UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Implementação de um Osciloscópio de baixo custo com exibição gráfica em aplicativo para Android

José Ernesto Almas de Jesus Junior

São Carlos

2016

José Ernesto Almas de Jesus Junior

Implementação de um Osciloscópio de baixo custo com exibição gráfica em aplicativo para Android

Trabalho de Conclusão de Curso do aluno José Ernesto Almas de Jesus Junior, do curso de Engenharia de Computação da Escola de Engenharia de São Carlos e do Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo.

Orientador: Prof. Dr. Maximiliam Luppe

VERSÃO CORRIGIDA

São Carlos

2016

Almas de Jesus Junior, José Ernesto
A446i Implementação de um Osciloscópio de baixo custo com exibição gráfica em aplicativo para Android/ José Ernesto Almas de Jesus Junior; orientador Maximiliam Luppe . São Carlos, 2016.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Microcontrolador. 2. Bluetooth. 3. Android. 4. Osciloscópio. I. Título.

FOLHA DE APROVAÇÃO

Nome: José Ernesto Almas de Jesus Junior

Título: "Implementação de um dispositivo de aquisição de sinais elétricos com microcontrolador e exibição gráfica em aplicativo Android"

Trabalho de Conclusão de Curso defendido em 23/06/2016.

Comissão Julgadora:	**Resultado:**
Prof. Dr. Maximiliam Luppe (Orientador) - SEL/EESC/USP	Aprovado
Prof. Associado Evandro Luís Linhari Rodrigues SEL/EESC/USP	Aprovado.
Profa. Assistente Luiza Maria Romeiro Codá SEL/EESC/USP	Aprovado

Coordenador do Curso Interunidades Engenharia de Computação pela EESC:

Prof. Dr. Maximiliam Luppe

Ao meu pai, meu eterno guia e companheiro.

Agradecimentos

Ao meu pai, minha mãe e minha irmã, por sempre terem me incentivado a crescer como ser humano e profissional.

Aos meus avós, pelo apoio que deram à minha educação.

Aos meus amigos Bati, Biga, Bruno, Cão, Felipe, Gigli e Romeu, por sempre estarem ao meu lado, tanto nos momentos difíceis quanto em uma mesa de bar.

Aos amigos “das torres”, Falqueto, Noia, Nub, Pocs, Lui, Represa, Soldera, Superman, Tim, Wiki, Yuri e Zuera, pelas incansáveis noites juntos (estudando ou não).

À equipe EESC USP Formula SAE, por ter sido minha segunda família e segunda casa.

Resumo

ALMAS DE JESUS JR., J. E. **Implementação de um Osciloscópio de baixo custo com exibição gráfica em aplicativo para Android.** 2016. Trabalho de Conclusão de Curso (Graduação) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016.

Neste projeto foi implementado um dispositivo que faz a leitura da tensão de um sinal analógico e envia seu valor a um aplicativo para Android para exibição. Um microcontrolador realiza a amostragem e processamento do sinal e o aplicativo desenvolvido recebe os dados via Bluetooth e os exibe em um gráfico. Foram implementadas diversas funcionalidades semelhantes às de um osciloscópio comercial, como *triggering*, escala de exibição de tempo variável e tempo de *holdoff* ajustável, todas acessíveis pela interface *touchscreen*. Como medida de desempenho, foi possível fazer aquisição de sinais a 150 mil amostras por segundo com um erro médio de 0,2% em ambos os eixos de tempo e tensão.

Palavras-chave: Microcontrolador, Bluetooth, Android, Osciloscópio.

Abstract

ALMAS DE JESUS JR., J. E. **Implementation of a low cost Oscilloscope with graphical display on an Android application.** 2016. Trabalho de Conclusão de Curso (Graduação) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016.

This project implemented a device that reads the voltage value of an analog signal and sends it to an Android application for viewing. A microcontroller was used for sampling and processing the signal and the developed application receives the data via Bluetooth and shows it in a graph. Some functionalities were implemented based on the ones found on a commercial oscilloscope, such as triggering, variable time display scale and adjustable holdoff time, all accessible and configurable through the touchscreen interface. As a performance index it was possible to sample the electrical signal at 150 thousand samples per second with an average error of 0.2% on both time and voltage axis.

Keywords: Microcontroller, Bluetooth, Android, Oscilloscope.

Lista de figuras

Lista de tabelas

Lista de siglas

AD – Analógico Digital

ADC – *Analog to Digital Converter*

bps – bits por segundo

DIP – *Dual In-line Package*

DSP – *Digital Signal Processor*

FPGA – *Field-Programmable Gate Array*

LED – *Light Emission Diode*

ICSP – *In-Circuit Serial Programming*

PCI – Placa de Circuito Impresso

PWM – *Pulse Width Modulation*

RGB – *Red Green Blue*

SPP – *Serial Port Profile*

sps – *samples per second*

SSID – *Service Set Identifier*

TTL – *Transistor-transistor logic*

USB – *Universal Serial Bus*

Sumário

1 INTRODUÇÃO

Neste capítulo encontram-se a apresentação do trabalho, as motivações e objetivos que culminaram em sua síntese.

1.1 Apresentação

Na área das engenharias elétrica e eletrônica uma ferramenta utilizada no desenvolvimento da maioria de suas atividades é o osciloscópio. Presente em empresas e instituições de ensino e pesquisa, privadas ou não, esta é uma ferramenta essencial nestas áreas, pois são utilizados para observar a mudança de um sinal elétrico pelo tempo, onde sua a variação de sua tensão no tempo pode ser vista em sua tela.

Outra ferramenta muito presente hoje em dia são os dispositivos móveis, como os *smartphones* e os *tablets.* Devido à sua alta versatilidade, grande gama de modelos, diversas funcionalidades oferecidas e à grande variedade de preços encontrada no mercado hoje, estes aparelhos estão cada vez mais presentes na vida das pessoas. Inúmeros modelos executam sistemas operacionais quase tão complexos quanto os utilizados nos computadores pessoais. Entre os sistemas utilizados no mercado está o sistema operacional Android que, atualmente desenvolvido pela Google, é baseado no *kernel* Linux e é desenhado primariamente para dispositivos móveis com interface *touchscreen*. Muito utilizado por vários fabricantes em seus produtos, prevê-se que em 2016 serão produzidos 1,519 bilhão de *smartphones* e que 82,6% destes aparelhos executarão o sistema operacional Android. [IDC, 2016]

Sabendo que hoje é muito comum as pessoas possuírem dispositivos como *smartphones* e *tablets*, que possuem um poder de processamento cada vez maior, teve-se a ideia de implementar um dispositivo com funções semelhantes às de um osciloscópio, utilizando um microcontrolador para captura de sinais elétricos e seu envio a um aplicativo para sua exibição gráfica.

1.2 Motivação e Objetivos

No Brasil os osciloscópios mais baratos e de marcas respeitadas partem da ordem de milhares de reais por unidade, o que em muitas vezes não permite que instituições como escolas e algumas universidades ofereçam este equipamento em quantidades maiores em seus laboratórios. Se fosse possível que cada aluno possuísse seu próprio osciloscópio para auxiliar no desenvolvimento de seus projetos de aula ou pessoais, poderia ser gerado um considerável retorno positivo à sua formação e aprendizado.

Visando criar uma solução para este problema, este projeto teve entre seus principais objetivos possuir um baixo custo. Ao utilizar um microcontrolador como principal componente do hardware de aquisição e ao retirar a necessidade da inclusão de uma tela LCD no projeto, já que será utilizada a tela do *smartphone* ou *tablet*, é possível manter o custo do projeto na ordem de poucas centenas de reais.

Tendo uma ferramenta de análise de sinais elétricos de baixo custo, é possível utilizá-la como parte do material de ensino, dando aos alunos uma maior independência das instalações da instituição e um obstáculo a menos para desenvolver projetos pessoais.

Trabalhos semelhantes a este já foram desenvolvidos, como por Biagori [BIAGORI, 2014], que desenvolveu uma plataforma para aferição de potência consumida por uma carga em corrente alternada utilizando um microcontrolador e exibição em aplicativo Android e em página Web. Como destaques do projeto de Biagori temos seu potencial de impacto econômico benéfico de seus usuários, que poderão monitorar de forma fácil seu consumo de energia ao longo do mês e a opção de visualização do consumo via Web. Como parte negativa temos que a comunicação do medidor de potência consumida com a interface Web se dá através do aplicativo móvel, que recebe os dados do medidor via Bluetooth e os envia a um banco de dados via *internet* para serem acessados pela interface Web. Esta arquitetura traz uma dependência da presença do aparelho móvel executando o aplicativo para o recebimento dos dados via Bluetooth e envio ao banco de dados *online*. Uma solução neste caso seria a utilização de conexão WiFi em vez de Bluetooth, tirando a necessidade do aplicativo para o

funcionamento da interface Web e mantendo a visualização no aplicativo. Enquanto que no projeto de Biagori os gráficos das interfaces não possuem controles de ajuste das escalas ou ferramentas para análise dos dados, a ferramenta desenvolvida neste projeto tem uma aplicação mais aberta e funcionalidades de visualização que permitem uma análise mais detalhada dos dados.

Pode-se citar também o trabalho desenvolvido por Salla [SALLA, 2015], que implementa um aplicativo para dispositivos móveis que faz a leitura de módulos de sensoriamento via WiFi e que permite tanto a visualização instantânea destes dados quanto o seu salvamento em arquivo. É interessante destacar que a comunicação por pacotes TCP por WiFi permite a visualização por múltiplos usuários simultaneamente, o que não é possível quando se utiliza Bluetooth. Um ponto a se levar em conta é a diferença de consumo destas duas tecnologias, onde um módulo WiFi tem um consumo na ordem de centenas de miliamperes e um módulo Bluetooth tem um consumo na ordem de dezenas de miliamperes. Este fator deve ser considerado caso se utilize alimentação do celular ou de baterias nos dispositivos de leitura.

1.3 Estrutura deste documento

Este documento é dividido nas seguintes seções: a introdução, que lista as Motivações que inspiraram e os objetivos do trabalho produzido, seguida de um embasamento teórico para familiarizar o leitor com alguns conceitos fundamentais para a compreensão deste projeto. Posteriormente é relatado como foi feita a escolha dos componentes do projeto e sua como foi feita sua implementação. Após, a seção de Testes, Resultados e Discussões detalha os testes feitos no projeto, cujos resultados são analisados e discutidos. Então é feita uma conclusão do documento que irá retomar o que foi feito no projeto.

2 EMBASAMENTO TEÓRICO

Neste capítulo serão apresentados os conceitos necessários para entendimento do projeto.

2.1 Principais funcionalidades de um osciloscópio

Para a análise de um sinal elétrico analógico com um osciloscópio digital tem-se algumas funcionalidades que são implementadas em quase todos os modelos, as quais são explicadas a seguir.

