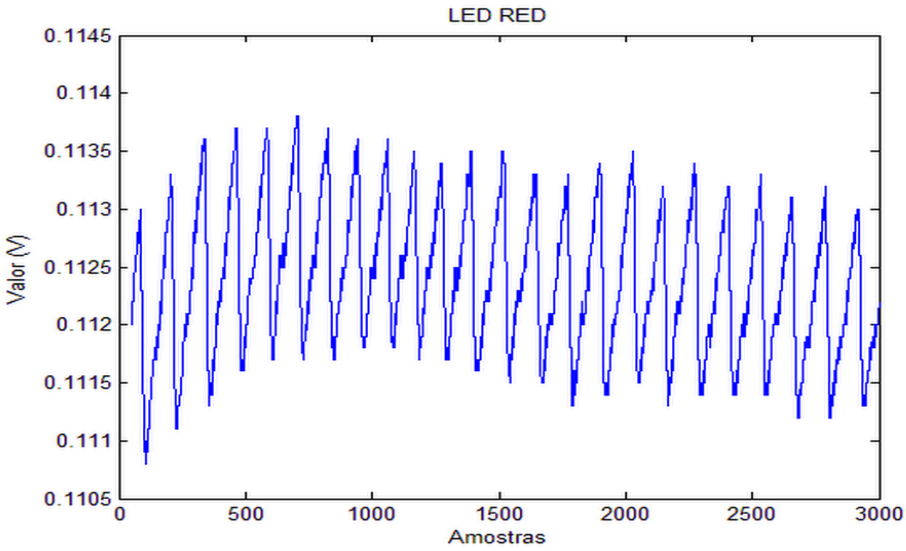



Figura 5.1. Captura Sinal LED vermelho

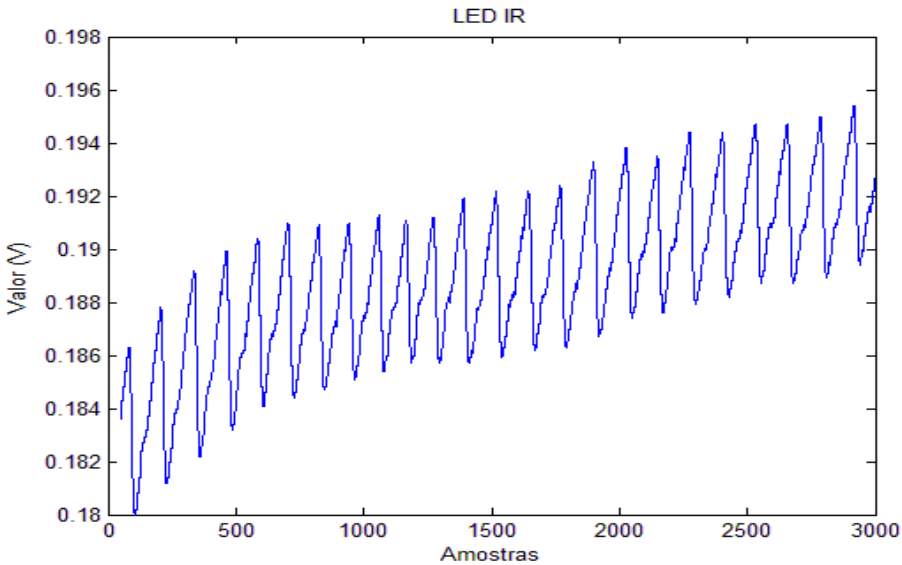



Figura 5.2 Captura Sinal LED infravermelho

## 5.2 Demonstração obtenção BPM

No cálculo de BPM, considerou-se o sinal de leitura referente ao LED infravermelho, pois comparando os dois sinais nas Figuras 5.1 e 5.2, o infravermelho possui uma maior amplitude, o que facilita os cálculos.

Devido aos possíveis atrasos para obtenção e transmissão do sinal faz-se necessário ajustar as variáveis para o cálculo de batimentos cardíacos por minuto. Para o ajuste da taxa de amostragem foram realizadas três leituras com a duração de um minuto e a partir do número de amostras obtidas (4644, 4668 e 4597) em 60s, pode-se afirmar que são lidas em torno de 77 amostras/s, definindo-se a taxa de amostragem. Esta taxa de amostragem deveria ser fixa e deveria considerar os possíveis atrasos de leitura e transmissão, porém optou-se por obtê-la através do número de amostras em certo intervalo para resultados mais aproximados.

Consideramos como teste o sinal de leitura referente ao LED infravermelho demonstrado, em parte, na Figura 5.3 e o mesmo sinal após a passagem pelo filtro passa-baixas na Figura 5.4.

