

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
CÂMPUS CORNÉLIO PROCÓPIO  
DIRETORIA DE GRADUAÇÃO E EDUCAÇÃO PROFISSIONAL  
DEPARTAMENTO DE COMPUTAÇÃO  
ENGENHARIA DE COMPUTAÇÃO

RHUAN EDSON CALDINI COSTA

**DESENVOLVIMENTO DE UM OSCILOSCÓPIO DIGITAL DE BAIXO  
CUSTO UTILIZANDO AS PLATAFORMAS ARDUINO E ANDROID**

TRABALHO DE CONCLUSÃO DE CURSO

**CORNÉLIO PROCÓPIO**

**2016**

**RHUAN EDSON CALDINI COSTA**

**DESENVOLVIMENTO DE UM OSCILOSCÓPIO DIGITAL DE BAIXO  
CUSTO UTILIZANDO AS PLATAFORMAS ARDUINO E ANDROID**

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina Trabalho de Conclusão de Curso 2, do curso de Engenharia de Computação, Departamento de Computação – DACOM, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. André Takeshi Endo

Co-orientador: Prof. Dr. André Sanches Fonseca  
Sobrinho

**CORNÉLIO PROCÓPIO**

**2016**



---

## **TERMO DE APROVAÇÃO**

Desenvolvimento de um Osciloscópio Digital de Baixo Custo Utilizando as Plataformas  
Arduino e Android

por

Rhuan Edson Caldini Costa

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Engenharia de Computação” e aprovado em sua forma final pelo Programa de Graduação em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Cornélio Procópio,        /        /

---

Prof. Dr. André Takeshi Endo  
Universidade Tecnológica Federal do Paraná

---

Prof. Dr. André Sanches Fonseca Sobrinho  
Universidade Tecnológica Federal do Paraná

---

Prof. Dr. Antonio Carlos Fernandes da Silva  
Universidade Tecnológica Federal do Paraná

Dedico este trabalho à minha família que sempre incentivou meus estudos, aos amigos que me acompanharam durante o curso e aos professores que contribuíram para o conhecimento que carrego hoje.

## **AGRADECIMENTOS**

Agradeço aos meus orientadores Prof. Dr. André Endo e Prof. Dr. André Sanches por todo o tempo dispendido me auxiliando na elaboração deste trabalho, pela dedicação e esforço nesta orientação, e por todo o conhecimento transmitido.

À minha família que sempre incentivou meus estudos, em especial meus pais que se esforçaram para me manter financeiramente enquanto estudante, possibilitando que eu focasse nos estudos e obtivesse um bom desempenho acadêmico.

Aos amigos conquistados durante o curso, em especial Aline Ferreira, Bianca Minetto, Bruno Moura, Felipe Dau e Marcelo Paglione que me acompanharam desde o início do curso e que contribuíram de diversas formas, não apenas em minha vida acadêmica, mas também na vida pessoal. Desde as incontáveis horas de estudo em grupo, até as comemorações de fim de semestre. São amigos que mesmo fisicamente distantes, estarão sempre em minhas lembranças.

Também quero registrar minha gratidão a todos os professores, coordenadores e demais funcionários da Universidade Tecnológica Federal do Paraná de Cornélio Procopio que contribuíram para minha formação e realização desta pesquisa.

## RESUMO

COSTA, Rhuan Edson Caldini. Desenvolvimento de um Osciloscópio Digital de Baixo Custo Utilizando as Plataformas Arduino e Android. 55 f. Trabalho de Conclusão de Curso – Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

O barateamento dos dispositivos eletrônicos aliado ao surgimento de plataformas acessíveis de prototipagem, como o Arduino, tem contribuído para o crescimento no número de pessoas adeptas ao “Faça Você Mesmo” - *Do it Yourself* (DIY). Em alguns projetos, ferramentas mais avançadas como o osciloscópio são necessárias para análise de seu funcionamento. Porém esta é uma ferramenta relativamente cara se comparada ao custo envolvido nos projetos de DIY. Desta forma, o presente trabalho tem como objetivo apresentar o desenvolvimento de um osciloscópio de baixo custo baseado nas plataformas Arduino e Android, bem como avaliar sua eficiência perante um osciloscópio profissional. Espera-se como resultado viabilizar aos praticantes de DIY a construção de um osciloscópio utilizando componentes de fácil acesso, como placas Arduino e dispositivos móveis Android.

**Palavras-chave:** Android, Arduino, Osciloscópio

## ABSTRACT

COSTA, Rhuan Edson Caldini. Development of a Low Cost Digital Oscilloscope Using Arduino and Android Platforms. 55 f. Trabalho de Conclusão de Curso – Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

The cheapening of electronic devices allied to the emergence of affordable prototyping platforms, such as the Arduino, have been contributing to the growth in the number of people adept to Do it Yourself (DIY). In some projects, advanced tools as the oscilloscope are required for the analysis of its operation. However, it is a relatively expensive tool compared to the cost involved in DIY projects. Thus, the present work has as objective to present the development of an oscilloscope based on the Arduino and Android platforms, as well as to evaluate its efficiency towards a professional oscilloscope. It is expected as a result to make feasible to the DIY practitioners the construction of an oscilloscope using easily accessible components, as Arduino boards and Android mobile devices.

**Keywords:** Android, Arduino, Oscilloscope

## LISTA DE FIGURAS

FIGURA 1	– Camadas da Plataforma Google Android .....	15
FIGURA 2	– <i>Layout</i> do Arduino Uno e Nano .....	19
FIGURA 3	– <i>Layout</i> do Arduino Mega 2560 .....	19
FIGURA 4	– Conexão entre ATmega32u4 e Atheros AR9331 no Arduino Yún .....	20
FIGURA 5	– <i>Layout</i> do Arduino Yún .....	21
FIGURA 6	– Esquemático Exemplo .....	21
FIGURA 7	– IDE Arduino .....	22
FIGURA 8	– Amostragem e Quantização de um Sinal Analógico .....	23
FIGURA 9	– Divisão de Tensão Através de uma Série de Resistores .....	24
FIGURA 10	– Configuração Não Inversora do Amp-Op .....	26
FIGURA 11	– Esquemático do Circuito de Amplificação com Offset .....	27
FIGURA 12	– Tela de um Osciloscópio .....	28
FIGURA 13	– Diagrama de Blocos de um Osciloscópio Digital .....	29
FIGURA 14	– Diagrama da Arquitetura Proposta .....	31
FIGURA 15	– Ligação Entre Arduino e Módulo Bluetooth .....	33
FIGURA 16	– Estrutura do Protocolo Implementado .....	34
FIGURA 17	– Bitshift Realizado no Envio de uma Variável de 32 bits .....	36
FIGURA 18	– Diagrama da Aplicação Implementada .....	39
FIGURA 19	– Interface Gráfica GraphActivity .....	41
FIGURA 20	– Esquemático do Circuito de Pré-Condicionamento do Sinal .....	45
FIGURA 21	– Onda Senoidal de 1 Hz, 2 Vpp + 1 Vo .....	48
FIGURA 22	– Onda Senoidal de 250 Hz, 5 Vpp + 2,5 Vo .....	48
FIGURA 23	– Onda Quadrada de 1 kHz, 2 Vpp + 1 Vo .....	49
FIGURA 24	– Onda Senoidal de 3 kHz, 5 Vpp + 2.5 Vo .....	50



## LISTA DE SIGLAS

DIY	Do it Yourself
GUI	Graphical User Interface
ASF	Apache Software Foundation
ART	Android Runtime
AOT	Ahead-of-Time
SDK	Software Development Kit
IDE	Integrated Development Environment
ROM	Read Only Memory
RAM	Random Access Memory
PWM	Pulse Width Modulation
IoT	Internet of Things
PoE	Power over Ethernet
ADC	Analog to Digital Converter
AD	Analógico-Digital
Amp-Op	Amplificador Operacional

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>10</b>
1.1	MOTIVAÇÃO	11
1.2	OBJETIVOS	12
1.3	ORGANIZAÇÃO TEXTUAL	12
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>13</b>
2.1	SISTEMA OPERACIONAL ANDROID	13
2.2	APLICAÇÕES PARA ANDROID	14
2.3	PLATAFORMA ARDUINO	16
2.3.1	Arduino Uno e Nano	18
2.3.2	Arduino Mega 2560	18
2.3.3	Arduino Yún	20
2.3.4	Exemplo de um Projeto com Arduino	21
2.4	CONVERSÃO ANALÓGICO-DIGITAL	22
2.5	PRÉ-CONDICIONAMENTO DO SINAL	24
2.5.1	Circuito de Atenuação	24
2.5.2	Circuito de Amplificação	25
2.5.3	Circuito de Offset	26
2.6	OSCIOSCÓPIO DIGITAL	27
2.7	TRABALHOS RELACIONADOS	29
<b>3</b>	<b>DESENVOLVIMENTO DO OSCIOSCÓPIO</b>	<b>31</b>
3.1	COMUNICAÇÃO ENTRE ARDUINO E DISPOSITIVO MÓVEL	31
3.1.1	Seleção do Meio de Comunicação	32
3.1.2	Implementação do Protótipo	33
3.1.3	Protocolo de Comunicação	34
3.2	FIRMWARE PARA ARDUINO	35
3.2.1	Transmissão Serial	35
3.2.2	Conversão AD	36
3.2.3	Implementação assíncrona	37
3.3	APLICAÇÃO ANDROID	39
3.3.1	Inicialização da Aplicação	39
3.3.2	Interface Gráfica	40
3.3.3	Recebimento e Processamento dos Dados	41
3.4	CIRCUITO DE AQUISIÇÃO DO SINAL	43
3.5	CIRCUITO DE PRÉ-CONDICIONAMENTO DO SINAL	44
<b>4</b>	<b>TESTES E ANÁLISE DOS RESULTADOS</b>	<b>47</b>
<b>5</b>	<b>CONSIDERAÇÕES FINAIS</b>	<b>51</b>
5.1	TRABALHOS FUTUROS	51
	REFERÊNCIAS	53

## 1 INTRODUÇÃO

Nas últimas décadas houve uma grande evolução no desenvolvimento de componentes e produtos eletrônicos. Um marco histórico foi a implementação do transistor na década de 40, que substituiu as caras, grandes e pouco duráveis válvulas, possibilitando a construção de equipamentos menores e mais duráveis (LATHI; DING, 2012; BOYLESTAD; NASHELSKY, 2004; MORIMOTO, 2011). Os processos de produção de dispositivos semicondutores estão cada vez mais otimizados, resultando em componentes mais eficientes, menores e mais baratos. Isto possibilitou a construção de equipamentos operados por meio de baterias, tornando-os compactos e portáteis, por exemplo os dispositivos móveis.

Os *smartphones* se tornaram extremamente populares, atualmente existindo mais de 2 bilhões de aparelhos no mundo, com previsão de ultrapassar a marca dos 2.6 bilhões até 2019 (EMARKETER, 2016). Segundo uma pesquisa da *International Data Corporation* IDC (2015), mais de 80% destes *smartphones* utilizavam o sistema operacional Android em 2015, desenvolvido pelo Google e *open-source*. Estes dispositivos se tornaram parte do cotidiano das pessoas, estando presentes não só como meios de comunicação e entretenimento, mas também como ferramentas na área de saúde, educação e até mesmo militar (FARTO, 2016).

Além do barateamento e miniaturização dos dispositivos eletrônicos, o surgimento de plataformas *open-source* de hardware e software como o Arduino, tem promovido o crescente número de pessoas adeptas ao *do it yourself* (DIY) (KUZNETSOV; PAULO, 2010; NICULESCU; LITA, 2015). Esta prática consiste em construir equipamentos, eletrônicos ou não, por conta própria, utilizando-se de componentes disponíveis no mercado (*off the shelf*).

Muitos dos projetos desenvolvidos pelos hobbystas utilizam-se de microcontroladores para a implementação da lógica de funcionamento. Alguns deles atingem complexidades maiores, envolvendo sinais de alta frequência, consequentemente exigindo ferramentas de diagnóstico mais aprimoradas para a análise de seu funcionamento. Tais ferramentas, como o osciloscópio, costumam ser relativamente caras se comparadas ao custo dos componentes envolvidos nos projetos.

O osciloscópio é um instrumento utilizado na visualização de sinais elétricos, analógicos ou digitais, caracterizado pela sua alta precisão (KARIM, 2014). Sendo muito utilizado por técnicos, engenheiros, mecânicos e até mesmo na área médica, este aparelho é capaz de interpretar sinais com frequência de até centenas megahertz e mostrá-los em um *display* de modo que a forma do sinal seja humanamente compreensível (BOYLESTAD; NASHELSKY, 2004).

## 1.1 MOTIVAÇÃO

Dada a portabilidade dos dispositivos móveis e a elevada capacidade de processamento quando comparados aos microcontroladores, estão surgindo projetos que utilizam ambos os dispositivos trabalhando em conjunto, aproveitando as particularidades de cada um. Um exemplo é a utilização de dispositivos móveis como interface humano-computador devido a sua capacidade de processamento de imagens em duas e três dimensões, além da entrada de comandos via *touch-screen*. Enquanto isso, o microcontrolador é responsável pela aquisição dos sinais de sensores, processamento destes sinais e envio de comandos aos atuadores, além da comunicação com o dispositivo móvel para a interação humana.

As tecnologias mais populares nestes tipos de sistemas são a plataforma Arduino e o sistema operacional móvel Android, ambos por serem dispositivos bem difundidos em meio a comunidade *maker*, como são chamados os praticantes de DIY. Outro ponto importante é a ideologia *open-source* destes dispositivos, que possibilita aos desenvolvedores mais avançados a customização dos mesmos, além da grande quantidade de material técnico disponível na Internet.

No desenvolvimento de projetos que utilizam transmissão de sinais, uma tarefa indispensável é a análise dos sinais envolvidos, sejam eles analógicos ou digitais. Para tal tarefa, é necessário o uso de um osciloscópio, que por ser um equipamento de alto custo, está disponível somente para profissionais da área ou em ambientes específicos, como universidades e empresas do ramo. Alunos de cursos relacionados à eletrônica e sistemas embarcados têm acesso ao osciloscópio das universidades, porém este acesso fica na maioria das vezes, restrito ao momento das aulas práticas, o que pode dificultar o desenvolvimento de projetos pessoais.

Uma forma utilizada para contornar este problema é a construção de osciloscópios simples, de baixo custo, sem as funcionalidades específicas dos instrumentos profissionais como por exemplo *trigger*, que possibilita uma visualização estática de sinais periódicos; e cursor, que auxilia no mensuramento de pontos específicos do sinal. Nestes projetos são utilizados conversores analógico-digital de microcontroladores, e até mesmo de placas de captura de áudio. Estes dispositivos não são desenvolvidos para tal finalidade, porém conseguem obter uma representação satisfatória do sinal analisado desde que este seja de baixa frequência, geralmente abaixo de 10 kHz (ATMEL, 2015). Nestes projetos, um computador é utilizado para o processamento dos dados e como *Graphical User Interface* (GUI), enquanto o microcontrolador fica responsável apenas pela conversão analógico-digital.

Existem também projetos que utilizam a entrada de áudio (microfone externo) de um

dispositivo Android para a captura dos dados analógicos, mostrando o gráfico correspondente na tela do dispositivo. Esta solução varia em precisão dependendo da capacidade do conversor analógico-digital contido no dispositivo móvel, estando limitada a sinais geralmente abaixo de 20 kHz, uma vez que estes dispositivos são projetados para trabalhar na faixa de frequência onde se encontram os sons audíveis ao ser humano, 20 Hz a 20 kHz (YI, 2010).

Diante dos pontos acima mencionados, o desenvolvimento de um osciloscópio de baixo custo construído a partir de componentes facilmente encontráveis no mercado, com satisfatória precisão para utilização em hobbies atenderia às necessidades de pessoas que desejam se aventurar no mundo da eletrônica, porém não pretendem investir em um equipamento caro e considerado de uso profissional.

## 1.2 OBJETIVOS

Devido à popularidade da plataforma Arduino entre os *makers* (praticantes de DIY), e também à grande quantidade de dispositivos móveis com sistema operacional Android, a proposta do presente trabalho é apresentar a construção e análise de um osciloscópio baseado nessas duas plataformas. Para tal análise, foi considerada a fidelidade de representação gráfica do sinal em relação a um osciloscópio profissional, bem como algumas de suas principais funcionalidades.

## 1.3 ORGANIZAÇÃO TEXTUAL

O desenvolvimento deste trabalho está dividido em cinco capítulos. A fundamentação teórica é apresentada no Capítulo 2, subdividido nos principais tópicos abordados nesta proposta: sistema operacional Android, aplicações para Android, plataforma Arduino, conversão analógico-digital, pré-condicionamento do sinal, osciloscópio digital e trabalhos relacionados. Já no Capítulo 3 é exposto o desenvolvimento deste trabalho. No Capítulo 4 são apresentados os testes e resultados. Por fim, no Capítulo 5 encontram-se as considerações finais.