2.1.1 Trigger

O *trigger* é uma das principais funcionalidades de um osciloscópio. Segundo o documento *Triggering Fundamentals*, da Tektronix, o evento de *trigger* define o ponto no tempo em que um trecho repetitivo de uma forma de onda está estabilizado para visualização. Isso permite ao aparelho exibir em sua tela uma imagem estável da forma de onda do sinal em análise para que esta possa ser vista e compreendida pelo usuário. Sem o *trigger*, ou outra forma de sincronização do sinal, não seria possível exibir uma imagem estática da forma de onda do sinal. [RADIO ELECTRONICS]

Entre os diversos métodos para detecção de *trigger* está o de detecção de borda. Sendo o mais presente nos osciloscópios modernos, este opera com dois principais parâmetros: a inclinação da borda (subida ou descida) e o valor de *trigger.*

Um *trigger* na borda de subida ocorre quando a forma de onda do sinal cruza o valor de *trigger* especificado com uma inclinação positiva. Analogamente, o *trigger* na borda de descida ocorre quando a onda cruza tal valor com uma inclinação negativa. A Figura 1 ilustra, para um sinal de exemplo, como ocorre a captura de *trigger* nas bordas de subida e descida.

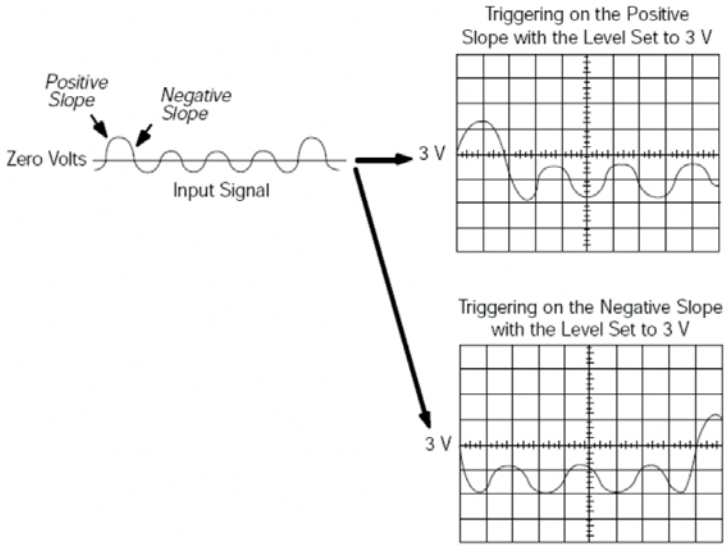

Figura 1: Exemplo de detecção de *trigger* nas bordas de subida e descida.

Fonte: Tektronix.

2.1.2 Tempo de *holdoff*

O tempo de *holdoff* é o tempo entre o momento quando o osciloscópio detecta um *trigger* e o momento quando ele está pronto para detectar outro. Ajustar este tempo facilita encontrar um bom enquadramento de formas de onda complexas, como trens de pulsos. No caso de um trem de pulsos pode-se ajustar o tempo de *holdoff* até que o *trigger* ocorra no primeiro pulso da sequência. [TEKTRONIX]

A Figura 2 ilustra a sequência em que ocorrem a detecção de *trigger*, a janela de aquisição e o tempo de *holdoff*.

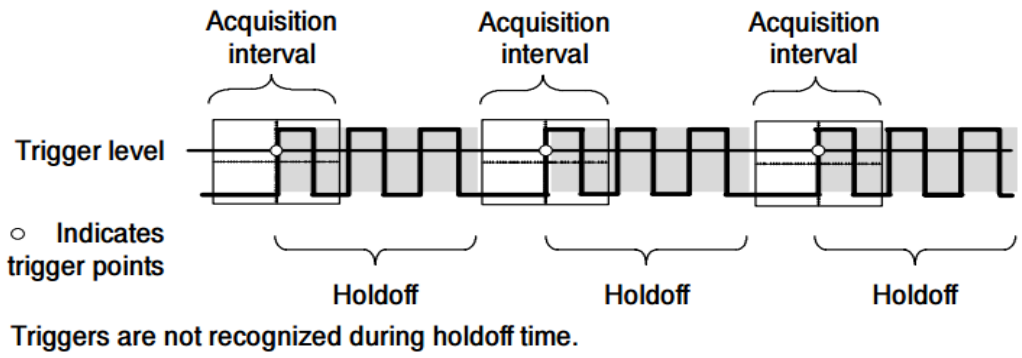

Figura 2: Ilustração da sequência de detecção de holdoff.

Fonte: Manual da série de osciloscópios digitais TDS-200, da Tektronix.

2.2 Bluetooth

O Bluetooth é um protocolo de comunicação sem fio para transmissão de dados em distâncias curtas, desenvolvido para ter um baixo consumo de energia. Por ter também um baixo custo de implementação, este está presente em dispositivos como celulares, *smartphones, tablets, notebooks,* fones de ouvido, controladores de jogos, mouses, teclados, impressoras, sensores, relógios, sons automotivos e câmeras.

Sua última versão é a 4.0 LE (*Low Energy*) que possui uma taxa de transmissão de 24Mbps. A versão utilizada pelo módulo Bluetooth utilizado neste projeto é a versão 2.0 + EDR (*Enhanced Data Rate*) possui uma taxa de transferência de 3Mbps.

A tecnologia é dividida em diversos modos de operação, chamados de perfis, que possuem diferentes configurações para parametrizar e controlar a comunicação entre os dois dispositivos conectados. Em suas especificações são listados suas dependências de outros perfis e quais opções e parâmetros são utilizados de cada camada da pilha de protocolos Bluetooth. Para dois dispositivos se comunicarem via Bluetooth, estes devem suportar o mesmo perfil. A diversidade dos perfis Bluetooth existentes permite que produtos possam utilizar o perfil que mais se adequa ao seu tipo de aplicação. [BLUETOOTH SIG, INC]

Um destes perfis é o *Serial Port Profile* (SPP), que implementa uma comunicação serial transparente aos dispositivos conectados. Este emula um cabo serial com uma porta em cada um dos dois dispositivos pareados, permitindo a comunicação semelhante ao padrão RS-232, incluindo também seus sinais de controle.

2.3 Produtos comerciais

Hoje são encontradas algumas opções de osciloscópios portáteis no mercado que podem ser classificadas em duas vertentes: os de baixo custo e os de alto custo. Os de baixo custo, como o Signtek DS202, podem ser encontrados com preços na ordem de poucas centenas de dólares nos Estados Unidos e têm frequências de amostragem da ordem de dezenas de milhões de amostras por segundo. Este pode ser visto na Figura 3. Já os de alto custo, como o Fluke ScopeMeter 124, têm seu preço na ordem de várias

centenas ou até milhares de dólares e possuem frequências de amostragem semelhantes a osciloscópios comuns de bancada. Este modelo pode ser visto na Figura 4.

Figura 3: Osciloscópio portátil Signtek DS202.

Figura 4: Osciloscópio portátil Fluke ScopeMeter 124.

Existem também aplicativos para Android e dispositivos iOS, como o Oscilloscope, da XYZ-Apps, que utilizam a entrada de microfone do dispositivo para fazer a leitura do sinal elétrico. Mesmo tendo um custo muito baixo, na ordem de unidades de dólares, ou sendo gratuitos, aplicativos como este exigem que seja utilizado algum circuito externo para adequadar os níveis do sinal de entrada, já que há uma limitação da tensão na entrada de microfone do dispositivo da ordem de milivolts, além de oferecerem um perigo à integridade do aparelho caso seja mal utilizado. Outra limitação neste caso é a taxa de aquisição da entrada de microfone, cujo valor máximo é de

44100Hz na maioria dos modelos, já que esta é projetada primariamente para captura de voz.

Outro produto disponível é o PicoScope, desenvolvido pela Pico Technologies. O PicoScope é um osciloscópio portátil sem tela, que possui as dimensões de um celular e se conecta a um computador por uma interface USB. Seu modelo de entrada, 2204A, tem uma largura de banda de 10MHz, possui 2 canais e é vendido a 130 dólares no site do fabricante. A Figura 5 mostra um exemplar da mesma série deste modelo, a série PicoScope 2000, conectado a um computador executando seu *software* de visualização.

Figura 5: PicoScope da série 2000 conectado a um computador.

O PicoScope é uma boa opção de osciloscópio com desempenho suficiente para pequenos projetos e com um preço mais acessível. Entretanto, considerando a conversão de moedas e impostos de importação para o Brasil, este ainda pode ser uma opção cara para quem ainda não é um profissional.

3 IMPLEMENTAÇÃO

Neste capítulo é detalhado o desenvolvimento do trabalho, sendo apresentadas as ferramentas utilizadas, as decisões tomadas durante o projeto e a bibliografia utilizada.

O projeto desenvolvido pode ser dividido em duas partes principais: o *hardware*, responsável pela leitura e conversão dos sinais analógicos para digital e pelo envio via Bluetooth deste sinal; e o aplicativo para Android, responsável por ler os dados recebidos via Bluetooth e exibi-los graficamente, além de ser responsável por repassar as configurações feitas pelo usuário para o *hardware*.

Ambas as partes operam basicamente em dois estados principais, cujas peculiaridades relativas a cada parte do projeto, bem como os motivos que levaram à implementação destes dois modos, serão aprofundadas a seguir. Estes estados são o de envio/recebimento contínuo de dados e o de envio/recebimento de dados em blocos. No primeiro, preza-se por uma atualização constante e fluida do gráfico, portanto os dados são enviados continuamente e o gráfico no aplicativo é atualizado com fluidez. No segundo, preza-se por uma visualização do sinal com maior qualidade e em uma imagem estática para sua análise. Neste, os dados são enviados em blocos, dentro de uma mensagem com sequências de marcação de início e fim, onde trechos do sinal são exibidos estaticamente para melhor visualização de sinais com frequências mais altas. Cada bloco tem todas as amostras que serão exibidas no gráfico simultaneamente.

3.1 Aplicativo para Android

Foi escolhida a plataforma Android por ser uma das mais difundidas em dispositivos eletrônicos, como *smartphones* e *tablets*. Para o desenvolvimento do aplicativo foi utilizado o ambiente de desenvolvimento Android Studio, versão 2.0. Somente as bibliotecas padrões de desenvolvimento da plataforma, disponíveis por padrão no ambiente, foram utilizadas.

Durante o desenvolvimento foi utilizado para teste um *smartphone* Android Motorola X Play com a versão 6.1 (Marshmallow) do sistema operacional, a mais recente no momento da produção deste trabalho.

O aplicativo desenvolvido tem como funções:

- Acessar a conexão Bluetooth do dispositivo;
- Receber os dados do microcontrolador e exibi-los em um gráfico de tensão por tempo;
- Prover uma interface para permitir ao usuário que ajuste as configurações de exibição;
- Enviar comandos de configurações para o microcontrolador.

O aplicativo, desenvolvido nas linguagens Java, para as funcionalidades, e XML, para a interface, tem como destaque quatro classes Java que executam as principais funcionalidades, as quais são descritas a seguir.