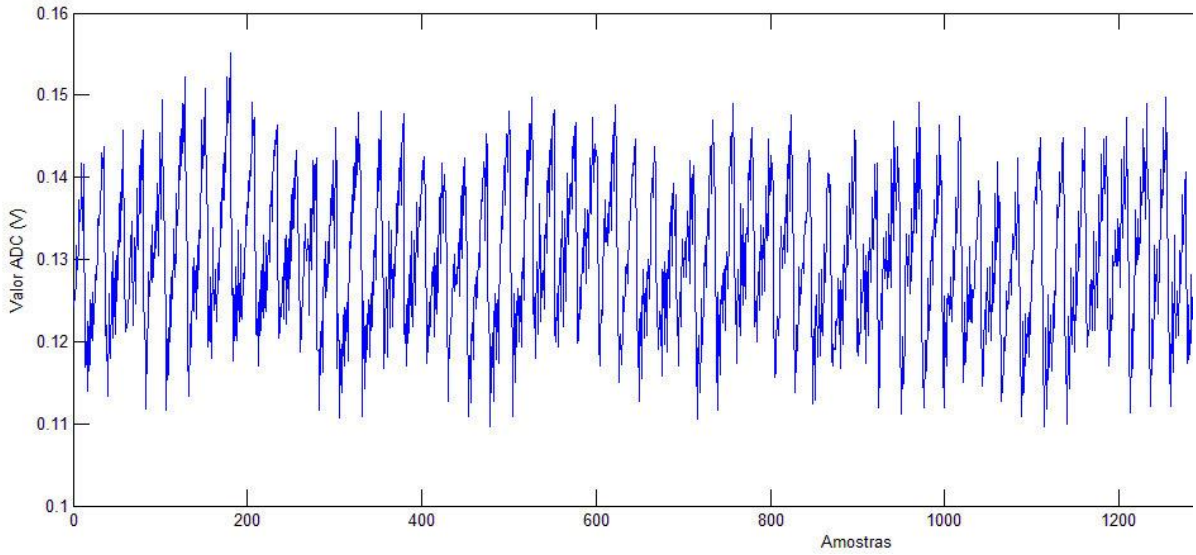



Figura 5.3 Captura Sinal LED infravermelho

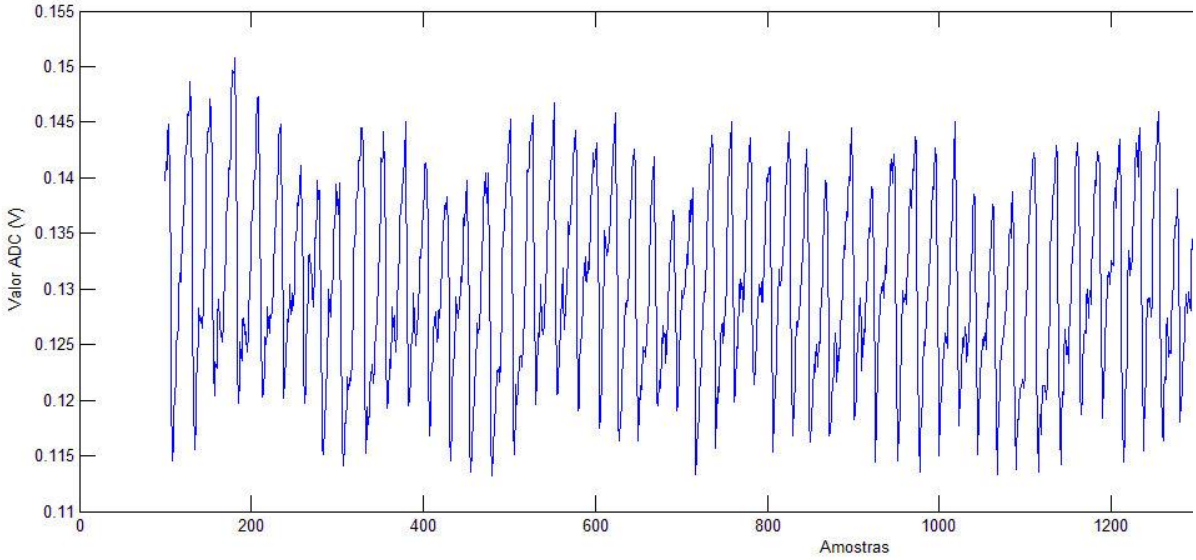



Figura 5.4 Sinal LED infravermelho após filtragem

As Figuras 5.5 e 5.6 ilustram o resultado após a aplicação do algoritmo para detecção de pico e vales do sinal LED infravermelho. Na Figura 5.5 observa-se que foram obtidos picos e vales incorretos, enquanto que na Figura 5.6 o algoritmo detectou de forma correta. Isto se deve ao ajuste do valor do delta, referente ao intervalo mínimo para obtenção de um pico ou vale. No entanto, este valor foi ajustado manualmente a partir da visualização do sinal, e como implementação futura, o algoritmo deve ser capaz de interpretar leituras incorretas e ajustar-se.