## 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentado o embasamento teórico relacionado a este trabalho, com o objetivo de auxiliar o leitor na compreensão do mesmo. Também serão mencionados trabalhos relacionados a esta proposta. Serão apresentadas as seguintes seções: Na Seção 2.1 são mostrados os principais aspectos do sistema operacional Android; a Seção 2.2 descreve as características das aplicações para Android; na Seção 2.3 a plataforma Arduino é explicada, detalhando alguns modelos de placa e mostrando um exemplo de projeto com Arduino; a Seção 2.4 demonstra o funcionamento da conversão de um sinal analógico para digital; o pré-condicionamento necessário para a conversão de um sinal é explicado na Seção 2.5; o funcionamento de um osciloscópio é demonstrado na Seção 2.6; e por fim são descritos alguns trabalhos relacionados na Seção 2.7.

### 2.1 SISTEMA OPERACIONAL ANDROID

Lançado em 2008, o sistema operacional para *smartphones* Android teve rápida aceitação no mercado. Nos Estados Unidos as vendas de telefones com Android aumentaram em mais de sete vezes em 2010 se comparado com o ano anterior (DEITEL et al., 2013). Desde então este sistema operacional vem ganhando mercado, atingindo 82,8% de participação em 2015 (IDC, 2015). Segundo Ableson et al. (2012), Android é uma plataforma de software que vem revolucionando o mercado global de telefones celulares.

O Android teve seu desenvolvimento iniciado pela Android Inc., comprada pelo Google em 2005. Em 2007 foi formado um consórcio com o objetivo de continuar o desenvolvimento do Android de forma que trouxesse um mercado de dispositivos móveis melhor e mais aberto (DEITEL et al., 2013; ABLESON et al., 2012). Composto inicialmente de 34 empresas, atualmente o consórcio conta com 84 companhias dos ramos de semicondutores, software, dispositivos móveis, operadoras de telefonia e comercialização (DEITEL et al., 2013; OPEN HANDSET ALLIANCE, 2016).

Considerado a primeira plataforma móvel *open-source*, o Android possibilita aos fabricantes de dispositivos móveis modificar, adicionar novas funcionalidades e otimizar o sistema operacional para seus aparelhos sem depender do Google (ABLESON et al., 2012; LECHETA, 2015). Sob a licença *Apache Software Foundation* (ASF), é permitido que alterações sejam feitas no código fonte para customização do sistema operacional sem que haja necessidade de compartilhamento destas alterações (LECHETA, 2015).

O Android conta com um *kernel* baseado em Linux, uma interface gráfica rica, aplicações para usuário final, bibliotecas de código, *frameworks* para aplicações, suporte multimídia e muito mais (ABLESON et al., 2012). O sistema de segurança do Android é baseado na segurança do Linux. No Android, cada aplicação é executada em um único processo, e cada processo possui uma *thread* dedicada. Para cada aplicação instalada é criado um usuário no sistema operacional com acesso a sua estrutura de diretórios. Dessa forma, nenhum outro usuário pode ter acesso a essa aplicação (LECHETA, 2015).

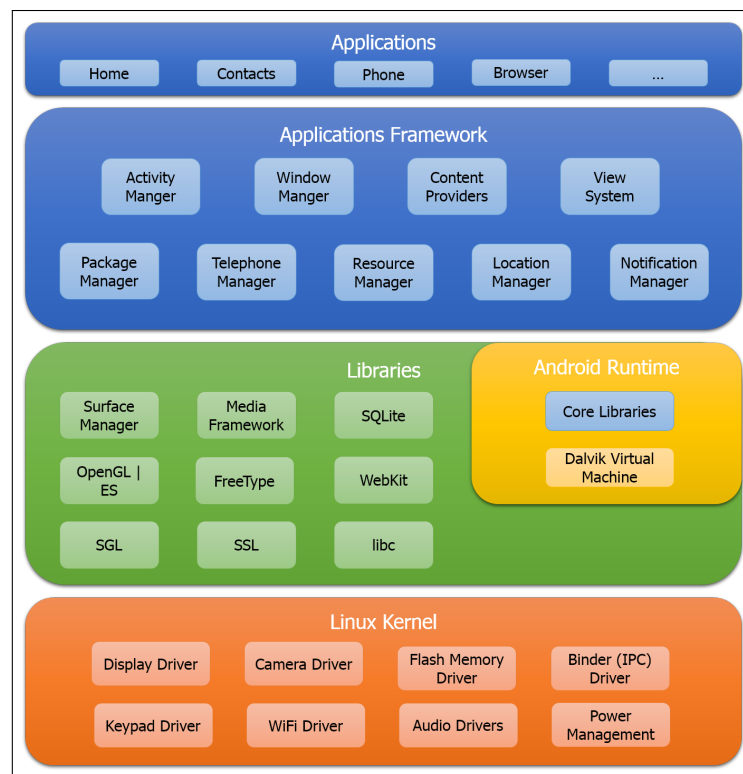
As aplicações desenvolvidas para Android rodam em uma máquina virtual chamada Dalvik, otimizada para a execução em dispositivos móveis. A partir do Android 4.4 (KitKat) foi criada a máquina virtual ART (*Android Runtime*) com o objetivo de substituir a Dalvik. Na versão 5.0 (Lollipop), o ART se tornou a máquina virtual padrão do Android. Uma das melhorias do ART é a compilação *Ahead-of-Time* (AOT), que tem como objetivo otimizar o código para melhorar o desempenho da aplicação. Além disso, houve melhorias no *Garbage Collector*, gerenciador de memória do Android (LECHETA, 2015).

O sistema operacional Android é organizado em cinco camadas projetadas para serem flexíveis, representadas na Figura 1 (ABLESON et al., 2012; VENKATASHAIAH, 2014; CAVALCANTE, 2015):

1. **Applications:** Aplicações do usuário;
2. **Application Framework:** Recursos para as aplicações como *activities* e *views*, janelas, serviços baseados em localização e telefonia;
3. **Libraries:** Bibliotecas com recursos para navegação web (Webkit), banco de dados (SQLite), gráficos avançados (OpenGL), criptografia (SSL), áudio e vídeo;
4. **Android Runtime:** Máquina virtual Dalvik/ART e bibliotecas centrais do Java;
5. **Linux Kernel:** Fornece uma camada de abstração do hardware, bem como *core services* (gerenciamento de processos, memória e arquivos). É no *kernel* onde são implementados os drivers de hardware como Bluetooth, Wi-Fi, *touch-screen*, câmera, GPS, acelerômetro, entre outros.

## 2.2 APLICAÇÕES PARA ANDROID

Enquanto os componentes internos do sistema operacional são escritos em C ou C++, as aplicações de usuário são construídas em Java (ABLESON et al., 2012). A linguagem Java



**Figura 1: Camadas da Plataforma Google Android**

**Fonte: (VENKATASHAIAH, 2014)**

foi uma escolha lógica para o desenvolvimento de aplicações para a plataforma Android, pois é poderosa, gratuita e de código-fonte aberto. Além disso, as aplicações desenvolvidas nesta linguagem podem ser executadas em uma variedade de dispositivos, sem necessidade de código específico para cada plataforma (DEITEL et al., 2013).

Uma característica da plataforma Android é que não há diferença entre aplicações inclusas no sistema operacional e aplicações criadas com o *Android Software Development Kit* (SDK). Isto significa que aplicações poderosas podem ser escritas utilizando os recursos disponíveis no dispositivo (ABLESON et al., 2012). Os principais recursos incluem sensores (acelerômetro, GPS, proximidade, luminosidade, microfone, câmera, giroscópio, entre outros), redes (Wi-Fi, Bluetooth, NFC, GSM, 3G, etc), bancos de dados e integração entre aplicações.

Para o desenvolvimento de aplicações para o Android é necessária a instalação do Android SDK, software contendo um emulador para simular dispositivos Android, ferramentas utilitárias e uma API completa para a linguagem Java. O Android Studio é a IDE (Integrated Development Environment) oficial de desenvolvimento, fornecida pelo Google, e já contém o SDK (LECHETA, 2015).



## 2.3 PLATAFORMA ARDUINO

Segundo o site oficial, Arduino é uma plataforma de prototipagem *open-source* baseada em hardware e software fáceis de utilizar. Ao longo dos anos, o Arduino tem sido o cérebro de milhares de projetos, desde objetos do dia a dia até instrumentos científicos complexos. Uma comunidade mundial de *makers* (estudantes, *hobbyistas*, artistas, programadores e profissionais) tem se reunido ao redor desta plataforma *open-source* e suas contribuições têm somado uma incrível quantidade de conhecimento acessível que pode ser de grande ajuda tanto para novatos quanto para especialistas. Segundo Arduino (2016a), a plataforma simplifica o processo de trabalhar com microcontroladores, oferecendo algumas vantagens para professores, estudantes e amadores, entre elas:

- **Baixo custo:** placas Arduino são relativamente baratas se comparadas com outras plataformas de microcontroladores;
- **Multi-plataforma:** a IDE Arduino roda em Windows, Mac OS e Linux enquanto a maioria dos sistemas de microcontroladores são limitados ao Windows;
- **Ambiente de programação simples e limpo:** fácil de usar para iniciantes enquanto flexível o suficiente para usuários avançados;
- **Software *open-source* e extensível:** O software Arduino está disponível para aprimoramentos por programadores experientes. A linguagem pode ser expandida por meio de bibliotecas escritas em C++ e usuários mais experientes podem utilizar a linguagem AVR-C, específica para microcontroladores Atmel AVR, diretamente nos programas Arduino;
- **Hardware *open-source* e extensível:** Os esquemáticos das placas Arduino são publicados sob a licença *Creative Commons*, possibilitando a confecção de versões modificadas das mesmas.

Para se aprofundar nas especificações de hardware do Arduino, primeiramente é necessário compreender o que é um microcontrolador. De acordo com Bayle (2013), este é um circuito integrado contendo todas as partes principais de um computador típico, sendo elas:

- O **processador** é o cérebro do microcontrolador, onde cálculos são realizados e todas as decisões são tomadas;

- As **memórias** são espaços onde o programa e os elementos de usuário (variáveis) estão. São encontradas nos tipos *Read Only Memory* (ROM), mais lentas, porém não voláteis, usadas para armazenar o programa; e *Random Access Memory* (RAM), rápidas porém voláteis, usadas para armazenar os dados em tempo de execução;
- Os **periféricos** possuem a função de auxiliar o processador e estender suas capacidades. Exemplos são conversores analógico-digital, *timers*, interrupções, barramentos de comunicação (serial, SPI, I2C), geradores de PWM (Pulse Width Modulation), entre outros;
- As **entradas e saídas** são as vias de comunicação entre o mundo e o microcontrolador. As entradas são usadas para ler sensores e receber mensagens, enquanto as saídas comandam atuadores e enviam mensagens.

O alto nível de integração fornecido pelos microcontroladores - tudo em um único chip - os fazem ideais para sistemas embarcados, sendo usados desde em sistemas simples como controles remotos, passando por dispositivos mais complexos como *smartphones*, até em sistemas críticos como a injeção eletrônica de automóveis (BAYLE, 2013). Os principais aspectos a serem analisados em um microcontrolador são: clock, número de bits do processador, quantidade de memórias e periféricos disponíveis.

Atualmente existem dezenas de modelos de placas Arduino, a maioria delas baseada em microcontroladores Atmel AVR de 8 bits. A primeira placa desenvolvida possuía um ATmega8 rodando a um clock de 16 MHz com 8 KB de memória *flash*. Posteriormente foi adotado o uso do ATmega168, com 16 KB de memória *flash*. As versões mais recentes do Arduino contam com o microcontrolador ATmega328, com 32 KB de memória *flash*. Para projetos que requerem mais memória existe ainda o modelo Arduino Mega 2560, com 256 KB (EVANS et al., 2013).

Placas especialistas denominadas *shields* têm como função expandir as funcionalidades básicas dos Arduinos. Elas podem ser empilhadas, umas em cima das outras, somando múltiplas funcionalidades ao microcontrolador (EVANS et al., 2013). Exemplos de funcionalidades adicionáveis por meio de *shields* são Bluetooth, Wi-Fi, Ethernet, controle de motores elétricos, comunicação GSM, alimentação por bateria, entre outros.

Grande parte das placas Arduino segue um padrão de pinagem para que, independente do modelo, os *shields* sejam compatíveis, além de tornar a elaboração de esquemáticos universal. Este padrão conta com 14 pinos digitais de I/O dos quais seis são capazes de fornecer saída PWM, seis entradas analógicas, além de protocolos de comunicação como serial, *Serial*

*Peripheral Interface* (SPI) e I2C. Cada placa também possui um conector (*header*) *In-Circuit Serial Programming* (ICSP) e um botão reset (EVANS et al., 2013). Existem modelos de placas que não seguem o padrão de pinagem, desenvolvidos em circuitos menores com o intuito de serem aplicados em projetos do tipo *Internet of Things* (IoT) e *wearables*. No momento do desenvolvimento do presente trabalho, os modelos mais conhecidos devido as suas particularidades são Uno, Mega 2560 e Yún, e serão descritos a seguir.

### 2.3.1 ARDUINO UNO E NANO

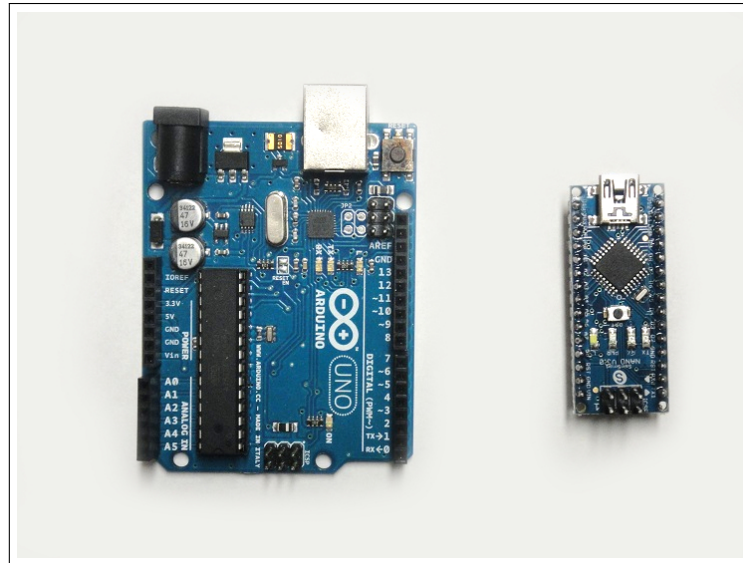
Muito utilizada por iniciantes, a versão mais popular do Arduino atualmente é a Uno, lançada em 2010. A principal diferença entre ele e seus antecessores é a presença de, além do microcontrolador principal ATmega328, um microcontrolador ATmega16U2 programado como conversor USB-serial. Esta conversão é necessária para que o Arduino se comunique com o computador através da porta USB, sendo assim programado. O ATmega16U2 pode ser reprogramado, fazendo com que o Arduino se comporte como outro dispositivo USB como mouse, teclado ou *joystick* (EVANS et al., 2013; ARDUINO, 2016d).

O Arduino Uno pode ser alimentado pela porta USB, a mesma utilizada para programá-lo, ou por uma fonte externa. A fonte de energia é selecionada automaticamente pela placa, que possui um regulador de tensão possibilitando-a ser alimentada com tensões entre 6 e 20 volts, porém recomenda-se tensões entre 7 e 12 volts para evitar instabilidade do microcontrolador ou superaquecimento do regulador (ARDUINO, 2016d).

O Arduino Nano é uma versão miniaturizada do Uno, medindo 18x45 mm contra 54x69 mm do maior. Possui as mesmas funcionalidades pois também é composto pelo microcontrolador ATmega328 (ARDUINO, 2016c). Suas dimensões e pinagem permitem que o mesmo seja conectado diretamente à *protoboards*, facilitando sua utilização em conjunto com pequenos circuitos. Esta versão também é utilizada em algumas soluções finais, sendo conectados ao PCB de forma semelhante à circuitos integrados com encapsulamento DIP. Na Figura 2 são mostrados o *layout* do Arduino Uno e Nano.