3.1.1 Classe Receiver

Classe que cria uma *thread* responsável por estabelecer a conexão Bluetooth com o módulo Bluetooth, ler cada byte recebido, interpretar a mudança entre os modos de recebimento contínuo e em blocos e enviar comandos de configuração para o microcontrolador.

Nesta classe os dados recebidos são armazenados em um *buffer* circular que é checado a cada byte recebido para obter as sequências de início e fim de mensagens e a sequência de mudança entre o modo de recebimento contínuo e o modo de recebimento em blocos. Para isso, foi implementada uma máquina de estado, cujo diagrama pode ser visto na Figura 6, que possui os seguintes estados:

- Estado de recebimento contínuo: estado ativado quando o microcontrolador está no modo de envio contínuo dos dados. Neste estado o gráfico é atualizado com uma frequência de 30Hz, suficiente para produzir uma fluidez em sua atualização.
    - **Condição de saída**: detecção da sequência de início da mensagem com um bloco de dados.

      Próximo estado: estado de recebimento em blocos - dentro de uma mensagem.

- Estado de recebimento em blocos - dentro de uma mensagem: estado ativado enquanto o aplicativo estiver recebendo uma mensagem com um bloco de dados. Quando uma mensagem é recebida com sucesso, o gráfico é atualizado com os dados recebidos.
    - **Condição de saída 1**: fim da mensagem.

      Próximo estado: estado de recebimento em blocos - fora da mensagem.
    - **Condição de saída 2**: recebimento da mensagem de mudança para o estado de recebimento contínuo.

      Próximo estado: estado de recebimento contínuo.
    - **Condição de saída 3**: estouro do tamanho da mensagem. Ocorre quando a sequência de fim de mensagem de bloco de dados não é recebida após ler o número de bytes informado no campo de tamanho da mensagem.

      Próximo estado: estado de recebimento em blocos - fora da mensagem.
- Estado de recebimento em blocos - fora da mensagem: estado ativado quando terminou-se a leitura de uma mensagem de bloco de dados e não se recebeu a mensagem de mudança para o estado de recebimento contínuo nem a sequência de início de outra mensagem de bloco de dados.
    - **Condição de saída 1**: recebimento da sequência de início da mensagem de bloco de dados.

      Próximo estado: estado de recebimento em blocos - dentro da mensagem.
    - **Condição de saída 2**: recebimento da mensagem de mudança para o estado de recebimento contínuo.

      Próximo estado: estado de recebimento contínuo.

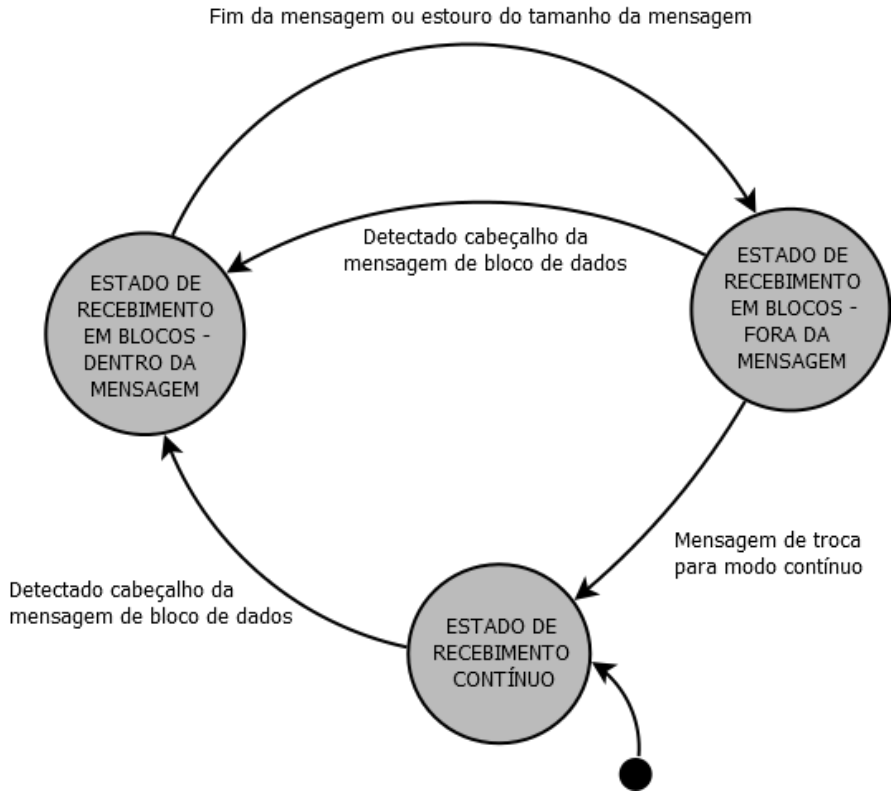

Figura 6: Diagrama de estados da classe Receiver.java.

3.1.2 Classe MySimpleGraph

Classe responsável por desenhar o gráfico do aplicativo. Possui métodos para sua configuração e atualização dos dados.

3.1.3 Classe Settings

Classe responsável por armazenar as configurações atuais do *hardware* e por gerar a sequência de bits de cada comando destas configurações.

3.1.4 Classe FragmentGraph

Classe que controla os elementos da interface de usuário e que faz a mediação entre os dados recebidos na classe Receiver e sua exibição pela classe MySimpleGraph. Esta classe também acessa a classe Settings para obter os valores permitidos para cada configuração.

Cada vez que o usuário altera uma configuração no aplicativo seu respectivo comando é enviado imediatamente ao microcontrolador. Para que isso ocorra, esta

classe faz chamadas à classe Receiver que estes comandos sejam enviados, também acessando a classe Settings para gerar a sequência de bits deste comando.

Para assegurar que o microcontrolador está sempre corretamente configurado, caso haja uma falha no envio de um comando ou caso este seja reiniciado, todos os comando de configuração são reenviados a cada um segundo.

3.1.5 Interface gráfica

O aplicativo é configurado para permanecer somente em modo paisagem, onde a maior dimensão da tela deve ficar no eixo horizontal, a fim de obter um melhor aproveitamento da tela.

A tela inicial do aplicativo pode ser vista na Figura 7. O fundo do gráfico possui um *grid* que o divide em 8 linhas e 10 colunas.

Figura 7: Tela principal do aplicativo de Android com o gráfico não pausado.

Através do aplicativo o usuário pode, além de visualizar a forma de onda do sinal, alterar as configurações de aquisição e visualização, as quais são:

- Ajustar escala do tempo: alteração do espaço de tempo exibido na tela, entre os seguintes valores por divisão: 100µs, 250µs, 500µs, 1ms, 2,5ms, 5ms, 10ms, 25ms, 50ms, 100ms e 250ms;

- Ajustar o tempo de *holdoff*: O tempo de *holdoff* pode ser alterado entre 7 valores distintos, nomeados -3/4, -1/2, -1/4, 0, 1/4, 1/2 e 3/4. Seu funcionamento e o significado de cada valor são melhor abordados na sessão 3.2.5 - Variação do período de *holdoff*;
- Habilitar *trigger*: O botão de *trigger*, localizado no canto inferior esquerdo da tela, configura a captura de *trigger* entre borda de subida, borda de descida e desligado.
- Ajustar o nível de *trigger*: Na lateral esquerda da tela pode-se ver os comandos para ajuste do nível de *trigger*, o qual é representado no gráfico com a linha horizontal pontilhada.
- Parar o gráfico: O botão de pausa, localizado no canto inferior direito da tela, permite que o usuário pare a atualização do gráfico para uma melhor análise do sinal. Este pode ser utilizado tanto durante o modo de recebimento contínuo quanto durante o modo de recebimento em blocos. Ao pausar o gráfico, os controles do ajuste de escala de tempo, nível de *trigger* e tempo de *holdoff* são desabilitados e os controles dos cursores são habilitados. A Figura 8 exibe duas imagens do aplicativo com a atualização do gráfico pausada.
- Cursores: Quando o gráfico está parado, pode-se habilitar os cursores de tensão ou de tempo, que são movidos no gráfico através da interface *touchscreen*, pressionando-se o cursor e arrastando-o até a posição desejada. Exemplos do uso dos cursores podem ser encontrados na Figura 8.

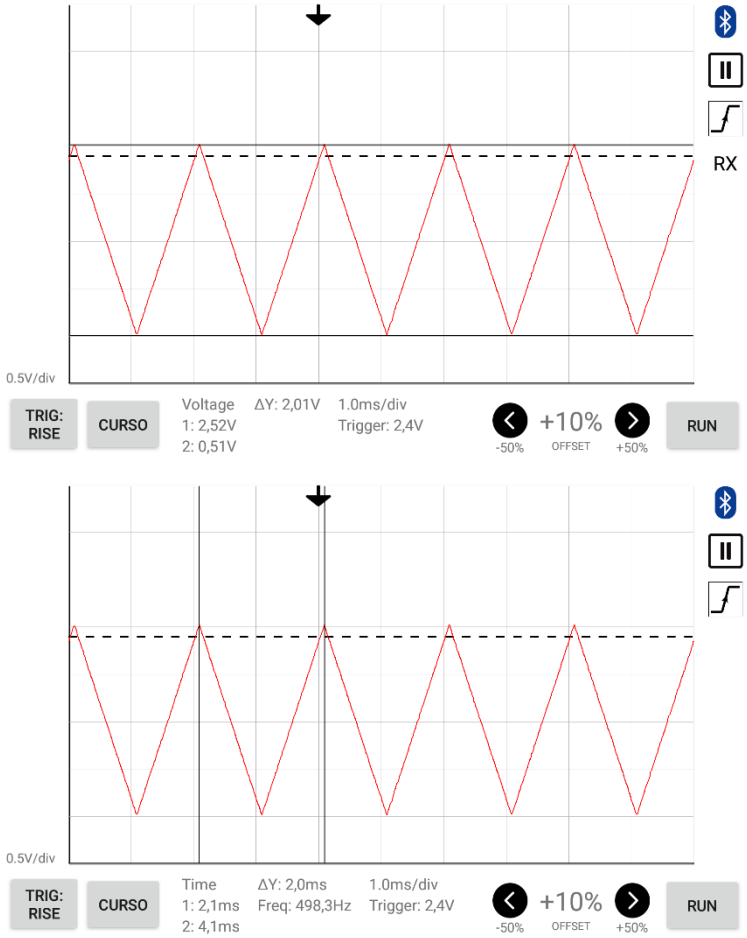

Figura 8: Imagens de telas do aplicativo exemplificando o uso dos cursores de tensão, acima, e dos cursores de tempo, abaixo. Em ambas a atualização do gráfico está pausada.

- Ajustar *offset* de tempo: No canto inferior direito há os botões para ajuste do *offset* de tempo, que permite mover o ponto onde o evento de *trigger* foi detectado pela tela, transladando a janela de exibição do sinal. Inicialmente este ponto, indicado pela seta na borda superior da área do gráfico, está no centro horizontal da tela, e pode ser movido desde o extremo esquerdo até o extremo direto do gráfico. A Figura 9 mostra alguns exemplos deste ajuste para um mesmo sinal capturado.