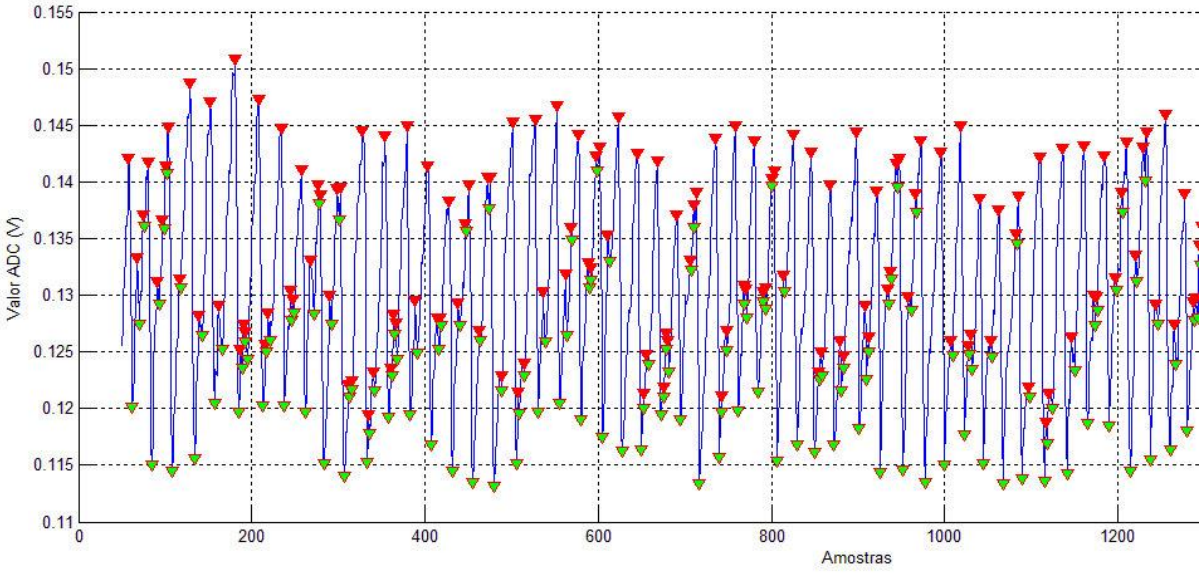



Figura 5.5 Resultado Incorreto Algoritmo Detecção de Pico

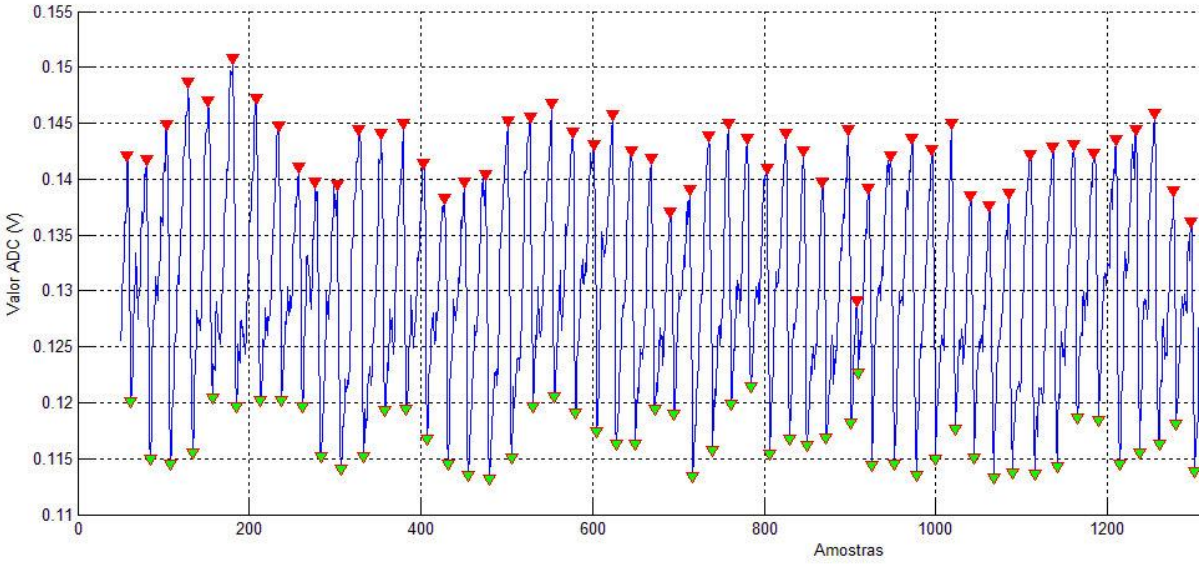



Figura 5.6 Resultado Correto Algoritmo Detecção de Pico

. Após a detecção dos picos, obteve-se através do algoritmo uma média de batimentos cardíacos por minuto variando entre 93 BPM e 96 BPM. Este resultado é satisfatório, conforme atestado por um oxímetro de uso comercial, o qual obteve resultados na mesma faixa de valores.

Com o objetivo de testar a funcionalidade do algoritmo, mais alguns testes foram realizados seguindo os cálculos e ajustes descritos acima. Os resultados estão apresentados na Tabela 5 de acordo com a faixa de valores médios que foram encontrados.

Tabela 5 - Obtenção BPM

| **BPM** | **Dispositivo em teste** | **Dispositivo Comercial** |
|---|---|---|
| **Teste 1** | **108 - 122** | **110 - 115** |
| **Teste 2** | **98 - 101** | **100 - 105** |
| **Teste 3** | **76 - 82** | **77 - 80** |

### 5.3 Demonstração de obtenção %Sp$O_2$

Após a confirmação da correta leitura dos dados, o próximo passo é utilizar o sistema desenvolvido e realizar o cálculo da porcentagem de oxigenação sanguínea. Este teste é realizado com 3 indivíduos com o oxímetro comercial em um dedo e no dedo ao lado o oxímetro em teste. Os resultados de ambos os dispositivos são comparados e apresentados na Tabela 6.