### 2.3.2 ARDUINO MEGA 2560

O Arduino Mega 2560 é geralmente utilizado em projetos que demandam uma maior quantidade de pinos de I/O, bem como maior espaço de memória para sua programação, ou ainda múltiplas portas seriais. Além da pinagem padrão compatível com o Arduino Uno, consequentemente com os *shields*, o Mega conta com mais três conjuntos de pinos totalizando 54

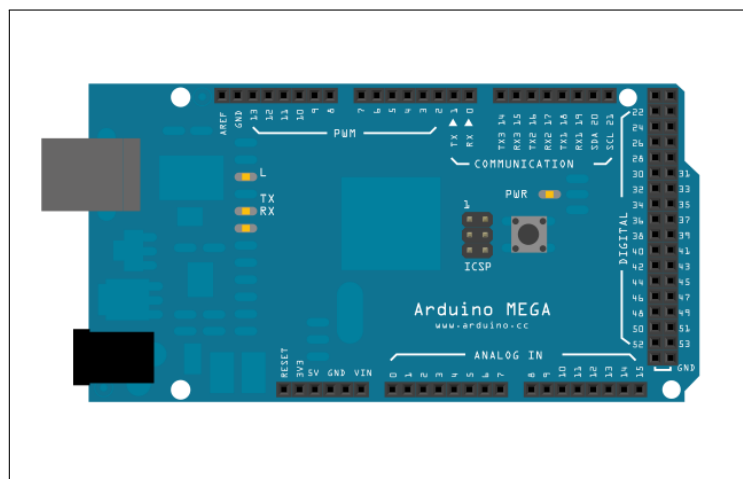


**Figura 2: *Layout* do Arduino Uno e Nano**

**Fonte: Extraído de (MAKERS, 2014)**

pinos de I/O dos quais 15 suportam PWM, 16 entradas analógicas e quatro portas serial UART (ARDUINO, 2016b).

O microcontrolador utilizado na construção desta placa é o ATmega2560, com 256 KB de memória flash, rodando aos mesmos 16 MHz da versão Uno. Este modelo também conta com um ATmega16U2 trabalhando como conversor USB-serial (EVANS et al., 2013). O *layout* do Arduino Mega 2560 é mostrado na Figura 3.



**Figura 3: *Layout* do Arduino Mega 2560**

**Fonte: Extraído de (ARDUINO, 2016b)**

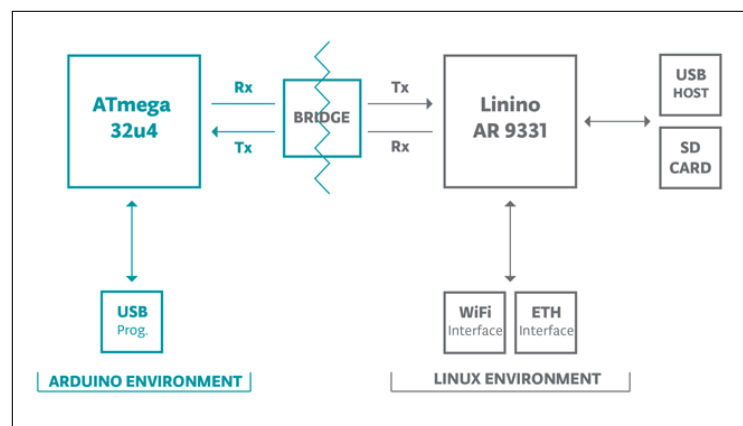
Da mesma forma que o Arduino Uno, é possível alimentar o Mega 2560 pela porta de

programação USB com tensão de 5 volts, ou por uma fonte externa entre 6 e 20 volts (recomendado de 7 a 12 volts) por meio do conector *jack* ou pelos pinos  $V_{in}$  e  $Gnd$  (ARDUINO, 2016b).

### 2.3.3 ARDUINO YÚN

Diferente das versões citadas anteriormente, o Arduino Yún possui além de um microcontrolador, um microprocessador com suporte a uma distribuição Linux baseada no OpenWrt chamado OpenWrt-Yun. Este processador fornece ao Yún as funcionalidades Wi-Fi, Ethernet, USB Host e um slot micro-SD (ARDUINO, 2016e). Portanto este modelo é geralmente utilizado em projetos que requerem conexão com a Internet.

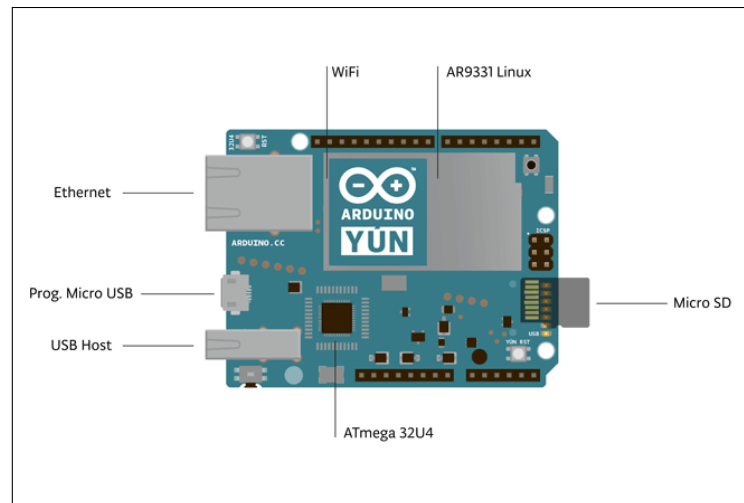
O microcontrolador utilizado nesta versão é o ATmega32u4, que possui especificações similares ao ATmega328 utilizado no Uno. Seu maior diferencial encontra-se na comunicação USB integrada, dispensando a necessidade de outro chip dedicado a esta função. O microprocessador Linux modelo Atheros AR9331 é capaz de executar scripts escritos em Python e se comunica com o microcontrolador através de uma conexão serial. Um diagrama representando a conexão entre os dois ambientes pode ser visto na Figura 4.



**Figura 4: Conexão entre ATmega32u4 e Atheros AR9331 no Arduino Yún**

**Fonte: Extraído de (ARDUINO, 2016e)**

Outra particularidade deste modelo de Arduino é que ele não possui regulador de tensão integrado. Portanto deve ser alimentado pela porta micro-USB ou injetando 5 volts no pino  $V_{in}$  da placa. Outra possibilidade é alimentá-lo por um módulo *Power over Ethernet* (PoE) vendido separadamente. O *layout* do Arduino Yún é mostrado na Figura 5.

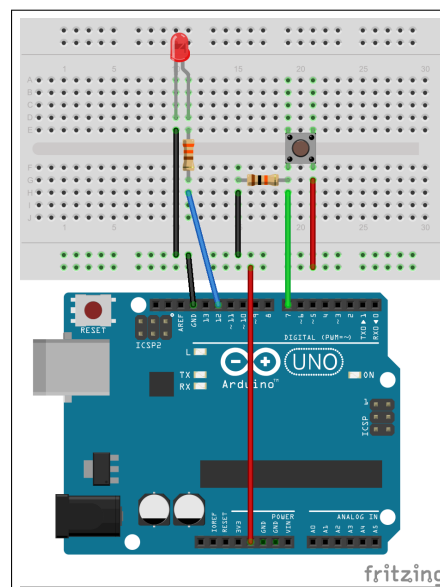


**Figura 5: Layout do Arduino Yún**

**Fonte: Extraído de (ARDUINO, 2016e)**

### 2.3.4 EXEMPLO DE UM PROJETO COM ARDUINO

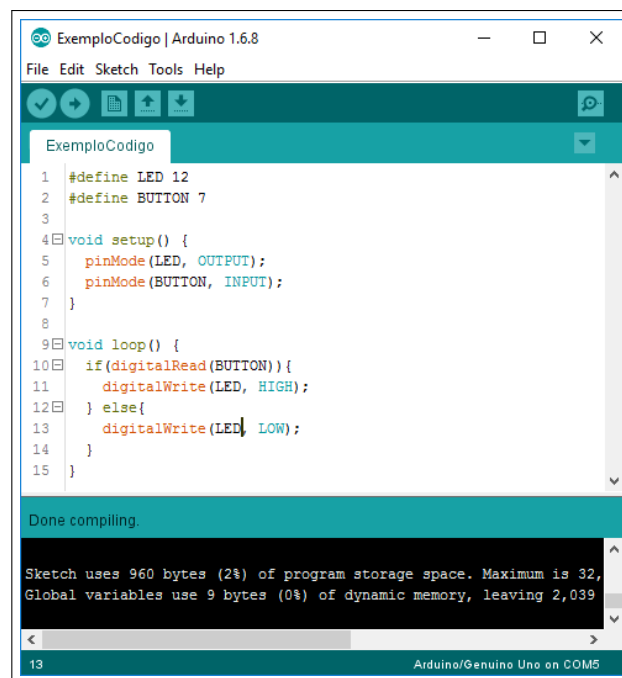
Um exemplo de projeto desenvolvido para a plataforma Arduino é mostrado na presente seção para fins de demonstração. O projeto consiste em acender um LED enquanto um botão estiver pressionado. O esquemático correspondente à este circuito pode ser visto na Figura 6. O mesmo foi desenvolvido no software Fritzing, de iniciativa *open-source*, que segue o mesmo espírito da plataforma Arduino (FRITZING, 2016).



**Figura 6: Esquemático Exemplo**

**Fonte: Autoria Própria**

Uma captura de tela da IDE Arduino contendo o código necessário para o funcionamento do projeto exemplo é mostrado na Figura 7. Os pinos correspondentes ao LED e botão são definidos em constantes no início do código para facilitar sua referência no decorrer do programa. A função `setup()` é executada somente uma vez ao ligar o microcontrolador e é utilizada para a inicialização dos pinos de I/O, configuração de sensores, entre outras atividades que não necessitam ser repetidas. Após sua execução, as instruções contidas na função `loop()` são executadas repetidamente caracterizando o funcionamento contínuo do microcontrolador. Nesta função são lidos sensores, realizado o processamento de acordo com a lógica do programa, e controlados os atuadores.



```

ExemploCodigo | Arduino 1.6.8
File Edit Sketch Tools Help

ExemploCodigo
1 #define LED 12
2 #define BUTTON 7
3
4 void setup() {
5   pinMode(LED, OUTPUT);
6   pinMode(BUTTON, INPUT);
7 }
8
9 void loop() {
10  if(digitalRead(BUTTON)){
11    digitalWrite(LED, HIGH);
12  } else{
13    digitalWrite(LED, LOW);
14  }
15 }

Done compiling.

Sketch uses 960 bytes (2%) of program storage space. Maximum is 32,
Global variables use 9 bytes (0%) of dynamic memory, leaving 2,039

13 Arduino/Genuino Uno on COM5

```

**Figura 7: IDE Arduino**

**Fonte: Autoria Própria**

## 2.4 CONVERSÃO ANALÓGICO-DIGITAL

A grande maioria dos sinais no mundo é analógica, podendo assumir qualquer valor em qualquer instante de tempo (SEDRA; SMITH, 2000). Sinais analógicos são contínuos no tempo e em um intervalo de valores, logo têm valores em cada instante de tempo, e esses valores podem ter qualquer amplitude no dado intervalo. Por outro lado, sinais digitais existem apenas em momentos discretos no tempo e podem assumir somente um número finito de valores (LATHI; DING, 2012).

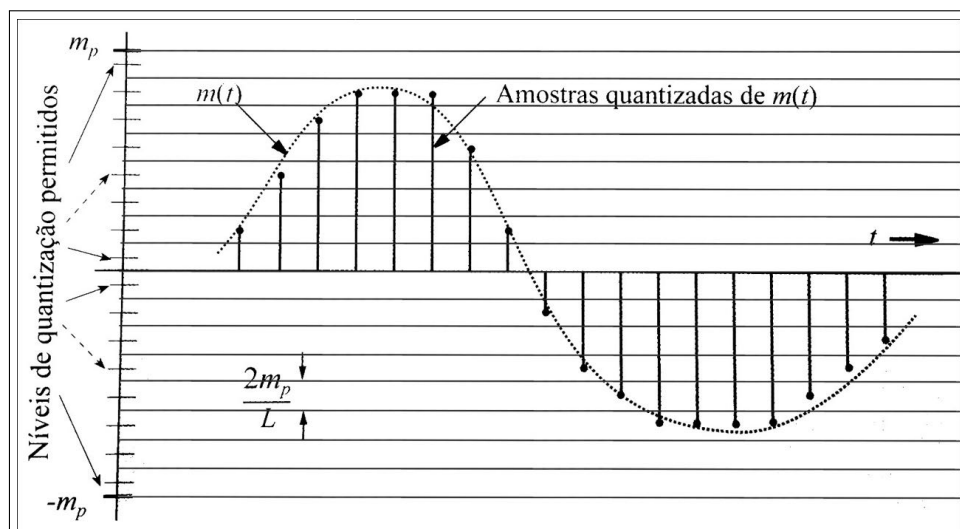
Para trabalhar com sinais analógicos em microcontroladores, que são dispositivos digi-

tais, primeiramente é necessário converter a tensão do sinal no dado momento para um valor em binário que o represente. Esta conversão é feita por meio de conversores analógico-digital ou, em inglês, *Analog to Digital Converter* (ADC). Grande parte dos microcontroladores possuem este periférico incluso em seu circuito integrado, como é o caso dos utilizados nos Arduinos. A conversão analógico-digital (AD) não é 100% precisa, entretanto, como a percepção humana não exige precisão infinita, a conversão AD pode capturar a informação necessária de uma fonte analógica de forma satisfatória (LATHI; DING, 2012).

A conversão AD ocorre em duas etapas: um sinal em tempo contínuo é amostrado produzindo um sinal em tempo discreto cujas amplitudes contínuas são então quantizadas em níveis discretos. O teorema de Nyquist afirma que para que um sinal possa ser reconstruído a partir de suas amostras sem perda de informações, é necessário que a frequência de amostragem seja no mínimo o dobro da maior frequência do espectro do sinal (LATHI; DING, 2012).

Estando o sinal amostrado em intervalos de tempo uniformes, os valores das amostras ainda não se encontram na forma digital, pois assumem valores em um intervalo contínuo. Então a quantização é efetuada, processo no qual cada amostra é aproximada ao nível de quantização mais próximo.

Para um sinal  $m(t)$  com valores no intervalo  $(-m_p, m_p)$ , o quantizador divide este intervalo em  $L$  subintervalos. Cada amostra de amplitude é aproximada ao valor médio do subintervalo que a contém, e passa a ser representada por um dos  $L$  números. Após estes dois passos, amostragem e quantização, a conversão AD se conclui. Tais etapas são ilustradas na Figura 8.



**Figura 8: Amostragem e Quantização de um Sinal Analógico**

**Fonte: Adaptado de (LATHI; DING, 2012)**



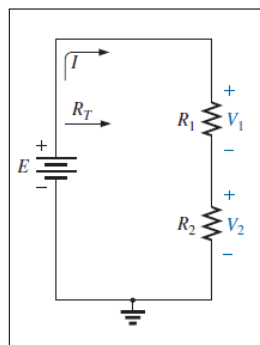
A precisão do sinal quantizado pode ser aumentada por meio do incremento do número de níveis  $L$  (LATHI; DING, 2012). Na prática, a quantidade de níveis é dada por  $2^n$ , onde  $n$  é a quantidade de bits utilizada para representar cada valor de forma digital. Por exemplo, quando temos um ADC de 8 bits, temos 256 níveis ( $2^8$ ), sendo cada nível representado por uma sequência de 8 bits.

## 2.5 PRÉ-CONDICIONAMENTO DO SINAL

Para que um conversor AD possa trabalhar com as mais variadas formas de onda é necessário um tratamento prévio do sinal para que o mesmo se adeque aos valores aceitos pelo conversor. Serão necessários três circuitos de condicionamento do sinal: atenuação, amplificação e *offset*.

### 2.5.1 CIRCUITO DE ATENUAÇÃO

Para que tensões acima de 5 V possam ser medidas, é necessário diminuir essa tensão para valores entre 0 e 5 V. Este efeito é conseguido através de divisores resistivos. Segundo Boylestad (2007), a tensão através uma série de elementos resistivos se dividirá conforme a magnitude dos níveis de resistência. A proporção de tensão através de resistores em série será a mesma proporção de suas resistências. Uma representação de um divisor resistivo com dois resistores pode ser visto na Figura 9.



**Figura 9: Divisão de Tensão Através de uma Série de Resistores**

**Fonte: Extraído de (BOYLESTAD, 2007)**

A regra do divisor de tensão (Equação 1) permite a determinação da tensão em qualquer resistor de uma série sem necessidade da determinação da corrente através deles. A tensão  $V_x$  no resistor  $x$  dá-se pela multiplicação de sua resistência ( $R_x$ ) pela tensão ( $E$ ) aplicada na série de resistores, dividida pela soma de todas as resistências ( $R_T$ ).

$$V_x = R_x \frac{V}{R_T} \quad (1)$$

Este método de atenuação fornece uma fração fixa da tensão de entrada, levando em consideração que a saída do divisor esteja conectada à um circuito com alta impedância de entrada, cuja corrente drenada do divisor possa ser considerada nula.