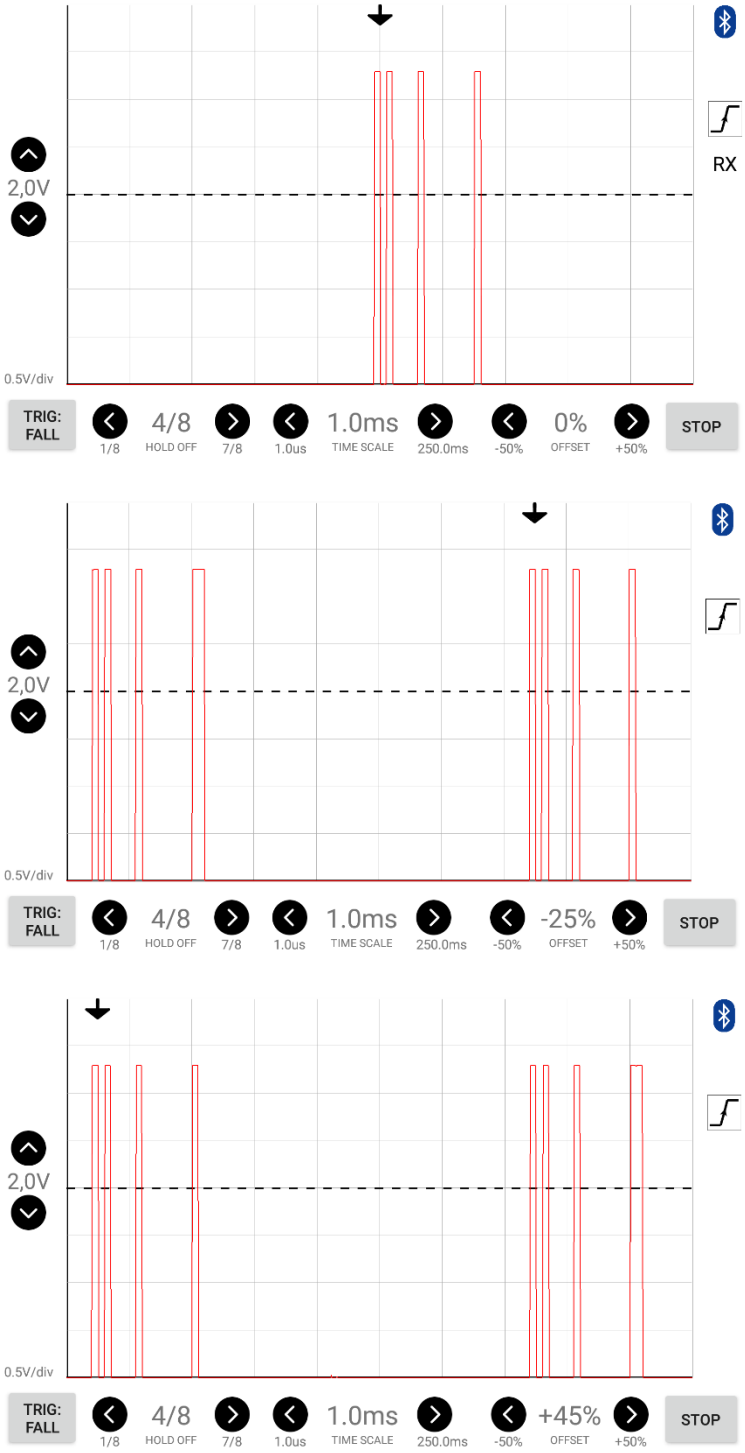

Figura 9: Imagens de telas do aplicativo exemplificando o uso da função de *offset* para um mesmo sinal de entrada, com seu valor em 0% (superior), -25% (centro) e +45% (inferior).

3.2 Microcontrolador

Optou-se por utilizar um microcontrolador para a aquisição do sinal elétrico e envio dos dados digitais para o *smartphone* devido seu baixo custo em relação à outras alternativas, como DSPs e FPGAs, e sua relativa facilidade de compra e desenvolvimento.

Entre as diversas opções no mercado, escolheu-se o microcontrolador da família ARM Cortex M4F, modelo LM4F120H5QR, um microcontrolador de 32 bits desenvolvido pela Texas Instruments. Suas especificações técnicas detalhadas podem ser encontrados no Anexo B. Este modelo foi escolhido por ser de uma arquitetura bem difundida e por ter um desempenho suficiente para executar as tarefas do projeto.

Este microcontrolador foi utilizado através do kit de desenvolvimento Stellaris LaunchPad, da Texas Instruments, o qual pode ser visto na Figura 10. O kit conta com interface para gravação de firmware via ICSP (método de gravação de *firmware* com o microcontrolador já inserido em um circuito) e USB, comunicação serial via USB, pinos de conexão com encapsulamento DIP com espaçamento de 2,54mm e alguns componentes já conectados ao microcontrolador, como um LED RGB, botões e um sensor de temperatura. Todas estas facilidades para desenvolver com o kit, aliadas ao seu baixo preço (cerca de US$12 no site da Texas Instruments nos EUA), também auxiliaram na sua escolha para utilização no projeto.

Figura 10: Kit de desenvolvimento Stellaris LaunchPad da Texas Instruments.

Fonte: [TEXAS INSTRUMENTS 2013c]

O desenvolvimento e depuração do *software* do microcontrolador foram feitos no ambiente de desenvolvimento Code Composer Studio, versão 6.0, mantido e distribuído gratuitamente pela fabricante do kit, Texas Instruments. Para a utilização dos periféricos do microcontrolador utilizou-se a biblioteca StellarisWare, também distribuída pela Texas Instruments, implementada para uso com a linguagem C. Como material de aprendizado sobre o kit, utilizou-se a documentação da biblioteca StellarisWare e o material de tutoriais feito e disponibilizado pela fabricante, o Stellaris Workshop. Também foram utilizados os artigos e manuais extras disponibilizados pela fabricante, os quais foram: *Application Note: Using the Stellaris Microcontroller Analog-to-Digital Converter (ADC)* [TEXAS INSTRUMENTS, 2013a] e *ADC Oversampling Techniques for Stellaris Family Microcontrollers* [TEXAS INSTRUMENTS, 2013b].

Para a aquisição e conversão do sinal elétrico analógico para digital utilizou-se o conversor interno do microcontrolador, que possui uma resolução máxima de 12 bits e uma taxa máxima de aquisição de 1Msps. Segundo a fabricante, entretanto, devido a ruídos e outros fatores que influenciam na precisão do conversor AD, sua resolução efetiva é menor do que 12 bits. [TEXAS INSTRUMENTS, 2013b]

Um módulo conversor AD externo não foi utilizado pois considerou-se o desempenho e a precisão do módulo interno suficientes para o trabalho produzido, além de evitar a adição de mais um componente ao projeto.

A Figura 11 mostra uma síntese dos blocos lógicos implementados em *software*, dos periféricos utilizados no microcontrolador e suas conexões. O bloco ADC faz a conversão da tensão do sinal de entrada para dados digitais, que são armazenados num *buffer.* Este envia os dados para o módulo serial que se comunica com o transmissor Bluetooth. O transmissor envia estes dados para o aplicativo e recebe deste os comandos de configuração, que são repassados para o microcontrolador. Os comandos recebidos são interpretados e alteram as configurações do *timer,* que controla a temporização do módulo ADC, e o tamanho do *buffer*.

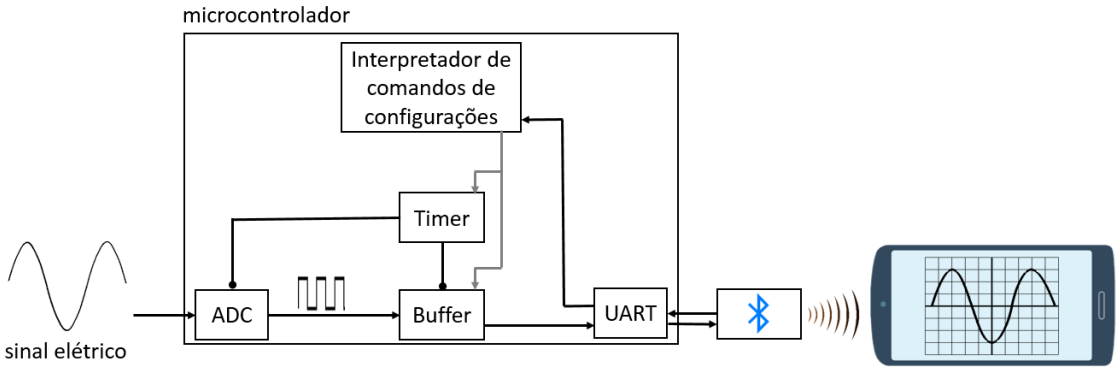

Figura 11: blocos básicos do microcontrolador.

3.2.1 Algoritmo

Baseando-se nos modos de exibição de um osciloscópio, foram implementados no microcontrolador dois principais modos de execução que determinam a taxa de leitura e forma de envio dos dados do sinal lido: o modo de envio em blocos e o modo de envio contínuo.

O modo de envio em blocos preza por uma alta taxa de amostragem do sinal enviado para permitir uma melhor análise deste pelo usuário. Este é ativado quando a detecção de *trigger* está habilitada e o nível do sinal lido cruzou o valor de *trigger* configurado. Neste, é feito em sequência um número fixo de amostras do sinal de entrada, que então é enviado em um único bloco para ser visualizado. Este bloco é exibido na tela e é atualizado com uma frequência de aproximadamente 1Hz, dependendo das configurações de *holdoff* e da forma de onda do sinal.

Esta forma de aquisição e envio em blocos do sinal foi escolhida pelo fato do tempo de envio de um byte de dado pela interface serial do microcontrolador ser de aproximadamente 21,7µs a 460800bps, valor que é alto quando comparado ao tempo médio de aquisição de uma amostra pelo módulo ADC e processamento pelo microcontrolador, cerca de 6,5µs.

Como neste modo o envio dos dados é feito durante o período de *holdoff*, onde o microcontrolador está em espera e não faz leituras do módulo ADC, obteve-se uma taxa de envio cerca de 3 vezes maior do que se cada amostra do ADC fosse enviada imediatamente após sua leitura.

Para determinar o número de amostras de um bloco, considerou-se as características do smartphone e do aplicativo de visualização, além das limitações de tempo das leituras ADC. Como citado na sessão 3.1 - Aplicativo para Android, foi determinado que o aplicativo seria executado apenas em modo paisagem (maior dimensão da tela no eixo horizontal) para permitir um maior aproveitamento da tela. A tela do *smartphone* utilizado possui a resolução Full HD (1920 por 1080 pixels), comum à maioria dos *smartphones* de médio e alto nível hoje. Considerando a maior dimensão e um desconto neste valor devido às bordas do aplicativo, escolheu-se o valor de 1000.