Tabela 6 - Obtenção SPO2

| %Sp$O_2$ | Dispositivo em teste | Dispositivo Comercial |
|---|---|---|
| **Indivíduo 1** | **98** | **98** |
| **Indivíduo 2** | **98** | **99** |
| **Indivíduo 3** | **98** | **97** |

# 6. CONCLUSÕES

A proposta deste trabalho parte da ideia de monitoramento de informações fisiológicas, como o nível de oxigenação do sangue e batimentos cardíacos, possibilitando uma monitoração direta para o paciente, e ou cuidador ou o médico, que pode ser realizada de forma remota, apresentando algumas vantagens, como: cuidados em casa (*home monitoring*), acesso e armazenamento de informações e o alerta de ocorrências. Este sistema também pode ser explorado para levantamento de dados em pesquisas para avaliar a performance de atividades físicas e esportivas.

Os resultados apresentados demonstram um funcionamento satisfatório do sistema, desde a captura dos sinais até o resultado final da porcentagem de oxigenação sanguínea e batimentos cardíacos, gerando dados aceitáveis para o usuário, levando em consideração os mesmos dados obtidos por um dispositivo comercial.

A principal contribuição deste trabalho, entende-se, está na pesquisa e indicação de referências, na implementação dos elementos e demonstração dos resultados necessários à obtenção de um sistema de medidas de monitoramento de oximetria e ritmo cardíaco. Com o material deste trabalho estão lançadas as bases para construção e o aprimoramento de um sistema portátil, de baixo custo e de fácil operação para tais sistemas.

Como sugestão e encaminhamento para continuação de futuros trabalhos, alguns apontamentos se fazem: principalmente no que diz respeito ao sistema de transferência dos dados, que se deseja, sejam em tempo real, para poderem ser processados no sistema móvel, com maior capacidade de processamento e algum paralelismo, fazendo a implementação dos algoritmos de cálculo da oximetria e dos batimentos cardíacos no software aplicativo do dispositivo móvel, de forma a gerar resultados a taxas regulares conforme a necessidade da aplicação. Assim como, projetar a substituição do microcontrolador Arduino pelo ZR-16 (proposta inicial), integrando todos os módulos em uma única PCI, implementando assim, um sistema portátil, semelhante a uma pulseira com relógio, que se comunica com o dispositivo móvel (*Smathphone*). Também pode-se considerar a inserção de outros tipos de sensores como por exemplo, acelerômetro e temperatura por exemplo.

# 7. REFERÊNCIAS

[1] Morón, J.M. et al. **A Wireless Monitoring System for Pulse-Oximetry Sensors**, IEEE Systems Communications – ICW, 2005.

[2] Costin, H. et al. **Complex Telemonitoring of Patients and Elderly People for Telemedical and Homecare Services**, 1st WSEAS International Conference on Biomedical Electronics and Biomedical Informatics, Greece, 2008.

[3] Webster, J. G. **Design of Pulse Oximeters**, Taylor & Francis Group, UK, 1997.

[4] DeMeulenaere, S. **Pulse Oximetry: Uses and Limitations**, The Journal for Nurse Practicioners – JNP, 2007.

[5] Li, Yun-Thai. **Pulse Oximetry**, Department of Electronic Engineering, University of Surrey, UK.

[6] Naoum, P. C; Naoum, F. A. **Fisiologia da Oxigenação.** Disponível em: http://www.hemoglobinopatias.com.br/hemoglobina/fisio-oxi.htm. Acesso 27/03/2015.

[7] Exadaktylos, A; Braun, C. T; Ziaka, M. **Pulse Co-Oximetry – Clinical Impact in the Emergency Department**, Trends in Anaesthesia and Critical Care, Volume 4, 2014.

[8] Lopez, S. **Pulse Oximeter Fundamentals and Design**, Freescale Semiconductor Application Note, Rev.2, 2012.

[9] Kamat, V. **Pulse Oximetry,** Indian J. Anaesth., 2002.

[10] Datasheet: **AFE4490 Integrated Analog Front-End for Pulse Oximeters.** Disponível em: http://www.ti.com/product/AFE4490/datasheet. Acesso 27/03/2015.

[11] **Página Oficial Arduino Uno**. Disponível em: https://www.arduino.cc/en/Main/arduinoBoardUno. Acesso: 18/08/2015.

[12] **SPI –** Serial Peripheral Interface. Disponível em: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. Acesso: 18/08/2015

[13] – **SPI –** Bus Interface. Disponível em: http://www.eeherald.com/section/design-guide/esmod12.html. Acesso: 18/08/2015

[14] Siqueira, S.T; **Bluetooth – Características, protocolos e funcionamento,** Instituto de Computação, Universidade Estadual de Campinas.