### 2.5.2 CIRCUITO DE AMPLIFICAÇÃO

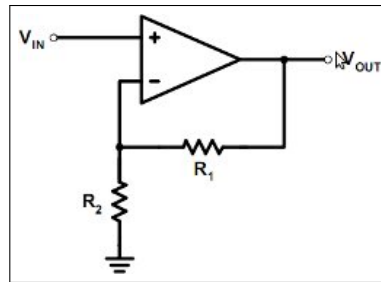
Em ocasiões onde se deseja ler um sinal com valor de tensão muito baixo, para que a representação do sinal tenha uma resolução satisfatória é necessário amplificá-lo de forma que ele se estenda pela maior faixa possível dentro da faixa de tensão aceita pelo conversor, usualmente entre 0 e 5 V. Para isso é utilizado o amplificador operacional (Amp-Op) que, de acordo com a combinação de resistores utilizados em sua realimentação, oferece determinados valores de ganho.

Ao amplificar um sinal, deve-se ter cuidado para não modificar a informação contida no mesmo. O intuito é que o sinal de saída do amplificador seja uma réplica exata do sinal de entrada, exceto sua amplitude (tensão) que deve ser maior. Dá-se à essa propriedade dos amplificadores o nome de linearidade (SEDRA; SMITH, 2000).

A proporção de amplificação do sinal é conhecida como ganho do amplificador e, no caso dos Amp-Ops, pode ser configurado dentro de um limite definido pelas características do componente, de acordo com a combinação de resistores utilizada na realimentação de sua entrada. As duas configurações mais comuns para um Amp-Op são amplificador inversor, o qual tem como saída o sinal amplificado defasado 180° em relação ao sinal de entrada; e amplificador não inversor, que não apresenta tal defasagem (JÚNIOR, 2003). A configuração não inversora é a que melhor atende este projeto pois alterações no sinal não são desejadas.

O circuito de configuração de um Amp-Op como amplificador não inversor pode ser visto na Figura 10. Nesta configuração o sinal de entrada  $V_{in}$  é aplicado diretamente ao terminal de entrada positivo (+) do Amp-Op, enquanto um dos terminais de  $R_2$  é conectado ao terra e o outro à entrada não inversora (-). O resistor  $R_1$  realimenta o sinal de saída para a entrada não inversora (-) do amplificador (SEDRA; SMITH, 2000; JÚNIOR, 2003). Nesta configuração, o ganho obtido pelo amplificador dá-se segundo a Equação 2.

$$A = 1 + \frac{R_2}{R_1} \quad (2)$$



**Figura 10: Configuração Não Inversora do Amp-Op**

**Fonte: Extraído de (ELETRÓNICA, 2016)**

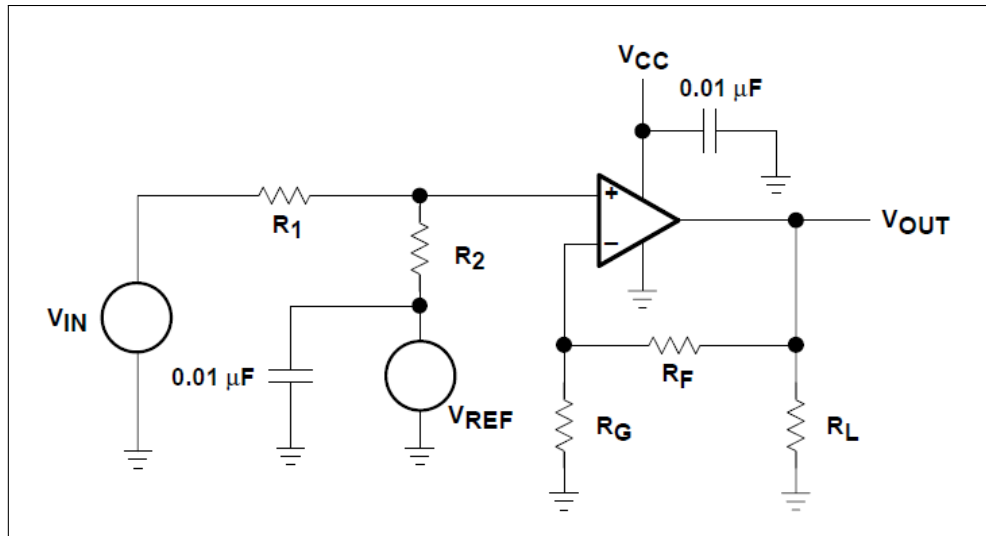
### 2.5.3 CIRCUITO DE OFFSET

Quando se deseja analisar uma forma de onda alternada, onde os valores oscilam acima e abaixo de zero, é necessário que os valores negativos sejam elevados para que fiquem acima de zero pois os conversores AD alimentados com 5 volts positivos não são capazes de reconhecer valores negativos. Desta forma, o sinal é acrescido de uma tensão contínua fazendo com que toda a onda fique dentro da faixa de tensão aceita pelo conversor. Este efeito é denominado *offset* e também é conseguido por meio do amplificador operacional (JÚNIOR, 2003; SEDRA; SMITH, 2000). Geralmente, é necessário utilizar o *offset* em conjunto com divisores resistivos para tensões alternadas maiores que a aceita pelo conversor AD, ou em conjunto com amplificadores para tensões alternadas pequenas, na casa dos milivolts. Caso o sinal de entrada já possua um *offset* qualquer, primeiramente é necessário realizar o desacoplamento deste *offset* por meio da utilização de um capacitor na entrada do circuito, formando um filtro passa-baixas, para só então aplicar o *offset* de valor conhecido no sinal.

Segundo a fabricante de componentes eletrônicos, TEXAS INSTRUMENTS (2008), para se obter um ganho positivo (não inversor) com uma tensão de offset pode-se utilizar o circuito representado na Figura 11.

Dado o ganho linear do amplificador operacional, a equação que representa a saída deste circuito pode ser representada resumidamente por uma equação de reta (Equação 3) onde  $G$  é o ganho do circuito e  $O$  é a tensão de offset. A equação real do circuito obtida utilizando as regras de divisor de tensão e superposição é mostrada na Equação 4.

$$V_{out} = G \cdot V_{in} + O \quad (3)$$



**Figura 11: Esquemático do Circuito de Amplificação com Offset**

**Fonte: Extraído de (TEXAS INSTRUMENTS, 2008)**

$$V_{out} = V_{in} \cdot \left( \frac{R_2}{R_1 + R_2} \right) \cdot \left( \frac{R_F + R_G}{R_G} \right) + V_{ref} \cdot \left( \frac{R_1}{R_1 + R_2} \right) \cdot \left( \frac{R_F + R_G}{R_G} \right) \quad (4)$$

Relacionando as Equações 3 e 4 obtém-se as equações que relacionam os valores de ganho (Equação 5) e *offset* (Equação 6) desejados com os valores dos resistores a serem utilizados no circuito.

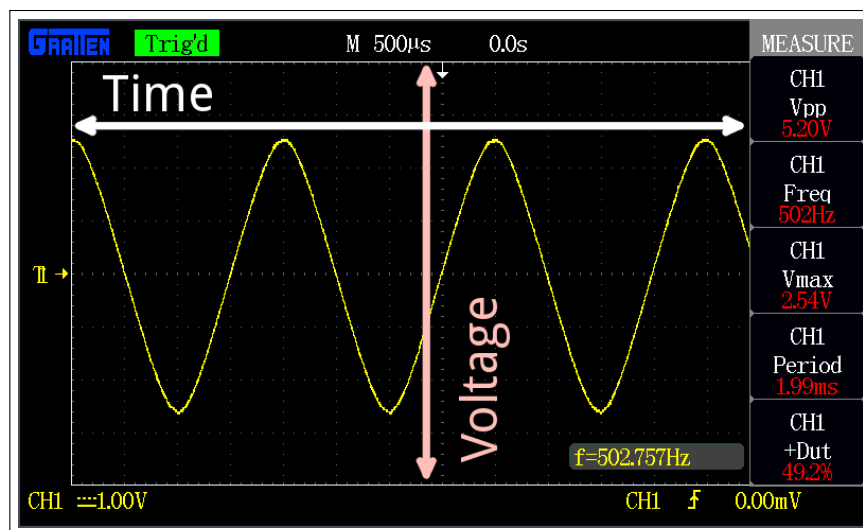
$$G = \left( \frac{R_2}{R_1 + R_2} \right) \cdot \left( \frac{R_F + R_G}{R_G} \right) \quad (5)$$

$$O = V_{ref} \cdot \left( \frac{R_1}{R_1 + R_2} \right) \cdot \left( \frac{R_F + R_G}{R_G} \right) \quad (6)$$

## 2.6 OSILOSCÓPIO DIGITAL

O principal propósito de um osciloscópio é demonstrar graficamente sinais elétricos conforme eles variam no tempo. A maior parte dos osciloscópios produz um gráfico de duas dimensões onde o tempo se encontra no eixo *x*, e a tensão no eixo *y* (SPARKFUN, 2016). Um exemplo de tela de um osciloscópio é mostrado na Figura 12.

O instrumento possui controles que permitem ajustar a escala tanto de tempo quanto de tensão. Há também a função *trigger* que ajuda a estabilizar o sinal na tela. Exemplos de



**Figura 12: Tela de um Osciloscópio**

**Fonte: Extraído de (SPARKFUN, 2016)**

grandezas que o osciloscópio consegue medir são frequência, período, *duty cycle*, tempo de subida e descida, amplitude da onda e valores máximo, mínimo e médio de tensão (SPARKFUN, 2016). Além de mostrar a forma de uma onda, com o auxílio do osciloscópio também é possível identificar a presença de ruído no circuito.

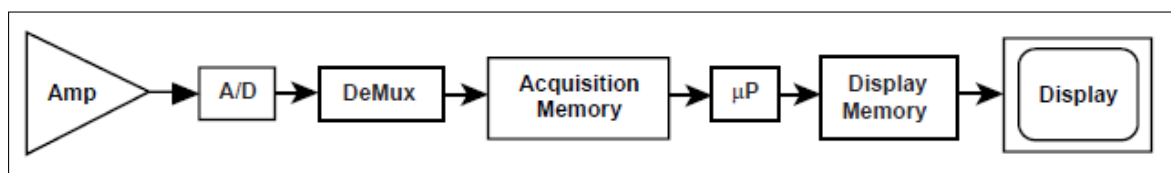
Algumas características são fundamentais na análise de um osciloscópio (SPARKFUN, 2016; TEKTRONIX, 2000):

- **Largura de banda:** o quão rápido um osciloscópio consegue identificar uma mudança de sinal. Especifica o limite de frequência que pode ser confiavelmente lido pelo aparelho;
- **Quantidade de canais:** quantidade de sinais que podem ser lidos e mostrados simultaneamente;
- **Taxa de amostragem:** define quantas vezes por segundo o sinal é lido;
- **Tempo de subida:** determina o pulso de subida do sinal mais rápido que pode ser lido;
- **Tensão máxima de entrada:** a maior tensão que pode ser lida pelo aparelho sem danificá-lo;
- **Resolução:** representa o quão precisamente a tensão de entrada pode ser lida. Este valor pode mudar conforme a escala vertical é ajustada;

- **Sensibilidade vertical:** representa os valores mínimo e máximo da escala de tensão, listado como volts por divisão;
- **Impedância de entrada:** quando a frequência do sinal é muito alta, mesmo uma pequena impedância adicionada ao circuito pode afetar o sinal. Todo osciloscópio adicionará uma certa impedância ao circuito que está lendo, chamada impedância de entrada, que pode ser representada por uma grande resistência (na ordem de  $M\Omega$ ) em paralelo com uma pequena capacitância (na ordem de pF).

O sistema de *trigger* de um osciloscópio é dedicado à estabilizar e focar a imagem da onda. O *trigger* indica ao osciloscópio em quais partes do sinal a medida deve ser tomada. Em casos de sinais periódicos, o *trigger* pode ser manipulado para manter a imagem do *display* estática (TEKTRONIX, 2000; SPARKFUN, 2016).

O funcionamento de um osciloscópio digital inicia-se com a amplificação ou atenuação vertical (tensão) do sinal lido para que este fique em conformidade com a faixa de tensão aceita pelo conversor AD. O próximo passo é a conversão do sinal pelo ADC, no qual amostras do sinal são tomadas em pontos discretos no tempo e suas tensões são convertidas para valores digitais chamados *sample points*. A taxa com que os dados são obtidos é chamada de taxa de amostragem. Os pontos amostrados passam então por um demultiplexador, responsável por conectar as múltiplas entradas à memória de aquisição, onde são armazenados para posterior processamento. O microprocessador lê os dados da memória, realiza os processamentos necessários e envia os resultados para a memória do *display*, onde finalmente a imagem é exibida (TEKTRONIX, 2000). O diagrama de blocos deste processo é mostrado na Figura 13.



**Figura 13: Diagrama de Blocos de um Osciloscópio Digital**

**Fonte: Extraído de (TEKTRONIX, 2000)**

## 2.7 TRABALHOS RELACIONADOS

Alguns projetos já desenvolvidos apresentam relação com esta proposta. Karim (2014) projetou um osciloscópio de baixo custo com um único canal baseado em Arduino e *display* LCD gráfico. Neste projeto são feitas leituras do sinal e armazenadas na memória RAM até

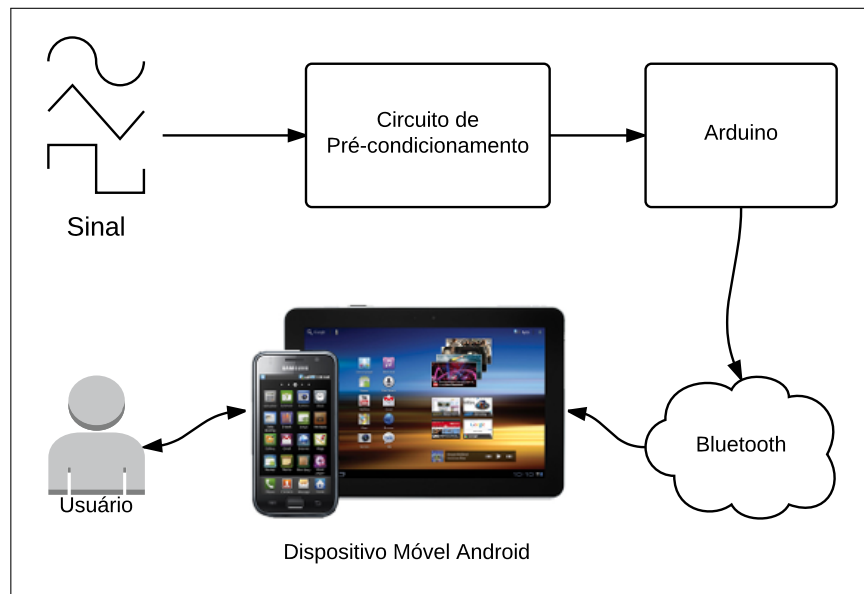
que esta esteja cheia, momento em que os dados são mostrados no *display*. O circuito de condicionamento do sinal utilizado é capaz de receber tensões alternadas (AC) e contínuas (DC) de até 100 volts e convertê-las em tensões entre 0 e 5 volts em corrente contínua (VDC). O autor não implementa a função *trigger* em hardware, mas sim na programação do microcontrolador. Esta solução foi capaz de ler, sem distorção, sinais de frequências entre 10 Hz e 7.7 kHz com uma taxa de amostragem de 10 mil amostras por segundo. O autor destaca que o ADC contido no Atmega328P não é rápido o suficiente para trabalhar com sinais de frequências mais altas.

Em Yi (2010), foi utilizada a placa de som de um computador para a construção de um osciloscópio de dois canais. Com uma taxa de amostragem de 44,1 kHz e 16 bits de resolução, o projeto foi capaz de interpretar sinais entre 10 Hz e 20 kHz. A interface gráfica do osciloscópio foi programada em LabVIEW, contendo todo o processamento do sinal, enquanto a placa de som ficou responsável apenas pela aquisição dos valores por meio da conversão AD.

No artigo intitulado *Open Source Oscilloscope for Hobby Users*, os autores Niculescu e Lita (2015) afirmam que a taxa de amostragem de 2 MSPS (*mega samples per second*) com 8 bits de resolução é considerada suficiente para utilização em hobbies. Neste trabalho, um osciloscópio de custo menor que 10 dólares é construído baseado em um microcontrolador PIC o qual possui um ADC com taxa de aquisição de 2 MSPS. A exibição dos dados é feita em um computador através de um programa desenvolvido em C# e a comunicação se dá por USB. O controle dos parâmetros de condicionamento do sinal é realizado através da interface gráfica do programa, sendo interpretado pelo microcontrolador que seleciona a atenuação ou ganho adequado por um multiplexador analógico.