O valor 1000 é número de amostras a ser exibido na tela nas escalas de tempo maiores (desde 250ms/divisão até 1ms/divisão). Para as escalas inferiores a 1ms/divisão este número teve que ser reduzido para respeitar o limite de tempo de uma leitura do módulo ADC e processamento do microcontrolador. A Tabela 1 mostra o tamanho do bloco de dados de todas as escalas, assim como o tempo entre uma amostra e outra. Alguns valores tiveram seu valor ajustado em relação ao limite teórico, já que este não leva em conta o tempo de execução das instruções de controle do fluxo do programa e de cálculos.

Tabela 1: Número de amostras do modo de envio em blocos e espaço entre duas aquisições subsequentes para cara valor da escala de tempo.

Escala de tempo (por divisão)	**Número de amostras no modo de aquisição e envio em blocos**	**Espaço de tempo entre duas aquisições subsequentes (s)**
100µs	130	7,69E-06
250µs	350	7,14E-06
500µs	700	7,14E-06
1ms	1000	1,00E-05
2,5ms	1000	2,50E-05
5ms	1000	5,00E-05
10ms	1000	1,00E-04
25ms	1000	2,50E-04
50ms	1000	5,00E-04
100ms	1000	1,00E-03
250ms	1000	2,50E-03

Para permitir a implementação de um *offset* de tempo na interface, o número de amostras enviado pelo microcontrolador é o dobro dos valores da Tabela 1, com a

amostra onde o *trigger* foi detectado estando no índice do meio deste bloco. Isso permite a interface a exibir o período de um bloco antes e um bloco depois do momento que o *trigger* foi detectado.

Já o modo de envio contínuo consiste na leitura de uma amostra pelo módulo ADC e seu subsequente envio, pois neste modo deseja-se visualizar o sinal continuamente na tela do dispositivo. Este é ativado quando a detecção de *trigger* estiver desabilitada ou quando esta estiver habilitada e o *trigger* não tiver ocorrido. Neste modo a cada um segundo é enviada uma mensagem para notificar o programa receptor (o aplicativo para Android, no caso) de que o microcontrolador está neste modo e que o gráfico deve ser atualizado constantemente com os valores recebidos.

O tempo entre o envio de duas amostras subsequentes no modo de envio contínuo foi obtido ao somar-se o tempo de amostragem e processamento do microcontrolador (6,5µs) com o tempo de envio de um byte pela comunicação serial (21,7µs) e estabelecendo-se um valor máximo de 500 pontos a serem exibidos no gráfico, metade dos pontos do maior bloco de dados do modo de envio em blocos.

Conforme a escala de tempo diminui, o número de amostras a serem exibidas no gráfico também diminui para respeitar o limite de tempo citado acima. A Tabela 2 exibe o número de amostras no modo de aquisição e envio contínuo para cada escala de tempo.

Tabela 2: Número de amostras no modo de aquisição e envio contínuo para cada escala de tempo.

Escala de tempo (por divisão)	**Número de amostras no modo de aquisição e envio contínuo**
100µs	10
250µs	20
500µs	50
1ms	100
2,5ms	200
5ms	500
10ms	500
25ms	500
50ms	500
100ms	500
250ms	500

Para a execução e mudança entre estes dois modos de envio, foi implementada uma máquina de estados com quatro estados no microcontrolador, sendo eles:

- Estado “envio contínuo”: estado onde o modo de envio contínuo está ativado.

Condição de saída: a captura de *trigger* está habilitada e um evento de *trigger* foi detectado.
Próximo estado: “*trigger* detectado”.

- Estado “*trigger* detectado”: estado onde o modo de envio dos dados em blocos está ativado e a captura de um bloco de dados é feita.

Condição de saída: o número de amostras de um bloco foi atingido. É na transição deste estado para o estado “*holdoff sleep”* que é feita a transmissão dos dados.
Próximo estado: “*holdoff sleep*”.

- Estado “*holdoff sleep*”: estado onde o microcontrolador fica em espera sem analisar as amostras do sinal. Sua duração é variável e é ajustada pelo valor de *holdoff* na interface do aplicativo.

Condição de saída: o tempo de espera foi atingido.
Próximo estado: “*holdoff detect*”.

- Estado “*holdoff detect*”: estado onde o microcontrolador não envia dados e aguarda a detecção de um *trigger* até seu limite de duração ser atingido.

Condição de saída 1: um evento de *trigger* foi detectado.
Próximo estado: “*trigger* detectado”.
Condição de saída 2: o tempo de espera máximo foi atingido.
Próximo estado: “envio contínuo”.

Uma representação gráfica dos estados implementados e de suas transições pode ser vista na Figura 12.

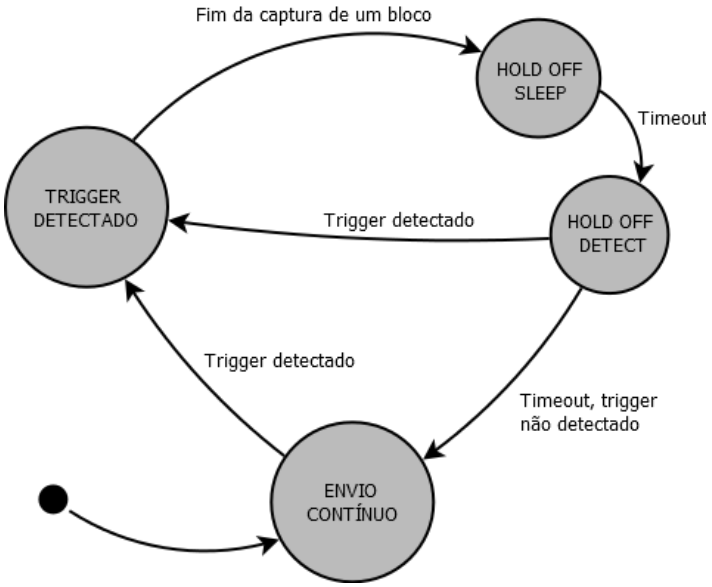

Figura 12: Diagrama de estados do microcontrolador.

Um exemplo do funcionamento do programa do microcontrolador pode ser visto na Figura 13, que exibe a sequência dos estados ao capturar um sinal. Nela o microcontrolador está no estado de "envio contínuo" (CONT, na figura), ou seja, no modo de envio contínuo, até o tempo 1. No tempo 2, é detectado um evento de *trigger*, que é descrito pelo sinal "detecção *trigger*". Do tempo 2 ao tempo 4 é feita a captura do sinal durante o estado de "*trigger* detectado" (TRIGGER, na figura).

Com o fim da aquisição do sinal no tempo 5, o estado de "*holdoff sleep"* (HOS, na figura) é habilitado e dura até o tempo 8. Neste período houveram duas transições do sinal, nos tempos 7 e 8, que poderiam ser detectadas como eventos de *trigger*. Porém, como abordado anteriormente, neste estado a detecção de eventos de *trigger* está desabilitada e, portanto, estes eventos são ignorados.

O estado de "*holdoff sleep"* dura até o tempo 9, onde é habilitado o estado de *"holdoff detect"* (HOD, na figura). Neste estado a detecção de *trigger* é reabilitada e, no tempo 12, outro evento de *trigger* ocorre, dando início a todo o processo a partir do estado de "*trigger* detectado".

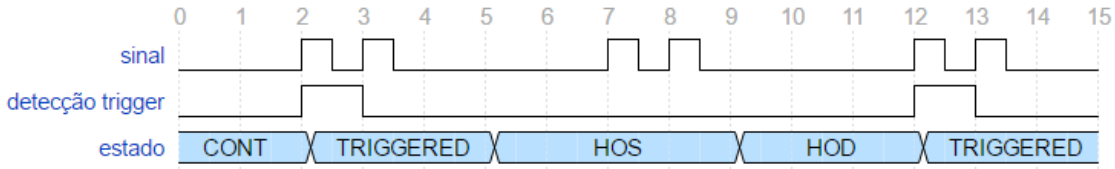

Figura 13: Exemplo das transições dos estados durante a aquisição de um sinal.

3.2.2 Temporização

Toda a temporização utilizada no funcionamento do microcontrolador é regrada de acordo com a escala de tempo selecionada. O algoritmo exibido no Trecho de Código 1 ilustra a base do funcionamento do microcontrolador quanto à temporização das tarefas.

```
booleano realizar_tarefas = falso;
Interrupção_temporizador()
{
    realizar_tarefas = true;
}
Método_principal()
{
    Enquanto(1)
    {
        Se realizar_tarefas:
        {
            realizar_tarefas = falso;

            amostra_atual = getAmostraAtual();

            buffer_circular[índice_atual] =
                        realizarLeituraSerial();

            Escolha(estado_atual):
                // realizar tarefas de acordo com o estado atual
        }
    }
}
```

Trecho de Código 1: Algoritmo de temporização das tarefas implementado no microcontrolador.

O tempo de estouro do temporizador é baseado no tempo entre cada amostra na escala de tempo atual durante o modo de envio em blocos, conforme a Tabela 1. partir deste período é calculado o tempo de duração do estado de "*holdoff sleep*" e o tempo de duração máxima do estado de "*holdoff detect*".

Cada vez que o programa entra no bloco “Enquanto”, é lida a última amostra capturada pelo módulo AD. Antes de retornar a amostra atual, a função `getAmostraAtual()` já rearma a captura da próxima amostra, que ocorrerá enquanto o programa principal executa o resto das suas ações.

3.2.3 Implementação de cada estado

Abaixo serão detalhadas as tarefas executadas em cada estado implementado no microcontrolador, em forma de algoritmo. Estas tarefas são executadas dentro do bloco de Escolha (*switch,* na linguagem C) contido no Trecho de Código 1.

3.2.3.1 Estado de envio contínuo

O Trecho de Código 2 mostra o algoritmo implementado no estado de “envio contínuo” onde, basicamente, é implementado um divisor de frequência, onde o valor inicial da variável `continuousSamplingHoldOff` é o fator desta divisão, tendo seu valor decrementado a cada período de amostra. A amostra é então enviada apenas quando esta variável atinge o valor 0, quando é recarregada com seu valor inicial, que varia de acordo com a escala de tempo selecionada.

```
1  caso CONTINUOUS:
2      se (continuousSamplingHoldOff != 0)
3          continuousSamplingHoldOff--;
4      senão {
5          enviarSerial(amostra_atual);
6          continuousSamplingHoldOff =
               getContinuousModeSamplingSpacing();
7      }
8      break;
```

Trecho de Código 2: Algoritmo das tarefas executadas no estado do modo de envio contínuo.

Para a obtenção do valor inicial da variável `continuousSamplingHoldOff` tomamos inicialmente a frequência que será dividida, a frequência de aquisição no modo de envio em blocos (chamada aqui de $f_{aq\ bloco}$) que, novamente, é a frequência base da temporização do algoritmo:

$$f_{aq\ bloco} = \frac{1}{t_{aq\ bloco}} = \frac{1}{\frac{duração\ bloco}{NAB}} = \frac{NAB}{duração\ bloco}$$

Onde:

- NAB é o número de amostras em um bloco, no modo de captura em bloco;
- $duração\ bloco$ é o tempo de duração total de um bloco de amostras na escala de tempo atual.