[15] **Tecnologia Bluetooth –** O que é e como funciona?. Disponível em: http://www.infowester.com/bluetooth.php. Acesso: 18/08/2015

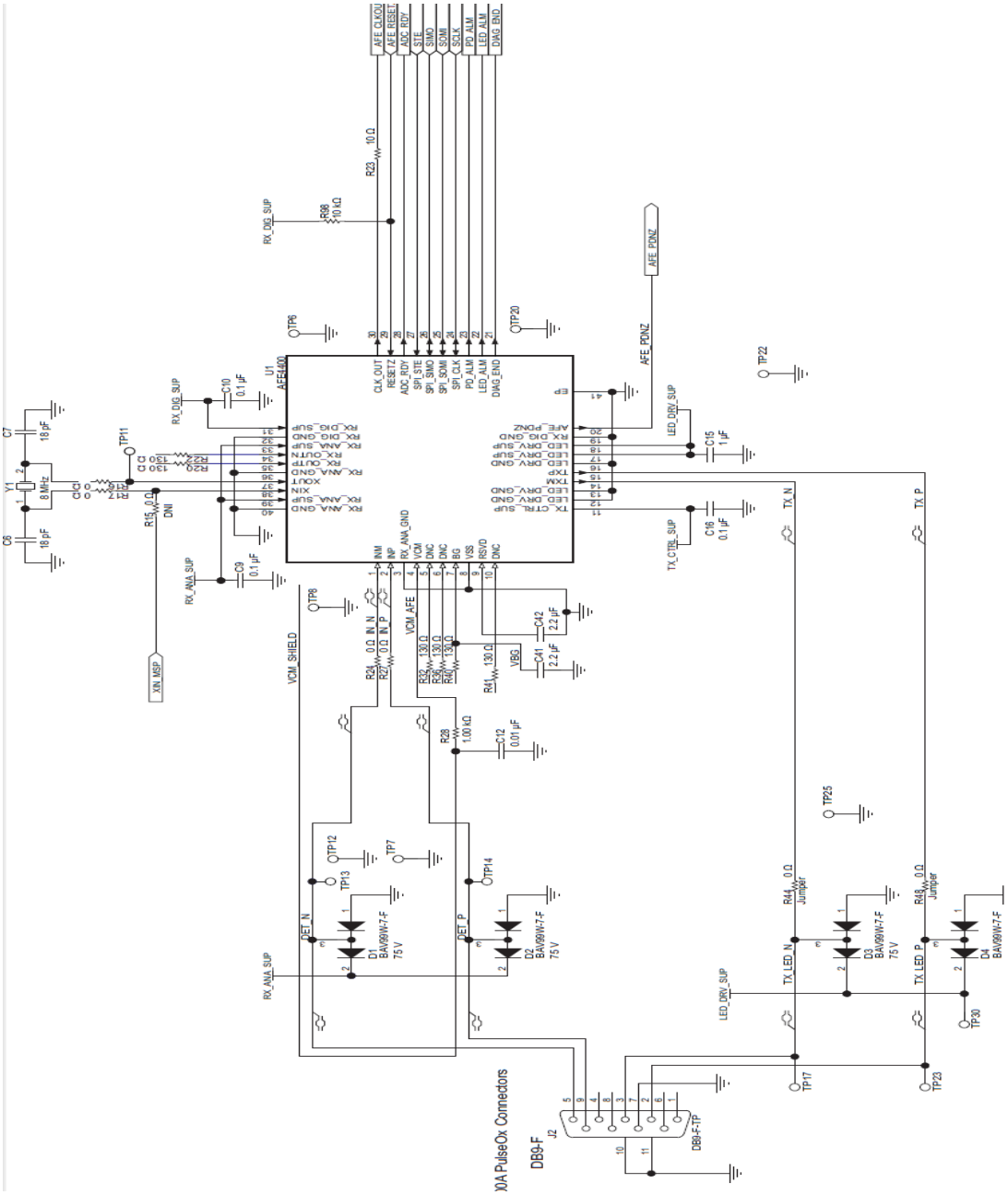
[16] **Android Developers: Bluetooth.** Disponível em: http://developer.android.com/guide/topics/connectivity/bluetooth.html. Acesso: 18/08/2015

[17] **Peak detection using MATLAB.** Disponível em: http://www.billauer.co.il/peakdet.html. Acesso: 18/08/2015

# ANEXOS

**ANEXO A –** Esquemático de uma aplicação utilizando o CI AFE4490

(página 85 do datasheet AFE4490)