### 3 DESENVOLVIMENTO DO OSCILOSCÓPIO

Um diagrama representando a arquitetura da solução proposta pode ser visto na Figura 14. Nesta, o sinal de entrada passa pelo circuito de pré-condicionamento, onde é tratado para que atenda à faixa de tensão aceita pelo conversor AD; no Arduino ocorrem as etapas de conversão AD e envio ao dispositivo móvel por meio de conexão Bluetooth; no dispositivo móvel os dados recebidos são processados e exibidos ao usuário de acordo com as opções escolhidas pelo mesmo.



**Figura 14: Diagrama da Arquitetura Proposta**

**Fonte: Autoria própria**

Nesta seção são descritas as atividades que foram desenvolvidas a fim de alcançar o objetivo deste trabalho, sendo subdividida na ordem de execução: Seção 3.1 - comunicação entre o Arduino e o dispositivo móvel; Seção 3.2 - *firmware* para o Arduino; Seção 3.3 - aplicação Android; Seção 3.4 - circuito de aquisição do sinal; e por fim, na Seção 3.5 o circuito de condicionamento do sinal.

#### 3.1 COMUNICAÇÃO ENTRE ARDUINO E DISPOSITIVO MÓVEL

No desenvolvimento da comunicação entre o Arduino e o dispositivo móvel, primeiramente foi necessário selecionar o meio de comunicação mais adequado ao projeto tendo em vista que um osciloscópio deve funcionar em tempo real, portanto altas velocidades de comunicação e processamento são requeridas. A comunicação entre os dispositivos foi esco-



lhida como primeira etapa de desenvolvimento pois, na ocorrência de problemas ou impedimentos nesta fase, todo o restante do projeto seria inviável.

Assim, foi implementado um protótipo para verificar a correta transmissão dos dados entre as duas partes. Verificado o funcionamento adequado, foi elaborado um protocolo para a transmissão dos dados referentes ao sinal analisado pelo osciloscópio.

### 3.1.1 SELEÇÃO DO MEIO DE COMUNICAÇÃO

Os meios de comunicação entre o Arduino e o dispositivo móvel disponíveis são: Bluetooth por meio de um módulo para o Arduino; Wi-Fi por meio de um *shield*, ou nativamente para o Arduino YUN; e USB podendo ser de duas formas, na primeira o dispositivo móvel atuando como *host* por meio de um adaptador USB OTG, e a segunda o Arduino YUN atuando como *host* por meio de sua porta USB nativa.

Cada opção foi estudada e verificada sua viabilidade a fim de encontrar a melhor solução para o projeto. Dado que o intuito principal era que o projeto fosse portátil, inicialmente as conexões sem fio Bluetooth e Wi-Fi foram preferidas. Os pontos analisados nas duas opções de conexão sem fio foram custo, velocidade de comunicação, popularidade entre os praticantes de DIY e facilidade de implementação.

No quesito custo, um módulo Bluetooth para Arduino custa aproximadamente U\$ 6, enquanto um *shield* Wi-Fi custa por volta de U\$ 15. Ambos podem ser utilizados com qualquer modelo de Arduino, que custam a partir de U\$ 4 na versão Nano genérica. Uma outra possibilidade é a utilização do Arduino YUN que já possui Wi-Fi integrado e custa em média U\$ 75. Os valores considerados são em dólar FOB (*Free on Board*), que representa o custo do produto desconsiderando transporte e encargos.

Todas as opções utilizam a interface serial UART para comunicação com o microcontrolador, estando então a velocidade máxima de comunicação sem fio inicialmente limitada pela velocidade máxima de comunicação serial do Arduino, sendo esta 2 Mbps para os microcontroladores que operam a um *clock* de 16 MHz. Este fato elimina a vantagem de velocidade que a comunicação Wi-Fi, em suas versões 802.11b (11 Mbps) ou 802.11g (54 Mbps), teria sobre o Bluetooth v2.0+EDR (2.1 Mbps), mais comumente encontrado como módulos para Arduino.

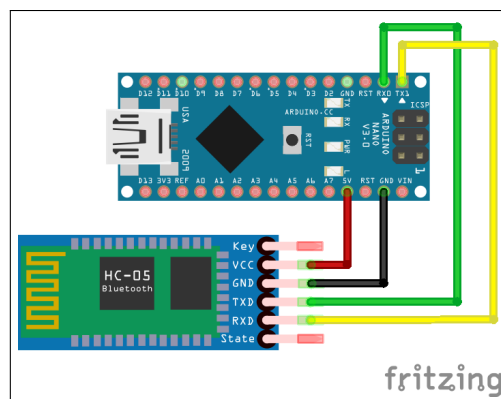
Caso a implementação ocorresse utilizando o Wi-Fi integrado do Arduino YUN, seria necessário o desenvolvimento de um script em Python que rodasse no núcleo Linux do mesmo e trabalhasse como ponte entre a comunicação serial com o microcontrolador e a comunicação Wi-Fi com o dispositivo móvel, além da implementação do *firmware* para o microcontrolador,

e da aplicação para o Android. Portanto, esta opção foi descartada devido ao seu alto custo perante as demais, e à maior complexidade de implementação.

Restando as opções de utilização de um módulo Bluetooth ou um *shield* Wi-Fi, o primeiro foi escolhido por ser mais barato, e também ser mais popular entre os *hobbyistas*, estando presente na maioria dos kits Arduino para iniciantes.

### 3.1.2 IMPLEMENTAÇÃO DO PROTÓTIPO

Na implementação do protótipo foi utilizado inicialmente o Arduino Mega 2560 pois o mesmo possui mais de uma porta serial UART, sendo possível a comunicação simultânea com o módulo Bluetooth e com o computador via USB para *debug*. A fim de minimizar o custo final do projeto, posteriormente o sistema foi transferido para um Arduino Nano genérico, que possui apenas uma porta serial UART utilizada para comunicação com o módulo Bluetooth. Foi então verificada sua capacidade de executar o *firmware* de forma equivalente ao Mega, excluindo-se apenas a comunicação USB com o computador. A Figura 15 ilustra a ligação entre o módulo Bluetooth HC-05 e o Arduino Nano, na qual é necessário apenas a alimentação elétrica e o par de fios para a comunicação serial UART.



**Figura 15: Ligação Entre Arduino e Módulo Bluetooth**

**Fonte: Autoria própria**

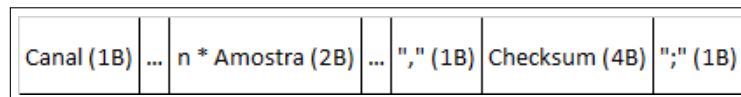
Para que a correta comunicação entre o microcontrolador e o módulo Bluetooth ocorra, ambos precisam estar configurados para se comunicar com a mesma taxa de transmissão de dados. As maiores opções disponíveis no módulo Bluetooth HC-05 são 230.400, 460.800, 921.600 e 1.382.400 bps. Já para o Arduino, por operar com um clock de 16 MHz as taxas de transmissão configuráveis são 230.400, 250k, 500k, 1M e 2M bps. Desta forma, a maior velocidade comum entre ambos os dispositivos é 230.400 bps, sendo esta a velocidade utilizada para a implementação do projeto.

A comunicação via Bluetooth do módulo utilizado é implementada de forma transparente, ou seja, toda informação enviada ao módulo via serial UART é transmitida para o dispositivo pareado; e toda informação recebida do dispositivo pareado é enviada ao microcontrolador pela serial UART. Desta forma, tem-se a impressão de que o microcontrolador está se comunicando diretamente com um dispositivo conectado em sua porta serial, e não com um dispositivo pareado por meio de uma conexão sem fio.

Esta forma de implementação facilita a utilização do módulo, porém não garante a integridade dos dados transmitidos, podendo ocorrer perda de pacotes. Por este motivo, foi desenvolvido um protocolo de comunicação que possibilita a conferência da integridade dos dados recebidos pelo dispositivo móvel.

### 3.1.3 PROTOCOLO DE COMUNICAÇÃO

Dada a relativamente baixa velocidade de comunicação conseguida entre o microcontrolador e o módulo Bluetooth, um protocolo de comunicação foi desenvolvido a fim de maximizar a quantidade de dados transmitida e garantir a integridade da informação. Este protocolo também permite a utilização de múltiplos canais do osciloscópio, ou seja, a exibição de múltiplos sinais na tela do mesmo. A Figura 16 representa a estrutura do protocolo implementado.



**Figura 16: Estrutura do Protocolo Implementado**

**Fonte: Autoria própria**

O primeiro byte do protocolo representa o canal correspondente às amostras seguintes, então são enviadas  $n$  amostras do sinal, cada uma com 2 bytes. Um caractere “,” define a transição entre os dados de amostras e os 4 bytes correspondentes ao *checksum*. Um caractere “;” simboliza o fim da série de dados.

Com base neste protocolo é possível verificar a equivalência entre o *checksum* calculado na origem dos dados e na recepção. Sendo ambos os valores iguais, nenhuma das  $n$  amostras teve seu valor corrompido. O cálculo do *checksum* é realizado somando-se os valores de todas as amostras contidas no protocolo.

A quantidade  $n$  de amostras contidas no protocolo pode ser definida pelo usuário por meio da alteração da constante `bufferSize` no código do *firmware* do microcontrolador. Vale

ressaltar que para valores de  $n$  grandes haverá menor perda de taxa de transmissão proveniente dos metadados do protocolo (*overhead*). Entretanto, caso ocorra a divergência na verificação do *checksum*, indicando falha na integridade dos dados, uma grande quantidade de amostras será descartada, causando maior deformação no sinal exibido. Nos testes foram utilizados valores entre 100 e 200 amostras por protocolo.

## 3.2 FIRMWARE PARA ARDUINO

O Arduino tem duas funcionalidades principais neste projeto: realizar a leitura de valores analógicos e enviar estes valores ao dispositivo móvel. Entre as duas operações alguns processamentos também são necessários.

### 3.2.1 TRANSMISSÃO SERIAL

O método de transmissão serial mais utilizado em projetos Arduino é o `Serial.print()`, que envia dados na formatação ASCII, tornando mais fácil a interpretação humana dos dados em terminais como o *Serial Monitor* da IDE Arduino e o PuTTY. Porém, ao enviar um número por esta função, cada dígito é convertido em sua representação por caractere, que ocupa um byte. No caso do envio do valor 1023, por exemplo, o dado seria convertido em quatro caracteres, totalizando 4 bytes contra apenas 2 bytes do valor em binário.

Visando a otimização da quantidade de dados transmitida, foi utilizada a função `Serial.write()`, que envia a representação binária do valor. Caso uma variável maior que um byte seja passada como parâmetro, apenas o byte menos significativo é enviado. Para que o valor completo seja enviado, é necessário então utilizar o operador *bitshift* “>>”, fazendo com que os bits mais significativos ocupem a posição dos menos significativos.

Na implementação realizada, foram enviados os bytes na ordem dos mais significativos para os menos significativos. A Figura 17 representa a sequência de *bitshifts* realizados para o envio do valor de *checksum*, uma variável `unsigned long` de 32 bits. O mesmo foi dividido em quatro etapas: *shift* de 24, 16 e 8 bits, e por último o byte menos significativo. Para o tipo `unsigned int` de 16 bits utilizado para as amostras do sinal, foi enviado primeiro o valor com um *shift* de 8 bits, depois os 8 bits menos significativos restantes.

A transmissão serial UART ocorre de byte em byte, tendo para cada byte um *start bit* e um *stop bit*, logo, para cada byte de informação enviado, são utilizados 10 bits da taxa de transmissão. Isto deve ser levado em consideração ao calcular a quantidade máxima de amostras alcançável com a taxa de transmissão configurada.

```
Serial.write(b.cks >> 24);  
Serial.write(b.cks >> 16);  
Serial.write(b.cks >> 8);  
Serial.write(b.cks);
```

**Figura 17: Bitshift Realizado no Envio de uma Variável de 32 bits**

**Fonte: Autoria própria**

Considerando o envio de uma amostra do sinal, 16 bits, este é dividido no envio de 2 bytes subsequentes. A transferência desta amostra ocupa então 20 bits da taxa de transmissão do serial devido ao *start* e *stop* bits. Para a taxa de 230.400 bps alcançável, o tempo teórico necessário para transmitir uma amostra é de 86,8 microssegundos.

Foram realizadas medidas empíricas para esta transmissão e o tempo resultante foi de aproximadamente 100 microssegundos. O aumento no valor ocorre pois não apenas a transmissão é levada em conta, mas também a chamada da função de transmissão, o *bit shift*, a bufferização dos dados para transmissão, entre outros processos internos ao microcontrolador.

### 3.2.2 CONVERSÃO AD

O conversor AD do Arduino possui resolução de 10 bits, portanto o resultado da conversão de uma tensão entre 0 e 5 volts retorna valores entre 0 e 1023, sendo o valor 0 para 0 volt e 1023 para 5 volts. Não há um tipo de dado com tamanho 10 bits na plataforma Arduino, e mesmo que houvesse, o microcontrolador apresenta uma arquitetura de 8 bits, sendo necessário a utilização de dois registradores deste tamanho para o armazenamento de um dado de 10 bits, desperdiçando 6 bits. Desta forma, o tipo de dado utilizado para o armazenamento e processamento do resultado da conversão AD foi o `unsigned int`, que possui tamanho 16 bits na plataforma Arduino.

A função disponível na plataforma Arduino para leitura de valores analógicos é a `analogRead()`. Esta é bloqueante, e a conversão AD é lenta em relação à velocidade de processamento do microcontrolador, logo o programa fica parado aguardando o resultado da conversão enquanto poderia estar executando outras funções. Em sua configuração padrão, o Arduino leva aproximadamente 112 microssegundos para realizar a conversão AD.

A plataforma Arduino permite que os registradores dos microcontroladores utilizados em suas placas sejam acessados, logo, é possível realizar uma programação de forma otimizada, modificando os parâmetros mais internos do microcontrolador, assim como na programação em AVR-C.

Os microcontroladores ATmega 328, 2560 e 32u4, utilizados nos Arduinos Uno/Nano, Mega e YUN, respectivamente, possuem registradores que configuram o valor de *prescaler* utilizado para gerar o *clock* do conversor AD (mais lento) a partir do *clock* principal do microcontrolador. O valor padrão utilizado pela plataforma Arduino é 128, gerando um *clock* para o conversor AD de 125 KHz (16 MHz/128). O *clock* máximo suportado pelo conversor destes microcontroladores é 1 MHz, indicando que o menor valor de *prescaler* utilizável é 16. No caso da utilização deste valor, uma conversão AD consumiria aproximadamente 15 microssegundos, entretanto, este aumento na velocidade resulta em uma diminuição na resolução de leitura, e maior suscetibilidade à ruídos.

### 3.2.3 IMPLEMENTAÇÃO ASSÍNCRONA

Tendo em vista que o envio de uma amostra via serial para o módulo Bluetooth leva 100 microssegundos, a conversão desta amostra leva 112 microssegundos usando *prescaler* = 128 ou 15 microssegundos usando *prescaler* = 16, o tempo mínimo necessário para conversão e envio de uma amostra seria de 115 microssegundos utilizando a conversão mais rápida, ou 212 microssegundos utilizando a conversão mais precisa.

Foi então verificada a possibilidade de conversão AD assíncrona, na qual é requisitada a conversão ao periférico e, enquanto a mesma é realizada o microcontrolador fica livre para executar outras tarefas, neste caso enviar dados via serial. Como a plataforma Arduino permite acesso aos registradores do microcontrolador, é possível utilizar este método.

Para utilização da leitura analógica assíncrona primeiramente é necessário efetuar algumas configurações que antes eram realizadas pela função `analogRead()`. São elas: especificação da tensão de referência para o conversor AD (5 volts); ajuste à direita do resultado da conversão nos registradores (2 bits mais significativos no registrador mais significativo, os 8 bits restantes no registrador menos significativo); e seleção do pino de entrada do sinal (A0). Estas configurações são armazenadas no registrador ADMUX do microcontrolador. Também é necessário ativar o conversor AD atribuindo o valor 1 ao bit ADEN do registrador ADCSRA. Estas configurações são executadas apenas uma vez na inicialização do microcontrolador, dentro da função `setup()` do Arduino.

Estando as configurações efetuadas, o funcionamento em loop se dá da seguinte forma:

- O bit ADSC do registrador ADCSRA é setado, iniciando a conversão;
- Caso seja a primeira amostra do protocolo, o byte correspondente ao canal do osciloscópio é enviado via serial ao módulo Bluetooth. Caso contrário, a amostra lida an-

teriormente é enviada;

- O programa aguarda até que o bit ADSC retorne ao valor zero, indicando o fim da conversão;
- O valor convertido é copiado dos registradores ADCL e ADCH para o vetor que armazena as  $n$  amostras do protocolo;
- O valor do *checksum* é incrementado na quantidade do valor lido;
- Caso a  $n$ -ésima amostra tenha sido lida, seu valor é enviado, seguido do caractere “;”, dos 4 bytes do *checksum* e do caractere “;” indicando o fim do protocolo.