Tendo que a frequência de aquisição e envio do modo de envio contínuo ($f_{aq\ cont}$) é igual ao número de amostras a serem exibidas simultaneamente no gráfico durante este modo (chamado aqui de NAC), que pode ser visto na Tabela 2, obtem-se:

$$f_{aq\ cont} = NAC = \frac{f_{aq\ bloco}}{fator\ de\ divisão}$$

Isolando o fator de divisão, que deseja-se obter:

$$fator\ de\ divisão = \frac{f_{aq\ bloco}}{NAC} = \frac{NAB}{duração\ bloco * NAC}$$

Finalmente, o valor inicial da variável `continuousSamplingHoldOff` será o fator de divisão obtido acima.

3.2.3.2 Estado de trigger detectado

O Trecho de Código 3 mostra o algoritmo implementado no estado de "*trigger* detectado", onde é feita a verificação de se o bloco de dados atual já foi preenchido. Quando o bloco for preenchido, as seguintes tarefas são executadas:

- Linha 5: A mensagem com o bloco de dados é enviada;
- Linha 7: É calculado o valor inicial do contador da duração total dos estados de "*holdoff sleep"* e "*holdoff detect",* armazenado em `hold_off_counter`;
- Linha 8: É calculado em qual valor da variável `hold_off_counter` haverá a mudança do estado "*holdoff sleep*" para o estado "*holdoff detect*";
- Linha 10: O estado atual é alterado para "*holdoff sleep".*

```
1  caso TRIGGERED:
2      // detectar fim do bloco de dados atual
3      se (indice_amostra_atual == indiceFimBlocoAtual())
4      {
5          enviarBlocoDeDados();
6
7          hold_off_counter = calculateHoldOffTicks();
```

```
 8          hold_off_sleep_limit = calculateHoldOffSleepTicks();
 9
10          estado_atual = HOLD_OFF_SLEEP;
11      }
12      break;
```

Trecho de Código 3: Algoritmo das tarefas executadas no estado "*trigger* detectado".

3.2.3.3 Estado de holdoff sleep

O Trecho de Código 4 mostra o algoritmo implementado no estado "*holdoff sleep*", que basicamente resulta em uma espera de `hold_off_counter - hold_off_sleep_limit` períodos de amostra.

```
1  caso HOLD_OFF_SLEEP:
2      se (hold_off_counter < hold_off_sleep_limit || hold_off_counter ==
   0)
3          estado_atual = HOLD_OFF_DETECT;
4      senão
5          hold_off_counter--;
6      break;
```

Trecho de Código 4: Algoritmo das tarefas executadas no estado "*holdoff sleep*".

3.2.3.4 Estado de holdoff detect

O Trecho de Código 5 mostra o algoritmo implementado no modo "*holdoff detect*", o qual apenas aguarda o contador `hold_off_counter` chegar ao valor zero. Caso a variável chegue a zero o evento de *trigger* não ocorreu e, portanto, muda-se para o estado "envio contínuo". A detecção do *trigger* é feita em outro trecho do código, a ser discutido a seguir.

```
1  caso HOLD_OFF_DETECT:
2      // Acabou a duração do estado
3      se (hold_off_counter == 0) {
4          ctrl_current_state = CONTINUOUS;
5          continuousSamplingHoldOff =
                   getContinuousModeSamplingSpacing();
6      }
7      senão
8          hold_off_counter--;
9      break;
```

Trecho de Código 5: Algoritmo das tarefas executadas no estado "*holdoff detect*".

3.2.4 Detecção de trigger

O tipo de detecção de *trigger* implementada foi a detecção de borda. No aplicativo pode-se configurar captura de borda de subida ou descida e o nível de tensão. Quando a detecção de *trigger* está habilitada, o microcontrolador compara a amostra atual do sinal com a anterior.

Quando configurado para borda de subida, se a amostra atual estiver acima do valor de *trigger* configurado e a amostra anterior estiver abaixo deste valor, o *trigger* é dado como caracterizado.

Análogo à detecção de borda de subida, a detecção de borda de descida ocorre quando a amostra atual tem um valor abaixo do configurado e a anterior tem um valor maior.

O Trecho de Código 6 mostra o algoritmo implementado na detecção do evento de *trigger*. Caso este seja detectado, o estado atual se torna o estado "*trigger* detectado" e o índice do bloco é armazenado. Este último é calculado tendo a posição atual no *buffer* e o valor de *offset* configurado.

```
 1  se (triggerHabilitado() E
 2      (estado_atual == CONTINUOUS OU estado_atual == HOLD_OFF_DETECT))
 3  {
 4      amostra_anterior = getAmostraAnterior();
 5
 6      se (
 7          (borda_trigger == RISE
 8           E amostra_atual >= getAtualNivelTrigger()
 9           E amostra_anterior < getAtualNivelTrigger())
10
11          OU
12
13          (borda_trigger == FALL
14           E amostra_atual <= getAtualNivelTrigger()
15           E amostra_anterior > getAtualNivelTrigger())
16         )
17      {
18          estado_atual = TRIGGERED;
19          indice_inicio_bloco = getInicioBloco();
20      }
21  }
```

Trecho de Código 6: Algoritmo de detecção de *trigger*.

3.2.5 Variação do período de *holdoff*

Durante o desenvolvimento percebeu-se que sinais simples como sinais senoidais com uma harmônica, sinais PWM, entre outros, eram capturados através da captura de *trigger* e exibidos corretamente na tela. Entretanto, quando se testou a leitura de sinais mais complexos, como trens de pulsos irregulares, percebeu-se que o trecho do sinal exibido na tela não era sempre o mesmo.

A Figura 14 mostra esta situação de uma forma simplificada. Nela são ilustrados em quais momentos cada estado do microcontrolador é habilitado. O sinal de entrada possui um período de 6 blocos e tem início nos tempos 0, 6 e 12. O sinal "captura" indica o trecho do tempo após a detecção do evento de *trigger* onde o sinal é capturado e exibido no gráfico, que é representado pelo estado de envio em blocos.

Os trechos capturados na situação descrita nesta figura podem ser vistos na Figura 15. É fácil notar que, mesmo sendo trechos do mesmo sinal, as imagens são diferentes. Vale ressaltar que a proporção entre as durações de cada estado na Figura 14 não são as mesmas das que ocorrem na prática, já que esta tem caráter apenas ilustrativo.

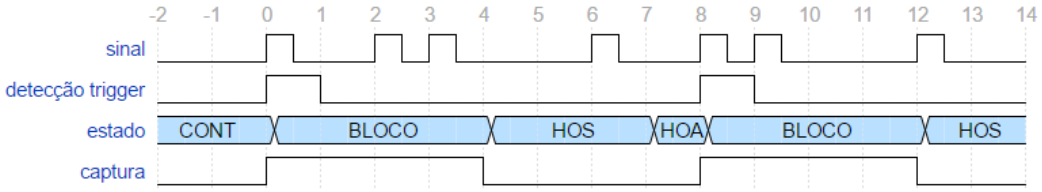

Figura 14: Exemplo de sinal com capturas subsequentes diferentes.

Figura 15: trechos do sinal capturado na situação da *Figura 14*.

Esta diferença entre os trechos capturados do sinal ocorreu pois a duração do estado "*holdoff sleep*" não foi compatível com o período do sinal. Se este terminasse na transição do tempo 5 para o 6 ou após o tempo 9, a detecção do *trigger* teria ocorrido

no momento certo para que a imagem da segunda captura do sinal fosse igual à primeira.

A Figura 16 mostra a captura do mesmo sinal da Figura 14 com uma duração diferente do estado de "*holdoff sleep*". Neste caso, as imagens capturadas do sinal serão todas iguais.

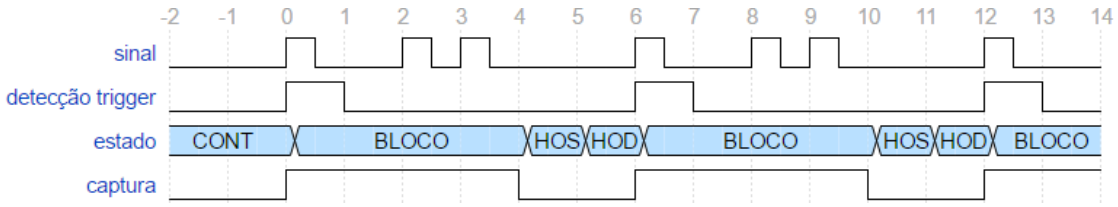

Figura 16: Exemplo de sinal com capturas subsequentes diferentes e com a duração de *holdoff sleep* ajustada.

Tendo em vista que situações como esta ocorreriam, e que não há um único período de *holdoff* que funcionaria para qualquer sinal de entrada, considerou-se interessante possibilitar a variação do período do estado "*holdoff sleep*".

A duração de *holdoff sleep* implementado no projeto segue a fórmula:

$$HOS_{efetivo} = HOS_{base} + fatorHOS * T_{bloco}$$

Onde:

- $HOS_{efetivo}$ é a duração de *holdoff sleep* que será utilizada;
- HOS_{base} é a duração de *holdoff sleep* base, que é de 1000ms;
- $fatorHOS$ é o valor ajustado pelo usuário, entre os seguintes valores: -3/4, -1/2, -1/4, 0, 1/4, 1/2 e 3/4;
- T_{bloco} é a duração de aquisição de um bloco de dados na escala de tempo atual.

Assim, a duração do estado de *holdoff sleep* é de 1000ms com possiblidade de um pequeno ajuste que é uma fração do tempo de duração do bloco de captura atual.

3.2.6 Ajuste da escala de tensão

Devido aos limites de tempo e recursos para o desenvolvimento deste projeto, decidiu-se não implementar o ajuste da escala de tensão de leitura. Assim, o sinal lido pelo microcontrolador é limitado pela sua tensão de alimentação, ou seja, não deve exceder 3,3V e não deve ter níveis abaixo de 0V.

3.3 Módulo Bluetooth

Para a comunicação entre o microcontrolador e o smartphone foi escolhida a tecnologia sem fio Bluetooth, tanto por estar presente em quase todos os dispositivos Android como por ser fácil de encontrar módulos de comunicação Bluetooth para microcontroladores. O módulo escolhido foi o modelo HC-05 [ITEAD STUDIO], o qual pode ser visto na Figura 17.

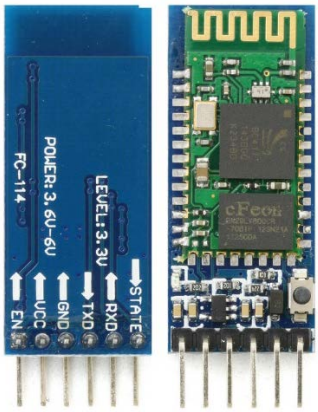

Figura 17: Módulo Bluetooth HC-05.