## **ANEXO B –** Código Arduino

```
//Inclusão bibliotecas
#include<SPI.h>
#include<SoftwareSerial.h>
#define chipSelectPin 10 //Definição pino SPI STE
SoftwareSerial BT(6,7); //Define os pinos para a serial  (RX, TX)


//------------------------------------------------------------
//Definição nome dos registradores com respectivos endereços
#define LED2STC 0x01
#define CONTROL0 0x00
#define LED2ENDC 0x02
#define LED2LEDSTC 0x03
#define LED2LEDENDC 0x04
#define ALED2STC 0x05
#define ALED2ENDC 0x06
#define LED1STC 0x07
#define LED1ENDC 0x08
#define LED1LEDSTC 0x09
#define LED1LEDENDC 0x0A
#define ALED1STC 0x0B
#define ALED1ENDC 0x0C
#define LED2CONVST 0x0D
#define LED2CONVEND 0x0E
#define ALED2CONVST 0x0F
#define ALED2CONVEND 0x10
#define LED1CONVST 0x11
#define LED1CONVEND 0x12
#define ALED1CONVST 0x13
#define ALED1CONVEND 0x14
#define ADCRSTCNT0 0x15
#define ADCRSTENDCT0 0x16
#define ADCRSTCNT1 0x17
#define ADCRSTENDCT1 0x18
#define ADCRSTCNT2 0x19
#define ADCRSTENDCT2 0x1A
#define ADCRSTCNT3 0x1B
#define ADCRSTENDCT3 0x1C
#define PRPCOUNT 0x1D
#define CONTROL1 0x1E
#define SPARE1 0x1F
#define TIAGAIN 0x20
#define TIA_AMB_GAIN 0x21
#define LEDCNTRL 0x22
#define CONTROL2 0x23
#define SPARE2 0x24
#define SPARE3 0x25
#define SPARE4 0x26
#define RESERVED1 0x27
#define RESERVED2 0x28
#define ALARM 0x29
#define LED2VAL 0x2A //[23:0]
#define ALED2VAL 0x2B
#define LED1VAL 0x2C
#define ALED1VAL 0x2D
#define LED2ALED2VAl 0x2E
```

```
#define LED1ALED1VAL 0x2F
#define DIAG 0x30

const int count = 60;
float IRsignal[count];
float REDsignal[count];
float IRdc[count];
float REDdc[count];
double difIR;
double difRED;
double powIR;
double powRED;
double IRac;
double REDac;
int SPO;
double ratio;
double somaIR = 0;
double somaRED = 0;
String teste;
String teste2;


//------------------------------------------------------------

void SPIWriteReg(byte regAddress, uint32_t regValue)
{
    byte LSB, midByte, MSB;
    digitalWrite(chipSelectPin, LOW);
    SPI.transfer(regAddress);

    uint32_t maskLSB = 0xFF;
    uint32_t maskMidByte = 0xFF00;
    uint32_t maskMSB = 0xFF0000;

    LSB = regValue & maskLSB;
    midByte = (regValue & maskMidByte) >> 8;
    MSB = (regValue & maskMSB) >> 16;

    SPI.transfer(MSB);
    SPI.transfer(midByte);
    SPI.transfer(LSB);
    digitalWrite(chipSelectPin, LOW);
}

void SPIEnableRead()
{
    SPIWriteReg(CONTROL0, 1);
    delay(1);
}

void SPIDisableRead()
{
    SPIWriteReg(CONTROL0, 0);
    delay(1);
}

uint32_t SPIReadReg(byte regAddress)
{
    byte temp_byte_1, temp_byte_2, temp_byte_3;
    long regValue;
    long regValue2;
```

```
    SPIEnableRead();
    digitalWrite(chipSelectPin, LOW);
    SPI.transfer(regAddress);
    // get first byte
    temp_byte_1 = SPI.transfer(0x00);
    delay(10);
    temp_byte_2 = SPI.transfer(0x00);
    delay(10);
    temp_byte_3 = SPI.transfer(0x00);
    delay(10);

    regValue = (temp_byte_1 << 8) | (temp_byte_2);
    regValue2 = (regValue << 8) | temp_byte_3;

    digitalWrite(chipSelectPin, HIGH);

    SPIDisableRead();
    return regValue2;
}


void setup()
{

    Serial.begin(9600);
    SPI.begin();
    BT.begin(9600);
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    pinMode(chipSelectPin, OUTPUT);

afe_init();
}


void loop()
{
    for(int i = 0; i < count; i++)
    {

      IRsignal[i] = (SPIReadReg(LED1VAL) * 1.2) / 2097152;

      //Serial.println(IRsignal[i], 4);
      //delay(20);

      REDsignal[i] = (SPIReadReg(LED2VAL)* 1.2) / 2097152;

      somaIR = somaIR + IRsignal[i];
      somaRED = somaRED + REDsignal[i];

      IRdc[i] = somaIR/i;
      REDdc[i] = somaRED/i;


      difRED = REDsignal[i] - REDdc[i];
      difIR = IRsignal[i] - IRdc[i];


      powIR = pow(difIR, 2.0);
      powRED = pow(difRED, 2.0);
```

```
      IRac =  sqrt(powIR)/i;
      REDac = sqrt(powRED)/i;

      ratio = (REDac/REDdc[i])/(IRac/IRdc[i]);
      SPO = 100 - ratio;




      teste =" ";
      teste.concat(String(SPO));

      BT.println(teste);
      Serial.println(teste);
      delay(300);

  }
}


void afe_init()
{

  SPIWriteReg(CONTROL0, 0);
  SPIWriteReg(CONTROL1, 257);
  SPIWriteReg(SPARE1, 0);
  SPIWriteReg(SPARE2, 0);
  SPIWriteReg(SPARE3, 0);
  SPIWriteReg(SPARE4, 0);
  SPIWriteReg(TIAGAIN, 00);
  SPIWriteReg(TIA_AMB_GAIN, 00);
  SPIWriteReg(LEDCNTRL, 70676);
  SPIWriteReg(CONTROL2, 0);
  SPIWriteReg(RESERVED1, 0);
  SPIWriteReg(RESERVED2, 0);
  SPIWriteReg(ALARM, 0);

  //LED2
  SPIWriteReg(LED2STC, 30080);
  SPIWriteReg(LED2ENDC, 39998);
  SPIWriteReg(LED2LEDSTC, 30000);
  SPIWriteReg(LED2LEDENDC, 39999);
  SPIWriteReg(ALED2STC, 80);
  SPIWriteReg(ALED2ENDC, 9998);

  //LED1
  SPIWriteReg(LED1STC, 10080);
  SPIWriteReg(LED1ENDC, 19998);
  SPIWriteReg(LED1LEDSTC, 10000);
  SPIWriteReg(LED1LEDENDC, 19999);
  SPIWriteReg(ALED1STC, 20080);
  SPIWriteReg(ALED1ENDC, 29998);

  SPIWriteReg(LED2CONVST, 6);
  SPIWriteReg(LED2CONVEND, 9999);
  SPIWriteReg(ALED2CONVST, 10006);
  SPIWriteReg(ALED2CONVEND, 19999);
  SPIWriteReg(LED1CONVST, 20006);
```

```
  SPIWriteReg(LED1CONVEND, 29999);
  SPIWriteReg(ALED1CONVST, 30006);
  SPIWriteReg(ALED1CONVEND, 39999);

  SPIWriteReg(ADCRSTCNT0, 00);
  SPIWriteReg(ADCRSTENDCT0, 05);
  SPIWriteReg(ADCRSTCNT1, 10000);
  SPIWriteReg(ADCRSTENDCT1, 10005);
  SPIWriteReg(ADCRSTCNT2, 20000);
  SPIWriteReg(ADCRSTENDCT2, 20005);
  SPIWriteReg(ADCRSTCNT3, 30000);
  SPIWriteReg(ADCRSTENDCT3, 30005);
  SPIWriteReg(PRPCOUNT, 39999);

  delay(1000);
}
```