Como o envio de uma amostra leva 100 microssegundos, foi preferido utilizar o valor de *prescaler* = 128, que resulta em 112 microssegundos para conclusão da conversão e em melhor precisão. Desta forma, o envio de uma amostra lida anteriormente cabe completamente dentro do espaço de tempo entre o início e o fim da conversão.

Este método também possibilitou obter amostragens em intervalos regulares de tempo, facilitando a reconstrução do sinal de forma gráfica no dispositivo móvel. Com amostras sendo realizadas e enviadas a cada 112 microssegundos é possível transmitir a quantia de 8.928 amostras por segundo.

Inicialmente era prevista a implementação de dois canais para o osciloscópio, porém, para se obter maior frequência máxima do sinal analisado foi utilizado apenas um canal devido a velocidade limitada da comunicação serial com o Bluetooth. A utilização de dois canais pode ser facilmente implementada caso não seja necessária a representação de sinais rápidos na utilização final do projeto, pois o protocolo implementado define em seu primeiro byte o canal correspondente à série de amostras. Utilizando um tamanho de buffer  $n$  pequeno e alternando a leitura e envio de amostras de cada canal é possível obter uma representação gráfica de dois sinais simultaneamente. Este método é análogo ao funcionamento *multithread* concorrente, no qual operações sequenciais são comutadas rapidamente, dando a impressão de estarem acontecendo em paralelo.

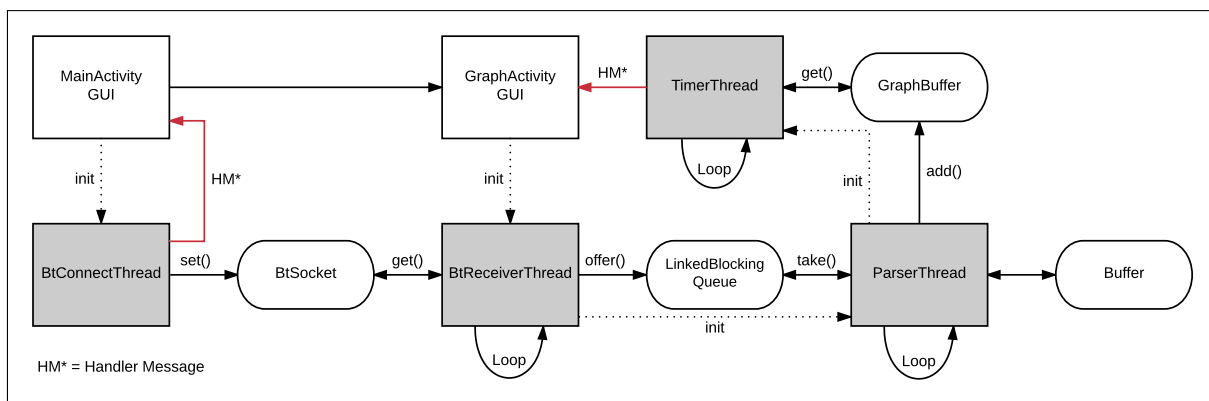
O código fonte resultante da implementação do *firmware* para Arduino foi disponibilizado no repositório GitHub sob o link <https://github.com/rhuanecc/Oscilino>.

### 3.3 APLICAÇÃO ANDROID

Foi desenvolvida uma aplicação que possui como função principal a exibição da forma de onda representante do sinal na tela do dispositivo móvel. Esta aplicação recebe os dados do microcontrolador no formato do protocolo implementado, faz o tratamento necessário e os exibe em forma de gráfico 2D, da mesma forma em que são mostrados em osciloscópios profissionais.

Para a implementação do gráfico foi utilizada a biblioteca Android GraphView (GRAPHVIEW, 2016). Esta biblioteca é *open-source*, disponível no repositório GitHub, podendo ser incluída no projeto tanto por um arquivo .jar, quanto como dependência no arquivo Gradle. Na última opção, os arquivos são baixados automaticamente pelo Android Studio a partir do repositório Maven.

A aplicação implementada é composta de duas interfaces gráficas, quatro *threads* e quatro estruturas de dados. A Figura 18 mostra a relação entre estes elementos. Retângulos com fundo branco são interfaces gráficas, retângulos com fundo cinza são *threads* e círculos são estruturas de dados utilizadas na comunicação entre as *threads*.



**Figura 18: Diagrama da Aplicação Implementada**

**Fonte: Autoria própria**

#### 3.3.1 INICIALIZAÇÃO DA APLICAÇÃO

A interface gráfica **MainActivity** é mostrada logo que a aplicação é aberta. Esta foi construída para disponibilizar a configuração de alguns parâmetros, tais como quantidade de pontos utilizada na exibição do gráfico, que pode ser reduzida para dispositivos com menor poder de processamento; e opção de manter tela do dispositivo sempre acesa. A mesma também possui um botão que inicia a conexão Bluetooth com o Arduino.



Ao pressionar o botão de conexão, é solicitada ao usuário a ativação do Bluetooth caso o mesmo esteja desativado. Então é exibida a lista de dispositivos pareados para que o usuário escolha com qual deseja se conectar.

A *thread* `BtConnectThread` é iniciada recebendo como parâmetro o endereço MAC do dispositivo escolhido. Esta *thread* tenta realizar a conexão e, caso consiga, o *socket* resultante é setado no atributo estático da classe `BtSocket`. Uma mensagem é enviada de volta à `MainActivity` indicando o sucesso ou falha na tentativa de conexão, e a *thread* é encerrada.

Um Handler implementado na `MainActivity` analisa a mensagem recebida da *thread* de conexão e em caso de sucesso abre a interface gráfica que contém o gráfico, `GraphActivity`. Caso contrário, exibe uma mensagem ao usuário indicando falha na tentativa de conexão.

### 3.3.2 INTERFACE GRÁFICA

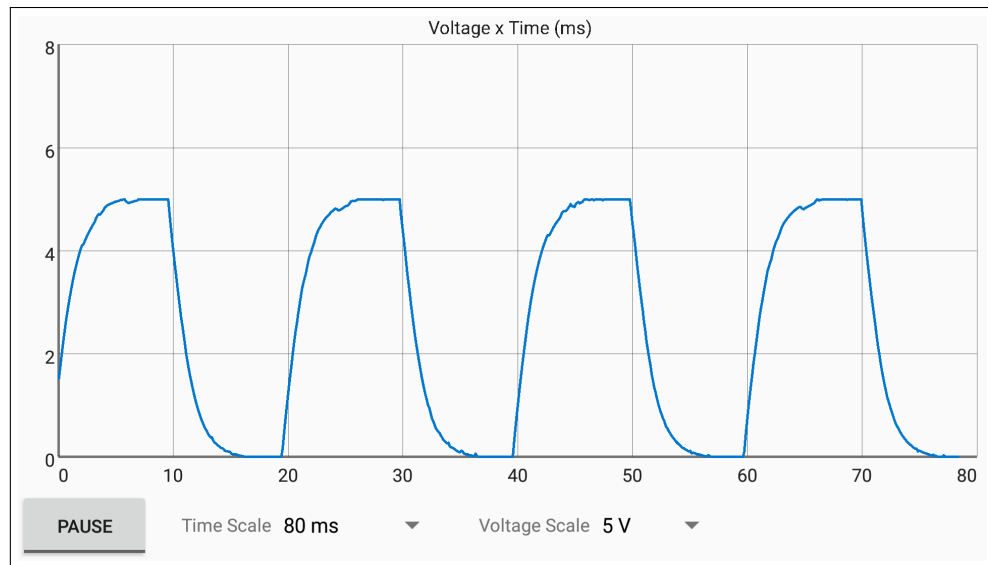
A interface gráfica `GraphActivity` contém os componentes necessários para a exibição e manipulação do gráfico, sendo estes: um componente da biblioteca `GraphView`, responsável pela exibição do gráfico de acordo com os parâmetros especificados; um botão pause, que permite ao usuário congelar a exibição do sinal na tela para melhor visualização; um *spinner* para seleção de escala de tempo; e um *spinner* para seleção de escala de tensão. Os componentes desta interface gráfica são mostrados na Figura 19.

Nesta classe também são definidas as constantes utilizadas na aplicação, tais como multiplicador da escala de tempo, multiplicador da escala de tensão, intervalo de atualização da tela e quantidade de pontos utilizados na construção do gráfico.

Foi implementada a função cursor, na qual ao usuário tocar sobre determinado ponto do gráfico, as informações exatas de tempo e tensão são mostradas, possibilitando leituras mais precisas do sinal. Esta função é mais facilmente utilizada em conjunto com a função pause, pois com a imagem do sinal congelada é mais fácil definir quais pontos se deseja analisar.

Dado que a aplicação recebe amostras do sinal em intervalos fixos de tempo, e a quantidade de amostras exibida no gráfico é limitada, por consequência, o período de tempo do sinal exibido também é limitado. A fim de tornar a aplicação mais flexível e mais próxima de um osciloscópio real, foi implementada a funcionalidade de escala de tempo, que permite ao usuário aumentar ou diminuir o período de exibição do sinal na tela do dispositivo.

Além da alteração da escala de tempo pelo *spinner*, também foi implementada a fun-



**Figura 19: Interface Gráfica GraphActivity**

**Fonte: Autoria própria**

cionalidade zoom usando manipulação do tipo pinça (dois dedos), assim como utilizada em imagens ou páginas web. Desta forma, o usuário pode aumentar ou diminuir a escala de tempo arbitrariamente. Os rótulos referentes ao eixo do tempo são atualizados de acordo com o zoom aplicado, possibilitando a visualização mais detalhada de um determinado intervalo do sinal. Ao aplicar o zoom, o *scroll* no eixo horizontal é habilitado, viabilizando por exemplo a visualização de todo o período de um sinal congelado, porém com mais detalhes.

De forma similar, a escala de tensão permite ao usuário especificar para a aplicação qual a amplificação ou atenuação utilizada no circuito de pré-condicionamento do sinal. Ao especificar no *spinner* esta proporção, a *thread* `TimerThread` realiza a manipulação necessária para converter o valor lido pelo Arduino, entre 0 e 5 volts, para o valor real.

### 3.3.3 RECEBIMENTO E PROCESSAMENTO DOS DADOS

Ao ser iniciada, a interface gráfica `GraphActivity` instancia e inicializa uma *thread* responsável pela recepção dos dados pelo Bluetooth, a `BtReceiverThread`. Esta *thread* recupera o *socket* da conexão Bluetooth gerado na fase de inicialização, contido na classe `BtSocket`, obtendo a partir dele os *streams* de *input* e *output* para comunicação com o micro-controlador.

A `BtReceiverThread` instancia e inicializa uma outra *thread*, `ParserThread`, para que esta efetue o processamento dos dados. Com uma *thread* dedicada apenas ao rece-

bimento dos dados e outra para o processamento, evita-se a possível perda de dados devido à ocupação da aplicação em uma fase de processamento no caso de tarefas sequenciais. Os dados recebidos são adicionados em uma fila da classe `LinkedBlockingQueue`, que é *thread-safe* e bloqueante (uma operação de leitura aguarda até que existam dados na fila), sendo desta forma adequada para utilização na comunicação unidirecional entre *threads*.

Enquanto a `BtReceiverThread` trabalha em *loop* recebendo os dados via Bluetooth e os colocando na fila, a `ParserThread` opera também em *loop* recuperando estes dados da fila e realizando o processamento necessário. Este processamento inicia-se consumindo a fila até que o início de uma série de dados, no formato do protocolo implementado, seja encontrado. Este início é reconhecido por suceder um caractere “;”. Esta operação se faz necessária pois o microcontrolador envia os dados independente de haver um dispositivo conectado ou não. Desta forma, a conexão Bluetooth pode ser estabelecida enquanto o microcontrolador está no meio do envio de uma série de dados. Neste caso, o Android recebe apenas parte do protocolo e deve descartá-la.

A partir do início do protocolo, o primeiro byte recebido identifica o canal correspondente à série de dados subsequente. Os bytes seguintes representam, dois a dois, os valores `int` correspondentes às amostras do sinal, e são armazenados em um buffer até que seja encontrado um caractere “;”, que representa a transição entre os bytes de amostras e os correspondentes ao *checksum*. Os próximos quatro bytes são convertidos em `long` e salvos como valor de *checksum* recebido. Ao receber o caractere “;” que indica o fim do protocolo, o buffer é processado, verificando se o valor de *checksum* calculado é equivalente ao recebido, e esvaziado.

Caso não haja divergência na checagem do *checksum*, indicando que os valores recebidos são íntegros, as amostras são adicionadas ao buffer da interface gráfica, `GraphBuffer`, que armazena as  $n$  amostras mais recentes do sinal, sendo  $n$  dependente da escala de tempo escolhida pelo usuário. Se o *checksum* falhar, as amostras são descartadas tendo em vista que o reenvio destas amostras atrasaria o funcionamento do sistema que tem como intuito ser em tempo real.

Uma terceira *thread* instanciada pela `ParserThread`, a `TimerThread`, é acordada em intervalos de tempo fixos com a finalidade de converter os dados bufferizados para as escalas adequadas e enviá-los para a interface gráfica. Este intervalo de tempo é necessário para que não haja um excesso de requisições de atualização da tela, sobrecarregando o dispositivo móvel, e deve ser superior a 17 milissegundos, já que o Android trabalha com no máximo 60 atualizações de tela por segundo.

A escala de tensão é conseguida multiplicando o valor `int` recebido do microcontro-

lador pela relação tensão/nível do conversor AD, que no caso do Arduino é  $4.88mV$ , e posteriormente multiplicando pela relação de amplificação ou atenuação utilizada no circuito de pré-condicionamento e indicada pelo usuário na interface gráfica.

A escala de tempo é conseguida da seguinte forma: para exibição de intervalos de tempo maiores que  $P * T_a$ , onde  $P$  é a quantidade de pontos utilizada para construção do gráfico e  $T_a$  é o período entre cada amostra (112 us); os pontos utilizados para a construção do gráfico passam a ser utilizados de forma intercalada. Para uma escala com tamanho  $n$  vezes maior que o tempo padrão, utiliza-se um ponto, descartando-se os  $n - 1$  seguintes. Por exemplo, para se obter uma escala com o cinco vezes o tempo padrão, utiliza-se o primeiro, ponto descarta-se os quatro seguintes, utiliza-se o sexto ponto, descarta-se os quatro seguintes, e assim por diante.

No caso da exibição de intervalos de tempo menores que  $P * T_a$ , sabendo-se que não é possível reduzir o intervalo entre as amostras, é então apenas reduzida a quantidade de pontos utilizada na construção do gráfico, tornando-os visualmente mais espaçados e evidenciando os detalhes.

Após o processamento das escalas, os pontos são enviados para a interface gráfica por meio de uma *Handler Message*, que é recebida por um Handler implementado na *GraphActivity*. Este Handler identifica o canal ao qual os dados pertencem e plota o gráfico resultante, finalizando o ciclo de operação da aplicação.

Em suma, após a conexão com o microcontrolador, três *threads* permanecem sendo executadas em paralelo: *BtReceiverThread*, recebendo os dados via Bluetooth; *ParserThread*, realizando o tratamento dos dados; e *TimerThread*, convertendo as escalas e requisitando a atualização da interface gráfica.

O projeto resultante da implementação da aplicação para Android foi disponibilizado no repositório GitHub sob o link <https://github.com/rhuanec/Oscilino>.

### 3.4 CIRCUITO DE AQUISIÇÃO DO SINAL

Assumindo um sinal de acordo com a faixa de tensão suportada pelo conversor analógico-digital, entre 0 e 5 V, este periférico faz a amostragem do sinal, representando o valor analógico lido em um valor digital correspondente. O circuito de aquisição do sinal consiste em um conversor analógico-digital, podendo este ser um componente discreto, ou interno ao microcontrolador.

A taxa máxima de aquisição do conversor AD interno ao microcontrolador ultrapassa

a quantidade de amostras que pode ser transmitida pela comunicação serial conseguida com o Bluetooth. Logo, o uso de um ADC externo não oferece vantagem, além de aumentar o custo e complexidade do projeto. No caso da utilização de um ADC externo, o mesmo se comunicaria com o microcontrolador via serial UART, I2C ou SPI, dependendo do modelo escolhido. Portanto, o microcontrolador teria que ser capaz de lidar com duas comunicações seriais com alta taxa de transmissão.