O HC-05 é um módulo Bluetooth SPP (*Serial Port Protocol*), feito para prover uma configuração de conexão sem fio transparente. Este modelo é fácil de ser encontrado no mercado e pode ser encontrado pronto para ser utilizado em uma placa de circuito impresso com uma barra de pinos com espaçamento de 2,54mm.

O HC-05 vem de fábrica pronto para ser utilizado, já configurado com uma SSID, senha e taxa de transmissão da interface serial. A Tabela 3 mostra os valores dessa configuração de fábrica e os novos valores utilizados para o projeto.

Tabela 3: Parâmetros de configuração do módulo HC-05.

Parâmetro	**Valor original**	**Valor configurado**
SSID	HC-2010-06-01	TCC
Configuração serial	38400/8-N-1	460800/8-N-1
Senha	1234	1234

Para configurá-lo, deve-se ligá-lo pressionando o botão de configuração, que pode ser visto na imagem da direita da Figura 17, próximo à barra de pinos. O LED da placa irá começar a piscar com um período de 2 segundos, demonstrando que o módulo entrou no modo de configuração. No modo de configuração a comunicação serial é fixa a 38400bps e os comandos são enviados em modo texto. No Apêndice C pode-se verificar os comandos utilizados para obter as configurações da Tabela 3.

3.4 Comunicação

A comunicação entre o microcontrolador e o dispositivo Android se deu via Bluetooth, utilizando o módulo Bluetooth HC-05, que implementa o perfil de comunicação Bluetooth via SSP. A comunicação do microcontrolador com o módulo se deu via protocolo serial com velocidade de 460800bps. Algumas convenções para a comunicação foram utilizadas para se assegurar uma comunicação eficiente, as quais serão explicadas nesta subseção.

3.4.1 Comunicação aplicativo – microcontrolador

A comunicação do aplicativo para o microcontrolador é utilizada para o envio de comandos de configuração. Os comandos possuem um byte de tamanho e têm o formato ilustrado na Figura 18.

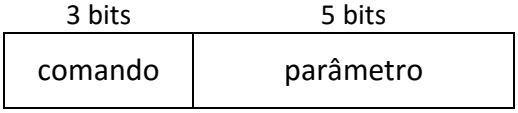

Figura 18: Formato do byte de comando enviado do aplicativo Android para o microcontrolador.

Os possíveis valores do campo de comando e dos seus respectivos campos de parâmetro estão listados na Tabela 4.

Tabela 4: Valores implementados do campo de comando e a interpretação do campo de parâmetro.

Comando	Valor	Interpretação do campo de parâmetro
Definir *holdoff*	001_B	-3/4 = 00000_B -1/2 = 00001_B -1/4 = 00010_B 0 = 00011_B 1/4 = 00100_B 1/2 = 00101_B 3/4 = 00110_B
Definir escala de tempo total	010_B	100µs = 00011_B 250µs = 00100_B 500µs = 00101_B 1ms = 00110_B 2,5ms = 00111_B 5ms = 01000_B 10ms = 01001_B 25ms = 01010_B 50ms = 01011_B 100ms = 01100_B 250ms = 01101_B 500ms = 01110_B 1s = 01111_B 2,5s = 10000_B
Desabilitar detecção de *trigger*	100_B	N/A
Habilitar detecção de *trigger* na borda de subida	101_B	Valor mínimo da escala de tensão = 00000_B ... Valor máximo da escala de tensão = 11111_B
Habilitar detecção de *trigger* na borda de descida	110_B	

3.4.2 Comunicação microcontrolador - aplicativo

Como elucidado anteriormente, o microcontrolador trabalha em dois principais modos, o modo de envio contínuo e o modo de envio em blocos.

No modo de envio contínuo, as amostras são lidas e enviadas continuamente. Visando maximizar a taxa de envio, neste modo apenas os 8 bits mais significativos das amostras são enviados, já que no protocolo serial os dados são enviados byte a byte.

Para alertar o aplicativo Android que o microcontrolador está nesse modo, é enviado a cada um segundo a sequência de bytes ilustrada na Figura 19.

	Sequência				
Caractere	U	S	P	C	N
Decimal	85	83	80	67	78

Figura 19: Sequência de caracteres de mudança para o estado de envio/recebimento contínuo de dados.

No modo de envio em blocos, os dados são enviados em uma mensagem que, além dos valores das amostras, possui algumas informações sobre o bloco de dados sendo enviado. Os campos desta mensagem e seus tamanhos podem ser vistos na Figura 20.

5 bytes	1 byte	2 bytes	n bytes	3 bytes
Preâmbulo (85 83 80 66 75 ou USPBK)	Escala de tempo	Tamanho dos dados	Dados	Sequência de fim (40 80 200)

Figura 20: Formação da mensagem de envio de bloco de dados.

3.5 Estrutura para desenvolvimento

Todo o *hardware* do dispositivo foi montado em uma matriz de contato (ou *protoboard*), a fim de facilitar o desenvolvimento do projeto, já que pequenas alterações nos circuitos eram feitas no decorrer do projeto.

Para alimentar os circuitos externos ao kit Stellaris foi utilizada uma placa fonte de alimentação para *protoboards*, com entrada USB e saída selecionável de 3,3V ou 5V.

Esta placa de alimentação permitiu ter trilhas de alimentação de 3,3V para os circuitos de testes conectados ao microcontrolador e alimentação de 5V para o módulo transmissor Bluetooth.

Ao decorrer do desenvolvimento foram utilizados osciloscópios e geradores de funções comerciais para leitura e geração de sinais analógicos. As formas de onda

obtidas pelo dispositivo desenvolvido eram constantemente comparadas à exibida pelo osciloscópio para ter-se certeza de que não haviam erros na implementação.

Para não depender apenas das simples formas de ondas obtidas no gerador de funções, também foram utilizados simples circuitos analógicos e o próprio microcontrolador para geração de ondas com formas diferentes.

3.6 Lista de componentes

A lista de componentes utilizado para a contrução da parte de *hardware* do projeto e seus respectivos preços podem ser vistos na Tabela 5. Estes preços foram retirados do site da Texas Instruments, no caso do kit Stellaris LaunchPad, e da loja DigiKey, e são apenas para ilustração.

Tabela 5: lista de componentes utilizados no projeto e seus preços.

Componente	**Preço**
Kit de desenvolvimento Stellaris LaunchPad	US$12,00
Módulo Bluetooth HC-05	US$9,00
Protoboard com fios de conexão	US$6,50
Placa fonte de alimentação	US$6,00
Total	**US$33,50**

4 TESTES, RESULTADOS E DISCUSSÕES

Para testar o projeto foram feitas leituras de sinais com variadas formas de onda e frequências. Nesta sessão, para cada caso, é apresentada a leitura feita pelo dispositivo projeto em uma ou mais escalas de tempo distintas e a leitura feita por um osciloscópio comercial e, em seu final, todas as taxas de erro obtidas são exibidas em uma tabela.

Para os teste foi utilizado o osciloscópio da marca Agilent, modelo DSO-X 2002A, que opera a uma frequência de amostragem máxima de 2Gsps e uma largura de banda máxima de sinal de 70MHz. Este osciloscópio possui uma precisão de ±0,01V na escala de tensão de 1V por divisão, utilizada nos testes aqui descritos, e de ±0,16% em todas as escalas de tempo utilizadas nos testes. Mais detalhes técnicos sobre este osciloscópio podem ser encontrados no Anexo C deste documento. O preço deste osciloscópio na loja do fabricante é de US$1300.

Na parte inferior das imagens da tela do aplicativo podem ser vistos os valores de frequência e período medidos com os cursores.

4.1 Onda quadrada

O sinal utilizado foi uma onda quadrada, com *duty cycle* de 50%, variando de 0V a 3,3V. Foi utilizado o gerador de função do próprio osciloscópio.

4.1.1 Frequência de 50Hz

A Figura 21 mostra a leitura de uma onda quadrada de 50Hz no osciloscópio.

A Figura 22 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 10ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,4% no eixo do tempo.

A Figura 23 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 5ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,4% no eixo do tempo.

A Figura 24 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 2,5ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,0% no eixo do tempo.

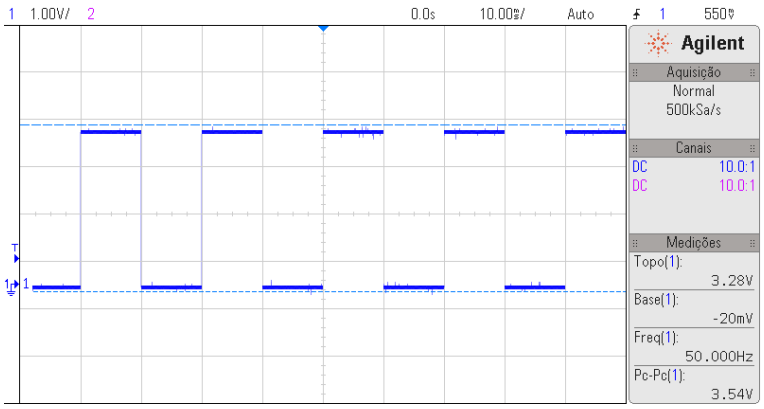

Figura 21: Onda quadrada de 50Hz visualizada no osciloscópio.

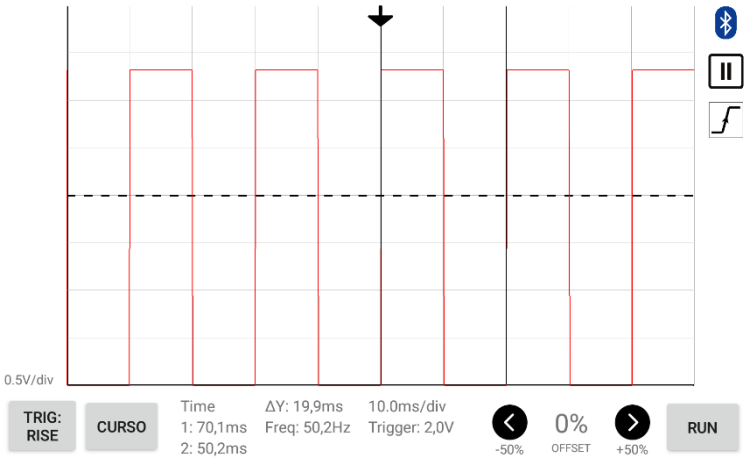

Figura 22: Onda quadrada de 50Hz visualizada no aplicativo com escala de 10ms/divisão.

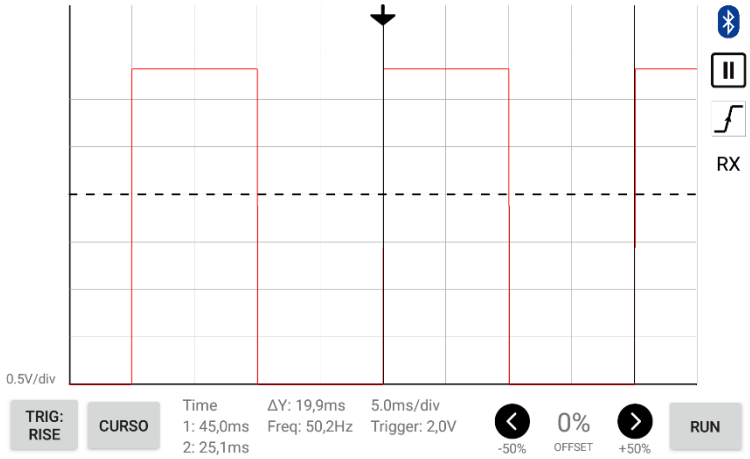

Figura 23: Onda quadrada de 50Hz visualizada no aplicativo com escala de 5ms/divisão.