## **ANEXO C –** Código Android

MainActivity.java

```
1    package com.example.sandra.bt_hc06_v2;
2
3            import android.bluetooth.BluetoothAdapter;
4            import android.bluetooth.BluetoothDevice;
5            import android.bluetooth.BluetoothSocket;
6            import android.content.Intent;
7            import android.os.Handler;
8            import android.support.v7.app.ActionBarActivity;
9            import android.os.Bundle;
10           import android.view.Menu;
11           import android.view.MenuItem;
12           import android.view.View;
13           import android.widget.Button;
14           import android.widget.EditText;
15           import android.widget.TextView;
16           import android.widget.Toast;
17           import android.os.Message;
18
19           import java.io.IOException;
20           import java.io.InputStream;
21           import java.io.OutputStream;
22           import java.nio.ByteBuffer;
23           import java.util.ArrayList;
24           import java.util.List;
25           import java.util.Set;
26           import java.util.StringTokenizer;
27           import java.util.UUID;
28           import java.util.Vector;
29
30   public class MainActivity extends ActionBarActivity {
31
32
33       Button TurnOn, Conectar,Send;
34       TextView tv, spo2, res;
35       EditText text;
36
37       BluetoothAdapter myBT;
38
39       //List values;
40       ArrayList<Double> list1 = new ArrayList<Double>();
41       ArrayList<Double> list2 = new ArrayList<Double>();
42
43       ArrayList<Double> pks = new ArrayList<Double>();
44       ArrayList<Double> vly = new ArrayList<Double>();
45
46       ArrayList<String> list1_ft = new ArrayList<String>();
47       int count3peak = 0;
48
49
50
51
52       protected static final int MESSAGE_RECEIVE = 1;
53       protected static final int SUCCESS_CONNECT = 0;
54
55       private static final UUID MY_UUID = UUID.fromString("00001101-
0000-1000-8000-00805f9b34fb");
```

```
56
57
58        Handler mHandler = new Handler() {
59            public void handleMessage(Message msg) {
60                super.handleMessage(msg);
61                switch (msg.what) {
62                    case SUCCESS_CONNECT:
63                        ConnectedThread connectedThread = new
ConnectedThread((BluetoothSocket)msg.obj);
64                        connectedThread.start();
65                        Toast.makeText(getApplicationContext(), "CONNECT",
0).show();
66                        break;
67                    case MESSAGE_RECEIVE:
68                        byte[] readBuf = (byte[]) msg.obj;
69
70                        String string = new String(readBuf);
71
72                        res.setText(string);
73
74
75                        break;
76                }
77            }
78        };
79
80
81
82        @Override
83        protected void onCreate(Bundle savedInstanceState) {
84            super.onCreate(savedInstanceState);
85            setContentView(R.layout.activity_main);
86
87            TurnOn = (Button) findViewById(R.id.button_on);
88            Conectar = (Button) findViewById(R.id.button_con);
89            Send = (Button)findViewById(R.id.button_send);
90            tv = (TextView) findViewById(R.id.textView);
91            text = (EditText) findViewById(R.id.editText);
92            spo2 = (TextView)findViewById(R.id.textView2);
93            res = (TextView) findViewById(R.id.textView3);
94
95            myBT = BluetoothAdapter.getDefaultAdapter();
96
97        }
98
99        public void TurnOn(View view) {
100           //Enabling Bluetooth
101           if(!myBT.isEnabled())
102           {
103               Intent TurnOn = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
104               startActivityForResult(TurnOn, 1);
105           }
106           else
107           {
108               Toast.makeText(getApplicationContext(),"Already On",
Toast.LENGTH_LONG).show();
109           }
110       }
111
112
```

```
113
114      public void Conectar(View view) {
115          // botão para conectar e iniciar a comunicação
116          Set<BluetoothDevice> pairedDevices = myBT.getBondedDevices();
117          BluetoothDevice mDevice = null;
118
119          if(pairedDevices.size() > 0) {
120              for (BluetoothDevice device : pairedDevices) {
121                  mDevice = device;
122              }
123          }
124
125          ConnectThread mConnectThread = new ConnectThread(mDevice);
126          mConnectThread.start();
127      }
128
129
130      private class ConnectThread extends Thread {
131
132          private final BluetoothSocket mmSocket;
133          private final BluetoothDevice mmDevice;