Para que fosse vantajosa a utilização de um conversor AD externo, tanto para melhorar na resolução de leitura, quanto na taxa de amostragem, primeiramente seria necessário o aumento da taxa de transmissão dos dados ao dispositivo móvel. Este aumento poderia ser conseguido utilizando um módulo Bluetooth ou um *shield* Wi-Fi que suportasse maior velocidade de comunicação com o microcontrolador. Dado que a quantidade máxima de amostras enviadas ao dispositivo móvel está limitada pela velocidade de comunicação serial UART com o módulo Bluetooth utilizado, foi decidido utilizar apenas o conversor AD interno ao microcontrolador, reduzindo-se também o custo e a complexidade do projeto.

### 3.5 CIRCUITO DE PRÉ-CONDICIONAMENTO DO SINAL

O circuito de condicionamento do sinal desenvolvido efetua o tratamento do mesmo por meio de combinações entre *offset* e amplificação ou atenuação, de forma que o sinal de saída deste circuito represente o mesmo formato do sinal de entrada, porém ajustado às condições de tensão de entrada do circuito de aquisição do sinal.

Para evitar danos aos componentes do circuito de pré-condicionamento, o primeiro tratamento aplicado é a atenuação da tensão do sinal. Desta forma, para sinais com amplitude maior que 5 volts, um divisor resistivo reduz esta tensão para valores aceitáveis para o restante do circuito.

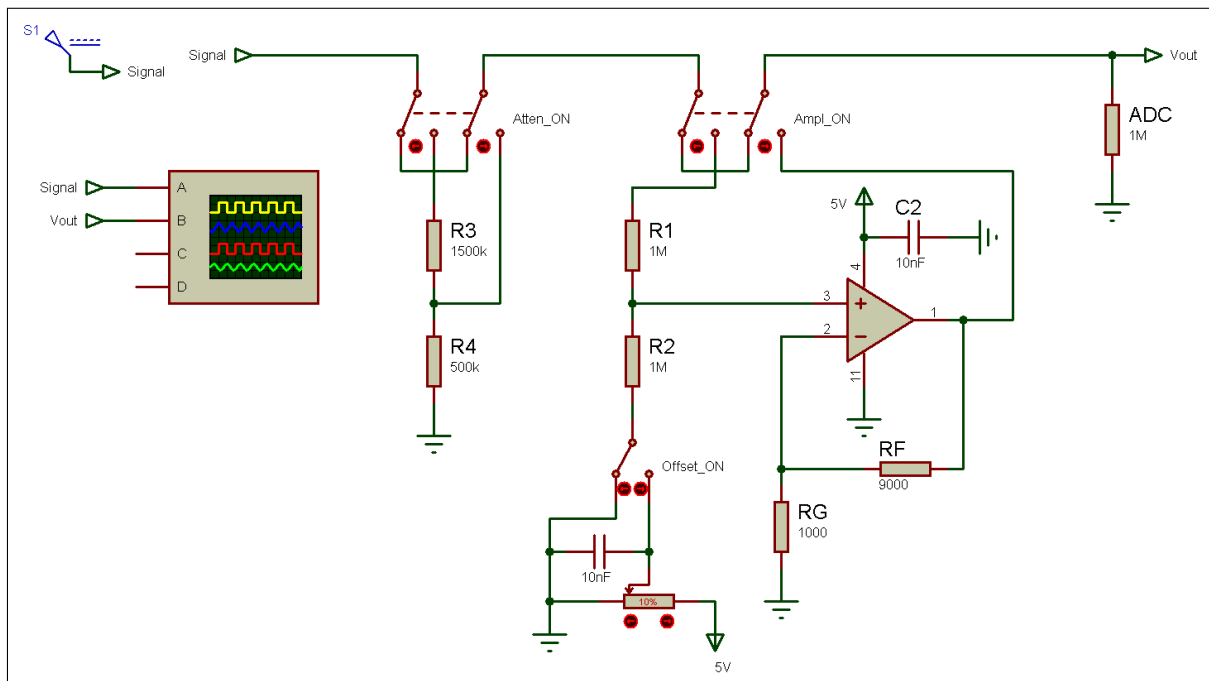
É recomendado que os resistores utilizados possuam altas resistências para que drenem neste estágio o mínimo possível de corrente do circuito sendo instrumentado. Uma outra alternativa seria adicionar um amplificador operacional entre o sinal e o divisor resistivo, para que o mesmo, devido à sua alta impedância de entrada, não interferisse no circuito sendo instrumentado. Porém, neste caso o amp-op deveria suportar a tensão do sinal e ser alimentado com uma tensão superior à esta, o que não é viável no projeto proposto pois o mesmo opera com apenas 5 volts.

Foi então construído um divisor resistivo com redução de 4x, viabilizando o osciloscópio trabalhar com sinais de até 20 volts em corrente contínua ou 20 volts pico a pico em

corrente alternada. Esta escala foi adotada por atender a faixa de tensão mais utilizada entre os *hobbyistas*, que é de até 12 volts para fontes de computadores e alguns motores DC, e até 14 volts em automóveis, por exemplo.

Caso o sinal analisado seja menor que 5 volts e o usuário deseje maior precisão na exibição do gráfico, deve-se utilizar o circuito de amplificação para que o sinal de entrada preencha o maior número possível de níveis do conversor AD. Este circuito foi simulado no *software* Proteus (LABCENTER ELECTRONICS LTD, 2016), e posteriormente testado em *proto board*, calculando-se os resistores para um ganho de 5x e opção de aplicação de *offset* de 2,5 volts, possibilitando que sinais com tensão de até 1 volt sejam exibidos com maior precisão. Neste caso, o circuito de amplificação recebe diretamente o sinal, não passando pelo circuito de atenuação.

A Figura 20 mostra o esquemático do circuito de pré-condicionamento desenvolvido. Os requisitos deste circuito são:  $R3 = 3 \cdot R4$ ;  $R1 = R2$ ;  $R_F = 9 \cdot R_G$ . É recomendada a utilização de um amplificador operacional *rail-to-rail*, permitindo que o sinal de saída se estenda entre 0 e 5 volts. Caso contrário a amplificação pode ser saturada em uma tensão abaixo de 5 volts, causando deformação no sinal de saída.



**Figura 20: Esquemático do Circuito de Pré-Condicionamento do Sinal**

**Fonte: Autoria própria**

No circuito há três chaves, uma para ligar/desligar a atenuação, outra para ligar/desligar a amplificação, e a terceira para ligar/desligar o *offset*. A atenuação é desligada fazendo um

*bypass* do sinal, ignorando o circuito correspondente. O mesmo ocorre para a amplificação. Logo, caso ambas as chaves estejam em posição desligado, como ilustrado na Figura 20, o sinal é enviado diretamente ao conversor AD do microcontrolador, condição utilizada para sinais com tensão entre 0 e 5 volts.

O *offset* é desativado reduzindo-se a tensão de referência do mesmo para zero, ou seja, aterrando o resistor R2. Ao desligar a amplificação, o *offset* deixa de ser aplicado mesmo que sua chave esteja em posição ligada, pois ambas as operações fazem parte do mesmo circuito.

Em suma, as opções de condicionamento disponíveis com o circuito proposto são: passagem direta do sinal, atenuação de 4x, amplificação de 5x sem *offset* e amplificação de 5x com *offset*. Na forma em que as chaves estão dispostas, também é possível aplicar atenuação de 4x seguido de amplificação de 5x, totalizando uma amplificação de 1,25x, que pode estar acompanhada ou não de *offset*.

No circuito de pré-condicionamento simulado foi desconsiderado o tratamento de sinais alternados que já carreguem um *offset*. Este tratamento seria conseguido utilizando-se um filtro passa-altas que rejeitaria a componente contínua, deixando passar somente a componente alternada. A frequência de corte deste filtro deve ser planejada cuidadosamente para que não haja a rejeição de sinais lentos, bem como a passagem de *offsets* variantes.

## 4 TESTES E ANÁLISE DOS RESULTADOS

Testes foram executados a fim de verificar a eficiência do canal de comunicação entre o microcontrolador e o dispositivo móvel, bem como se as formas de ondas obtidas pelo dispositivo construído são semelhantes às de um osciloscópio profissional. Os parâmetros utilizados foram 180 amostras enviadas por protocolo e 700 amostras utilizadas para construção do gráfico.

No primeiro caso, o microcontrolador foi programado para enviar valores de forma incremental, sendo assim possível identificar caso alguma informação fosse perdida. Os valores foram conferidos tanto por meio da geração de log dos dados recebidos no Android, quanto pela verificação de descontinuidades no gráfico apresentado. Em ambos os métodos não foi identificada a perda de pacotes utilizando a comunicação Bluetooth com velocidade de 230.400 bps em um ambiente residencial.

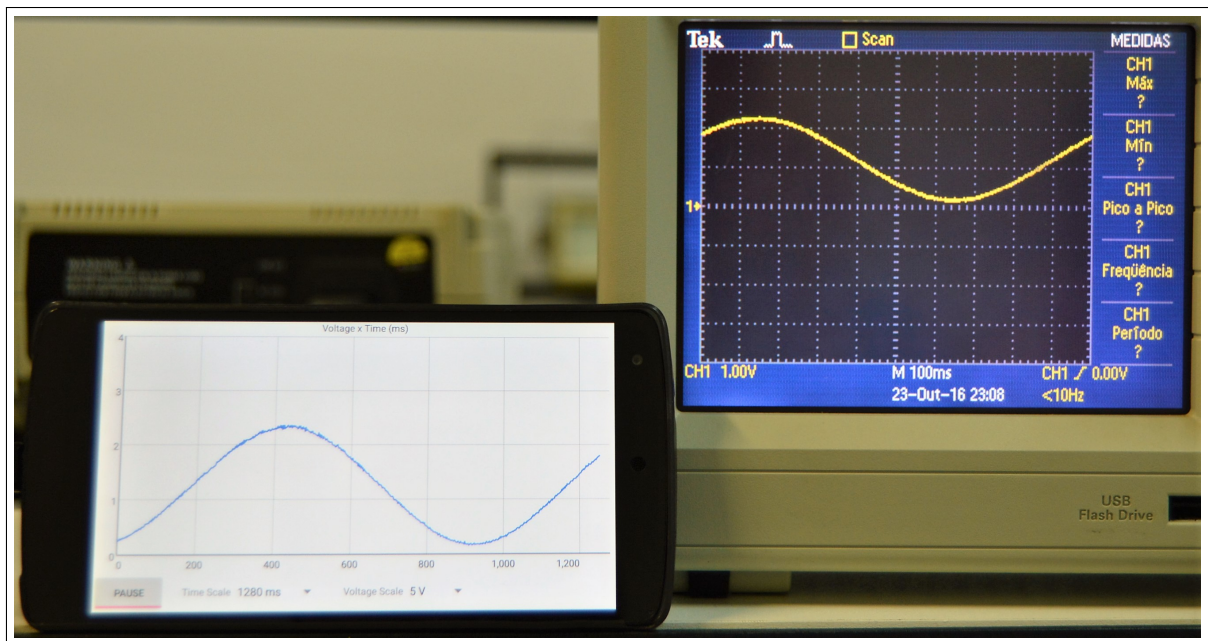
Na verificação das formas de onda obtidas, um gerador de função (Bk Precision 4017B) foi utilizado para gerar ondas de diversos formatos, frequências e amplitudes, sendo assim possível identificar os limites do projeto construído. Um osciloscópio (Tektronix TDS 2002B) foi utilizado em comparação ao projeto desenvolvido. Durante os testes foi constatado em ambos os osciloscópios que o gerador de função não estava emitindo exatamente a amplitude de onda e offset configurados. Entretanto, os valores exibidos no projeto implementado e no Tektronix foram visualmente similares.

Dado que o osciloscópio é um instrumento de representação visual de sinais, o método utilizado para avaliação da implementação foi a inspeção visual dos resultados obtidos, sendo eles forma da onda e valores máximo e mínimo de tensão.

Foram testadas as formas de onda senoidal, quadrada e triangular, nas frequências 1 Hz, 100 Hz e 500 Hz, com tensão 2 volts pico a pico ( $V_{pp}$ ) + 1 volt de *offset* ( $V_o$ ). Em todos os testes o projeto apresentou gráficos bastante similares aos do Tektronix, tanto em escala de tensão quanto de tempo. Um exemplo pode ser visto na Figura 21 onde foi testada uma onda senoidal relativamente lenta, 1 Hz.

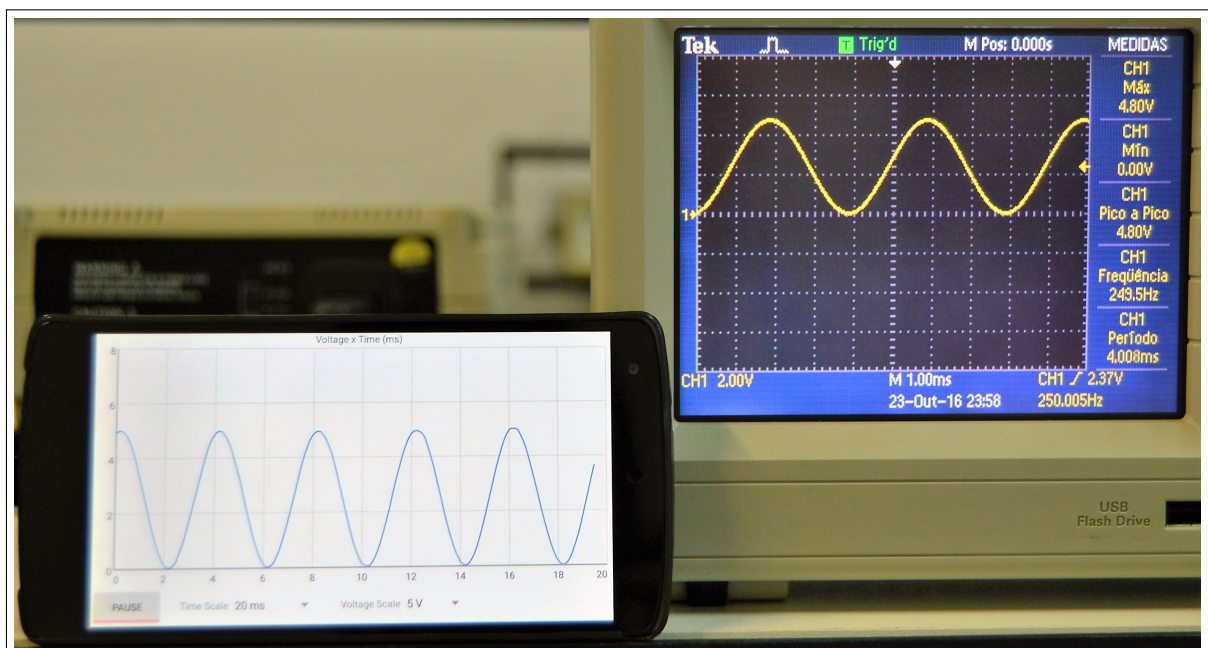
Buscando explorar os limites de tensão do projeto, foram testadas ondas quadradas, triangulares e senoidais nas frequências de 250 e 500 Hz com amplitude de 5 V<sub>pp</sub> e *offset* de 2,5 V. Os valores de tensão máximo e mínimo corresponderam aos apresentados pelo Tektronix, bem como o formato da onda. A Figura 22 ilustra uma onda senoidal com frequência de 250 Hz.