134
135
136          public ConnectThread(BluetoothDevice device) {
137              BluetoothSocket tmp = null;
138              mmDevice = device;
139              try {
140                  tmp =
device.createRfcommSocketToServiceRecord(MY_UUID);
141              } catch (IOException e) { }
142              mmSocket = tmp;
143          }
144
145          public void run() {
146              myBT.cancelDiscovery();
147              try {
148                  mmSocket.connect();
149              } catch (IOException connectException) {
150                  try {
151                      mmSocket.close();
152                  }catch (IOException closeException) { }
153                  return;
154              }
155
156              mHandler.obtainMessage(SUCCESS_CONNECT,
mmSocket).sendToTarget();
157
158
159          }
160          public void cancel() {
161              try {
162                  mmSocket.close();
163              } catch (IOException e) { }
164          }
165
166      }
167
168
169      private class ConnectedThread extends Thread {
170          private final BluetoothSocket mmSocket;
171          private final InputStream mmInStream;
```

```
172          private final OutputStream mmOutStream;
173
174          public ConnectedThread(BluetoothSocket socket) {
175              mmSocket = socket;
176              InputStream tmpIn = null;
177              OutputStream tmpOut = null;
178
179              try {
180                  tmpIn = socket.getInputStream();
181                  tmpOut = socket.getOutputStream();
182              } catch (IOException e) { }
183
184              mmInStream = tmpIn;
185              mmOutStream = tmpOut;
186          }
187
188          public void run() {
189              byte[] buffer;
190              int bytes;
191
192              while (true) {
193                  try {
194
195                      buffer = new byte[1024];
196                      bytes = mmInStream.read(buffer);
197                      mHandler.obtainMessage(MESSAGE_RECEIVE, bytes, -1,
buffer).sendToTarget();
198
199                  } catch (IOException e) {
200                      break;
201                  }
202              }}
203
204          public void write(String message) {
205
206              byte[] msgBuffer = message.getBytes();
207              try {
208                  mmOutStream.write(msgBuffer);
209              } catch (IOException e) {
210
211              }
212          }
213      }
214
215
216      public void Clear(View view) {
217          tv.setText("");
218      }
219
220
221      @Override
222      public boolean onCreateOptionsMenu(Menu menu) {
223          // Inflate the menu; this adds items to the action bar if it
is present.
224          getMenuInflater().inflate(R.menu.menu_main, menu);
225          return true;
226      }
227
228      @Override
229      public boolean onOptionsItemSelected(MenuItem item) {
230
```

```
231          int id = item.getItemId();
232
233          if (id == R.id.action_settings) {
234              return true;
235          }
236
237          return super.onOptionsItemSelected(item);
238      }
239  }
240
```

## **ANEXO D** - Código Matlab

```
function peakdec_hr(amostra)

pks = (1:500);
vly = (1:500);

t = (1:length(amostra));

pks_lcs = (1:500);
vly_lcs = (1:500);
count = 1;
count2 = 1;

delta = 0.0065;
mn = inf;
mx = -inf;
mnpos = 0;
mxpos = 0;
lookformax = 1;

count3peak = 0;

for i = 50:length(amostra)
    this = amostra(i);
    if this > mx, mx = this; mxpos = i; end
    if this < mn, mn = this; mnpos = i; end

    if lookformax
        if this < mx - delta
            pks(count) = mx;
            pks_lcs(count) = mxpos;
            count = count + 1;
            mn = this; mnpos = i;
            lookformax = 0;

            count3peak = count3peak + 1;

            if(count3peak == 1)
                xposi = mxpos;

            else
                if(count3peak == 4)
                count3peak = 0;
                xposf = mxpos;
                sub = xposf -xposi;
                rate = 5397/sub;
                disp(rate);
                end
            end


        end
    else
        if this > mn + delta
            vly(count2) = mn;
            vly_lcs(count2) = mnpos;
            count2 = count2 + 1;
            mx = this; mxpos = i;
```

```
            lookformax = 1;
        end
    end

end

figure
hold on
plot(t(50:length(amostra)), amostra(50:length(amostra)));
plot(pks_lcs(1:count-1),amostra(pks_lcs(1:count-
1)),'rv','MarkerFaceColor','r');
plot(vly_lcs(1:count2-1),amostra(vly_lcs(1:count2-
1)),'rv','MarkerFaceColor','g');
grid on;

end
```