**Figura 21: Onda Senoidal de 1 Hz, 2 Vpp + 1 Vo**

**Fonte: Autoria própria**

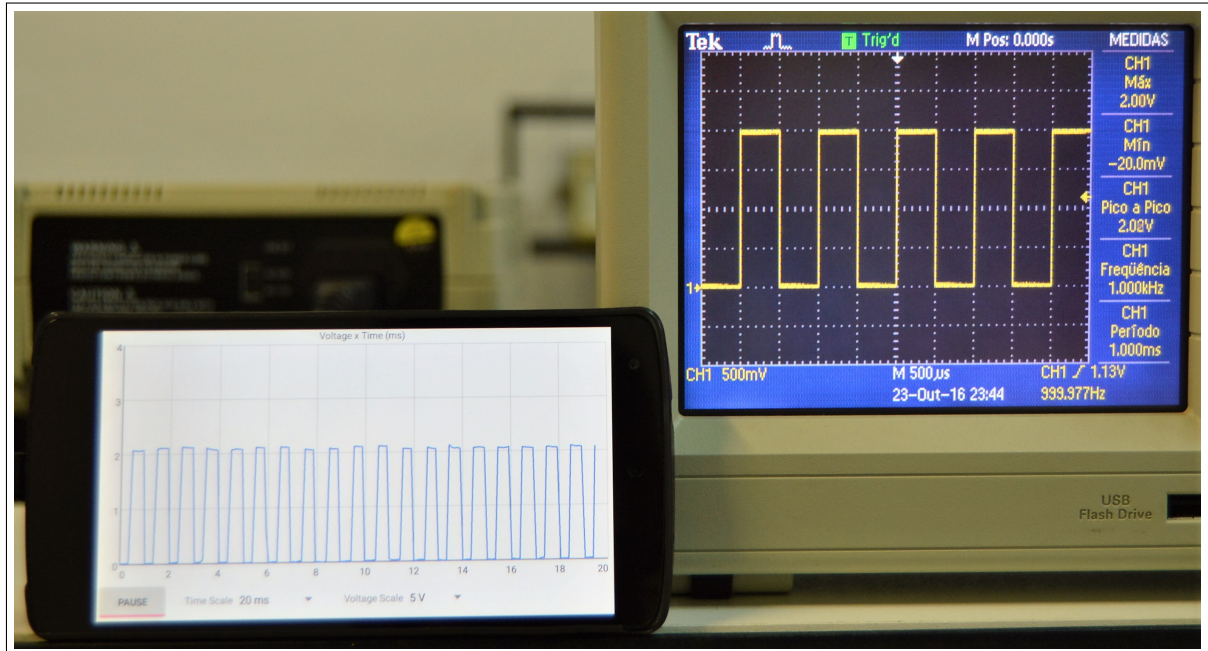


**Figura 22: Onda Senoidal de 250 Hz, 5 Vpp + 2,5 Vo**

**Fonte: Autoria própria**

Para sinais mais rápidos, como no caso da onda quadrada de 1 kHz mostrado na Figura 23, a representação gráfica do sinal sofre uma leve distorção nas transições entre os níveis alto e baixo do sinal quadrado pois a frequência de amostragem alcançada não garante a representação

da transição abrupta da onda quadrada, tornando a transição levemente inclinada. Entretanto, ainda é possível identificar o período do sinal, bem como seus valores de tensão máxima e mínima.

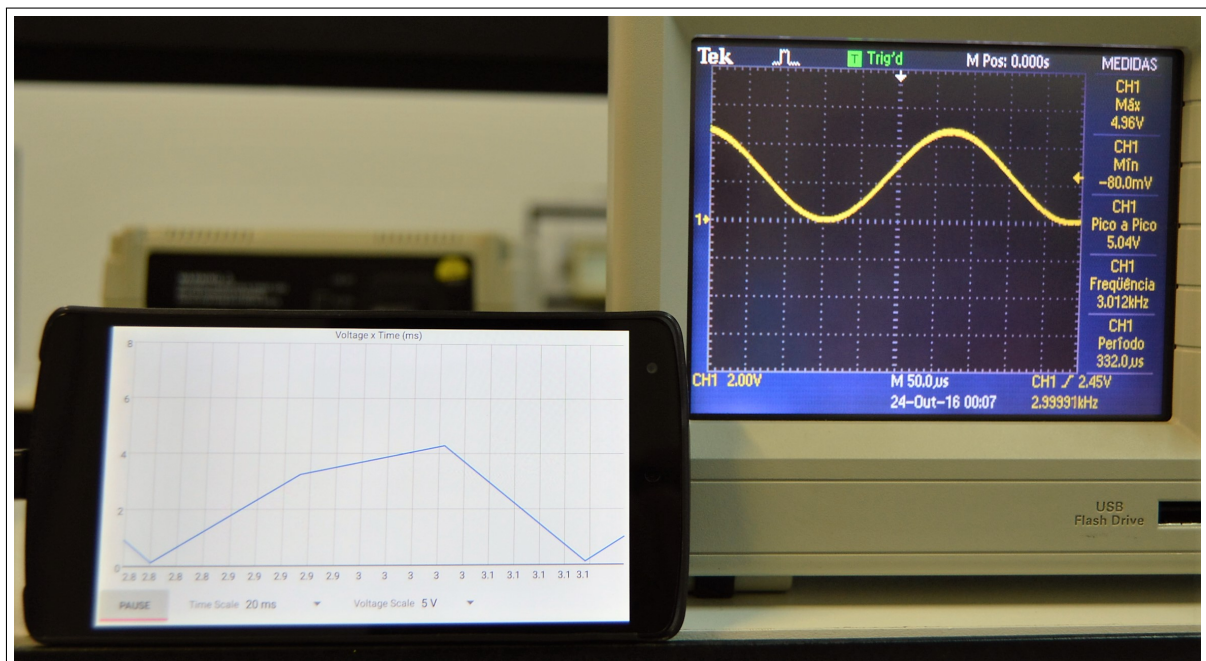


**Figura 23: Onda Quadrada de 1 kHz, 2 Vpp + 1 Vo**

**Fonte: Autoria própria**

Ainda que a taxa de amostragem seja de 8.928 amostras por segundo, e pelo teorema de Nyquist seria possível reconstituir um sinal de frequência até a metade da taxa de amostragem, para frequências acima de 2 kHz a forma da onda começa a ser perdida pois apenas quatro amostras ou menos são utilizadas para desenhar cada período do sinal. Entretanto, o período do sinal ainda pode ser estimado, como mostrado na Figura 24, na qual é exibida uma onda senoidal com frequência 3 kHz (período 0,333 ms) e amplitude 5 Vpp + 2,5 Vo.

O circuito de pré-condicionamento proposto na Seção 3.5 foi montado em *protoboard* e testado utilizando sinais de entrada com tensão contínua e verificando a saída correspondente com um multímetro (Minipa ET-1649). A atenuação de quatro vezes, a amplificação de cinco vezes e o *offset* de 2,5 V acompanhado de amplificação de cinco vezes foram obtidos conforme esperado.



**Figura 24: Onda Senoidal de 3 kHz, 5 Vpp + 2.5 Vo**

**Fonte: Autoria pr&#xpria**

## 5 CONSIDERAÇÕES FINAIS

O projeto desenvolvido envolve conhecimentos adquiridos em diversas disciplinas do curso de Engenharia de Computação, tais como Sistemas Embarcados, Programação para Dispositivos Móveis, Transmissão de Dados, Eletrônica Geral, Eletrônica Digital, Sistemas Distribuídos, Estrutura de Dados, Programação Orientada à Objetos, entre outras. Este fato tornou o projeto mais desafiador ao requerer a integração entre várias disciplinas que são ministradas de forma majoritariamente isolada. Entretanto, esta experiência se assemelha mais ao que é enfrentado no mercado de trabalho, onde o desenvolvimento de um produto requer a combinação de conhecimentos em múltiplas áreas.

O maior obstáculo encontrado no desenvolvimento deste projeto foi a velocidade de comunicação relativamente baixa conseguida entre o microcontrolador e o módulo Bluetooth, limitando a taxa de amostras do sinal que são transmitidas ao dispositivo móvel, consequentemente limitando a frequência máxima do sinal que consegue-se representar.

Com o objetivo de desenvolver uma ferramenta acessível aos praticantes de DIY, que muitas vezes têm seu primeiro contato com sistemas embarcados e eletrônica através da plataforma Arduino devido ao baixo custo e as facilidades oferecidas pela mesma, o projeto foi disponibilizado de forma *open-source* no repositório GitHub, visando permitir que *hobbyistas* usufruam do mesmo e possivelmente aperfeiçoem-no de acordo com suas necessidades específicas.

Dado o objetivo de construir uma ferramenta de baixo custo, utilizável em *hobbies*, consideram-se os resultados obtidos promissores, tendo em vista que para sinais de até 5 volts em corrente contínua o custo total foi de aproximadamente 10 dólares, compreendendo o Arduino Nano e o módulo Bluetooth. Para tensões maiores que 5 volts, porém ainda em corrente contínua, basta utilizar um divisor resistivo como circuito de pré-condicionamento do sinal, o que aumenta em alguns centavos o custo do projeto. No caso da utilização com sinais de corrente alternada, a implementação do *offset* requer um amplificador operacional e alguns resistores, que juntos custam menos de cinco dólares. O mesmo vale para sinais com amplitude na faixa dos milivolts, onde é recomendada a amplificação do sinal antes da conversão para obtenção de melhor resolução.

### 5.1 TRABALHOS FUTUROS

Alguns aspectos podem ser otimizados em relação ao estágio de desenvolvimento alcançado neste trabalho. O primeiro e mais impactante é a otimização do circuito de con-



dicionamento do sinal, permitindo que mais escalas sejam adotadas, ou até mesmo a seleção automática do circuito de condicionamento de acordo com a escala selecionada na interface gráfica. Para tal, a comunicação bidirecional entre Arduino e Android deve ser implementada de forma que o dispositivo móvel consiga informar ao microcontrolador quais opções foram selecionadas.

Um passo mais avançado seria a detecção e condicionamento do sinal totalmente automáticos, assim como nos osciloscópios profissionais. Porém, isto elevaria o custo e a complexidade do projeto, talvez deixando de atender o requisito de baixo custo e facilidade de implementação.

Dado que a limitação deste projeto foi causada majoritariamente pela velocidade de comunicação entre o microcontrolador e o Bluetooth, a utilização de módulos para a comunicação com o dispositivo móvel que alcancem maior velocidade permitiriam a exibição de sinais de maior frequência e também a utilização de múltiplos canais. Outra alternativa seria a substituição do cristal oscilador do Arduino para que o mesmo trabalhasse em um *clock* compatível com a maior velocidade do módulo Bluetooth. Entretanto, além de mais complexa, esta abordagem poderia resultar em problemas com outros *firmwares* Arduino que baseiam-se no *clock* padrão de 16 Mhz, inviabilizando a utilização da placa em outros projetos, o que talvez não seja desejável pelos hobbistas.

A biblioteca utilizada para a exibição do gráfico no dispositivo móvel é *open-source* e está em constante desenvolvimento. Isto dá margem para futuras otimizações de desempenho tanto nas funcionalidades relacionadas à biblioteca, quanto no restante da aplicação.

Em um viés educacional, a aplicação Android pode ter suas funcionalidades estendidas para salvar logs de experimentos, permitindo a análise posterior dos mesmos e como um meio para professores avaliarem os resultados obtidos pelos alunos. Também é possível a elaboração de práticas guiadas, onde o aluno recebe as instruções de montagem de circuitos pela própria aplicação e, ao obter o resultado proposto, a próxima etapa é liberada.

## REFERÊNCIAS

ABLESON, W. F. et al. **Android in Action**. 3. ed. Shelter Island, NY: Manning Publications Co., 2012. ISBN 9781617290503.

ARDUINO. **Arduino - Introduction**. 2016. Disponível em: <<http://www.arduino.cc/en/Guide/Introduction>>. Acesso em: 23 de abril de 2016.

ARDUINO. **Arduino Mega 2560**. 2016. Disponível em: <<https://www.arduino.cc/en/Main/ArduinoBoardMega2560>>. Acesso em: 24 de abril de 2016.

ARDUINO. **Arduino Nano**. 2016. Disponível em: <<https://www.arduino.cc/en/Main/ArduinoBoardNano>>. Acesso em: 24 de abril de 2016.

ARDUINO. **Arduino Uno**. 2016. Disponível em: <<http://www.arduino.cc/en/Main/ArduinoBoardUno>>. Acesso em: 24 de abril de 2016.

ARDUINO. **Arduino Yún**. 2016. Disponível em: <<https://www.arduino.cc/en/Main/ArduinoBoardYun>>. Acesso em: 24 de abril de 2016.

ATMEL. **Atmel 8-bit Microcontroller with 4/8/16/32 kbytes in-system Programmable Flash Datasheet**. 1600 Technology Drive, San Jose, CA 95110 USA: Atmel, 2015.

BAYLE, J. C **Programming for Arduino**. Birmingham B3 2PB, UK: Packt Publishing, 2013. ISBN 978-1-84951-758-4.

BOYLESTAD, R. L. **Introductory Circuit Analysis**. 11. ed. Upper Saddle River, New Jersey: Pearson, 2007. ISBN 0-13-173044-4.

BOYLESTAD, R. L.; NASHELSKY, L. **Dispositivos Eletrônicos e Teoria de Circuitos**. 8. ed. São Paulo, SP: Pearson, 2004. ISBN 978-85-87918-22-2.

CAVALCANTE, B. M. **Apoio a Localização de Defeitos Durante a Execução de Sequências de Teste para Aplicações Android**. Dissertação (Trabalho de Conclusão de Curso) — Universidade Tecnológica Federal do Paraná, 2015.

DEITEL, P. et al. **Android para Programadores**. Porto Alegre, RS: Bookman, 2013. ISBN 9788540702103.

ELETRÔNICA. **Amplificadores Operacionais AmpOp**. 2016. Disponível em: <<http://www.electronica-pt.com/amplificadores-operacionais-ampop>>. Acesso em: 30 de abril de 2016.

EMARKETER. **Smartphone users worldwide 2014-2019**. 2016. Disponível em: <<http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>>. Acesso em: 30 de abril de 2016.

EVANS, M.; NOBLE, J.; HOCHENBAUM, J. **Arduino in Action**. Shelter Island, NY: Manning Publications Co., 2013. ISBN 9781617290244.

FARTO, G. de C. **Uma Contribuição ao Teste Baseado em Modelo no Contexto de Aplicações Móveis**. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2016.

FRITZING. **Fritzing: electronics made easy**. 2016. Disponível em: <<http://fritzing.org/home/>>. Acesso em: 4 de maio de 2016.

GRAPHVIEW. **Android GraphView**. 2016. Disponível em: <<http://www.android-graphview.org/>>. Acesso em: 30 de setembro de 2016.

IDC. **Smartphone OS Market Share, 2015 Q2**. 2015. Disponível em: <<http://www.idc.com/prodserv/smartphone-os-market-share.jsp>>. Acesso em: 22 de abril de 2016.

JÚNIOR, A. P. **Eletrônica Analógica: Amplificadores Operacionais e Filtros Ativos**. 6. ed. São Paulo, SP: Bookman, 2003. ISBN 978-85-363-0190-7.

KARIM, I. A. A low cost portable oscilloscope based on arduino and gld. In: **Informatics, Electronics Vision (ICIEV), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 1–4.

KUZNETSOV, S.; PAULO, E. Rise of the expert amateur: Diy projects, communities, and cultures. In: **Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries**. New York, NY, USA: ACM, 2010. (NordiCHI '10), p. 295–304. ISBN 978-1-60558-934-3. Disponível em: <<http://doi.acm.org/10.1145/1868914.1868950>>.

LABCENTER ELECTRONICS LTD. **Proteus PCB Design & Simulation Software**. 2016. Disponível em: <<https://www.labcenter.com/>>. Acesso em: 15 de agosto de 2016.

LATHI, B. P.; DING, Z. **Sistemas de Comunicações Analógicos e Digitais Modernos**. 4. ed. Rio de Janeiro, RJ: LTC, 2012. ISBN 978-85-216-2027-3.

LECHETA, R. R. **Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 5. ed. São Paulo, SP: Novatec Editora Ltda., 2015. ISBN 9788575224687.

MAKERS. **Arduino Uno vs Arduino Nano**. 2014. Disponível em: <<http://makers-with-myson.blog.so-net.ne.jp/2014-04-07>>. Acesso em: 24 de abril de 2016.

MORIMOTO, C. E. **A História da informática (Parte 6: Sistemas embarcados e supercomputadores)**. aug 2011. Disponível em: <<http://www.hardware.com.br/guias/historia-informatica/transistor.html>>. Acesso em: 1 de maio de 2016.

NICULESCU, V.; LITA, A. I. Open source oscilloscope for hobby users. In: **2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)**. [S.l.: s.n.], 2015. p. 203–207. ISSN 2068-1038.

OPEN HANDSET ALLIANCE. **Alliance Members**. 2016. Disponível em: <[http://www.openhandsetalliance.com/oha\\_members.html](http://www.openhandsetalliance.com/oha_members.html)>. Acesso em: 22 de abril de 2016.

SEDRA, A. S.; SMITH, K. C. **Microeletrônica**. 4. ed. São Paulo, SP: Makron Books, 2000. ISBN 85.346.1044-4.

SPARKFUN. **How to Use an Oscilloscope**. 2016. Disponível em: <<https://learn.sparkfun.com/tutorials/how-to-use-an-oscilloscope>>. Acesso em: 30 de abril de 2016.

TEKTRONIX. **XYZs of Oscilloscopes**. Beaverton, Oregon, USA: [s.n.], 2000. Disponível em: <<http://ecee.colorado.edu/mcclurel/txyzscopes.pdf>>.

TEXAS INSTRUMENTS. **Single-Supply Op Amp Design Techniques**. [S.l.: s.n.], 2008.

VENKATASHAIAH, L. **Android Architecture: Layers in the Android Stack**. jun 2014. Disponível em: <<https://lokeshv.wordpress.com/tag/android/>>. Acesso em: 22 de abril de 2016.

YI, D. Research and design of virtual oscilloscope based on sound card. In: **Electrical and Control Engineering (ICECE), 2010 International Conference on**. [S.l.: s.n.], 2010. p. 1566–1569.