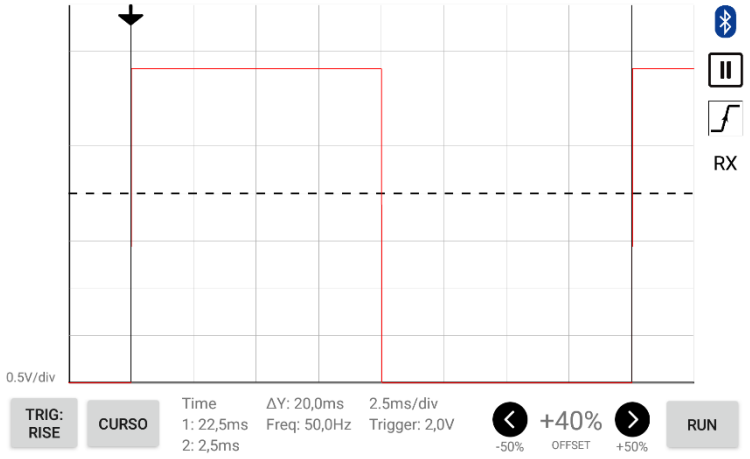

Figura 24: Onda quadrada de 50Hz visualizada no aplicativo com escala de 2,5ms/divisão.

4.1.2 Frequência de 100Hz

A Figura 25 mostra a leitura de uma onda quadrada de 100Hz no osciloscópio.

A Figura 26 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 5ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,0% no eixo do tempo.

A Figura 27 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 2,5ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,0% no eixo do tempo.

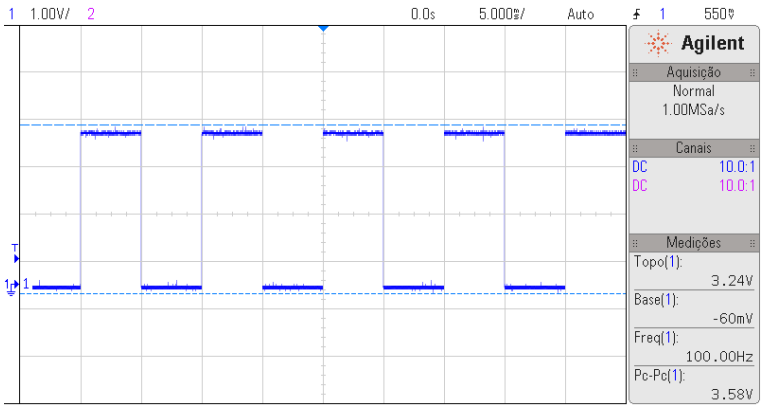

Figura 25: Onda quadrada de 100Hz visualizada no osciloscópio.

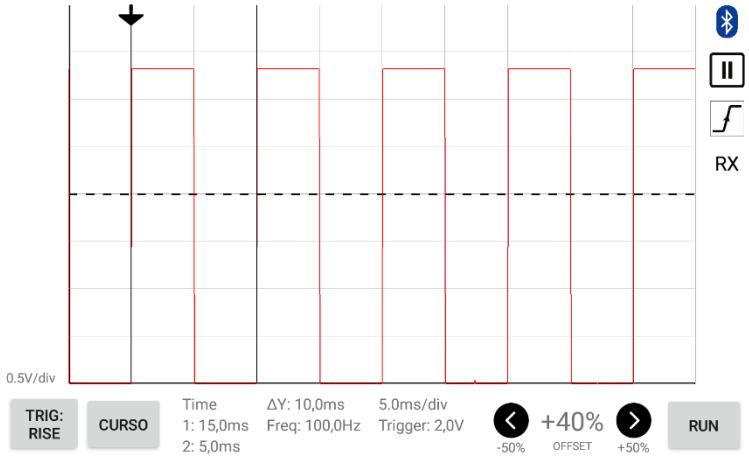

Figura 26: Onda quadrada de 100Hz visualizada no aplicativo com escala de 5ms/divisão.

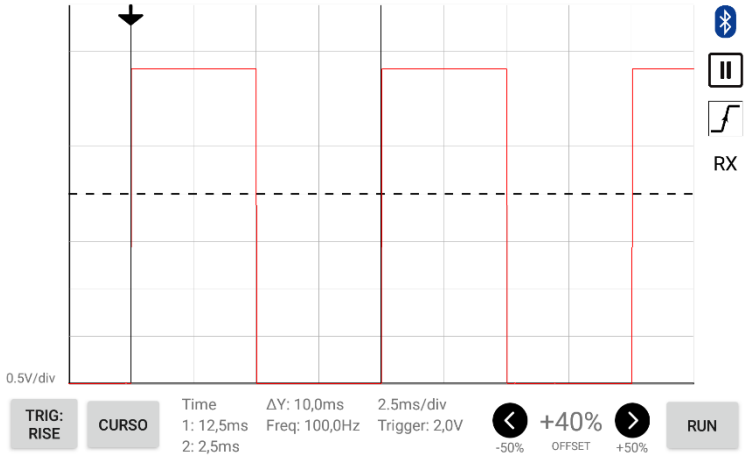

Figura 27: Onda quadrada de 100Hz visualizada no aplicativo com escala de 2,5ms/divisão.

4.1.3 Frequência de 500Hz

A Figura 28 mostra a leitura de uma onda quadrada de 500Hz no osciloscópio.

A Figura 29 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 1ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,01% no eixo do tempo.

A Figura 30 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 500µs por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,01% no eixo do tempo.

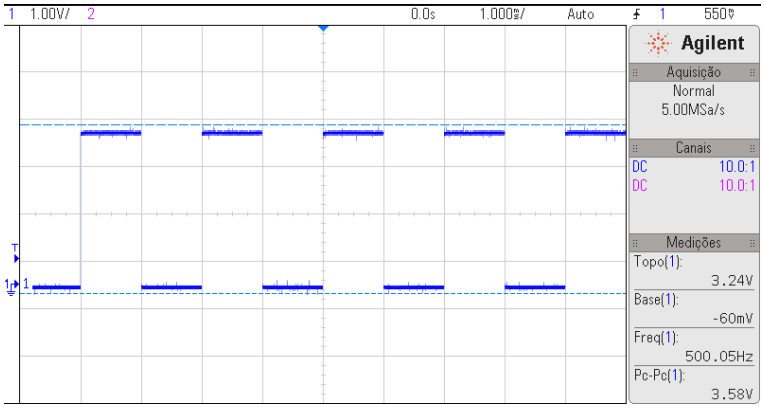

Figura 28: Onda quadrada de 500Hz visualizada no osciloscópio.

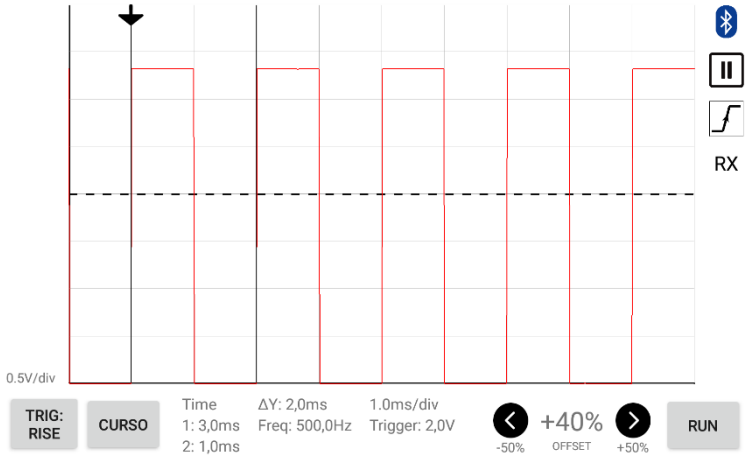

Figura 29: Onda quadrada de 500Hz visualizada no aplicativo com escala de 1ms/divisão.

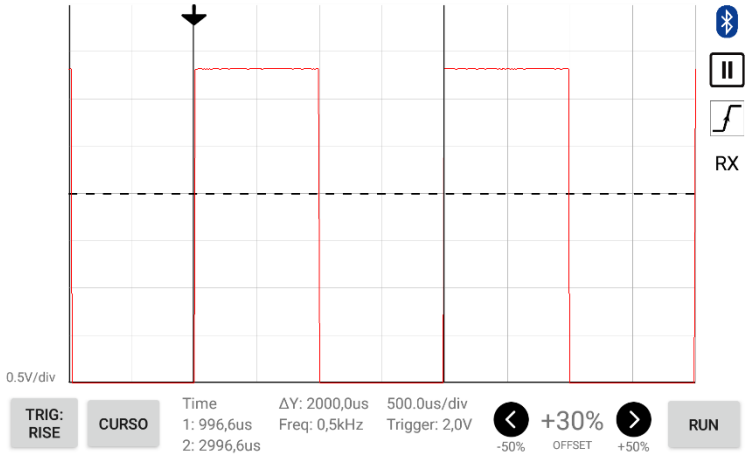

Figura 30: Onda quadrada de 500Hz visualizada no aplicativo com escala de 500µs/divisão.

4.1.4 Frequência de 1000Hz

A Figura 31 mostra a leitura de uma onda quadrada de 1000Hz no osciloscópio.

A Figura 32 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250µs por divisão, com os cursores de tempo aferindo o período de dois ciclos desta. O erro obtido nesta escala com esta onda foi de -0,36% no eixo do tempo.

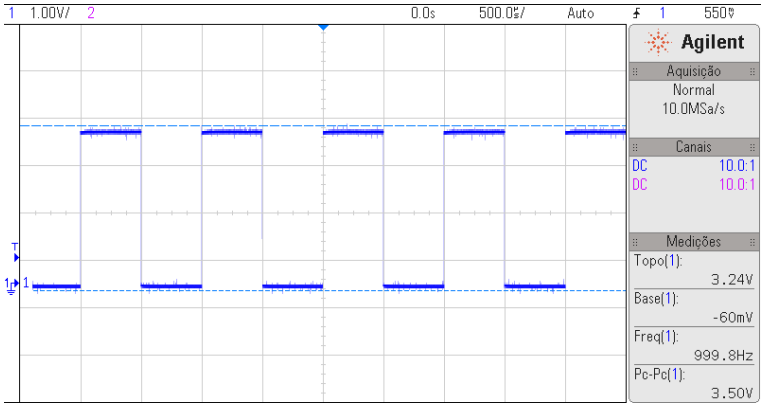

Figura 31: Onda quadrada de 1000Hz visualizada no osciloscópio.

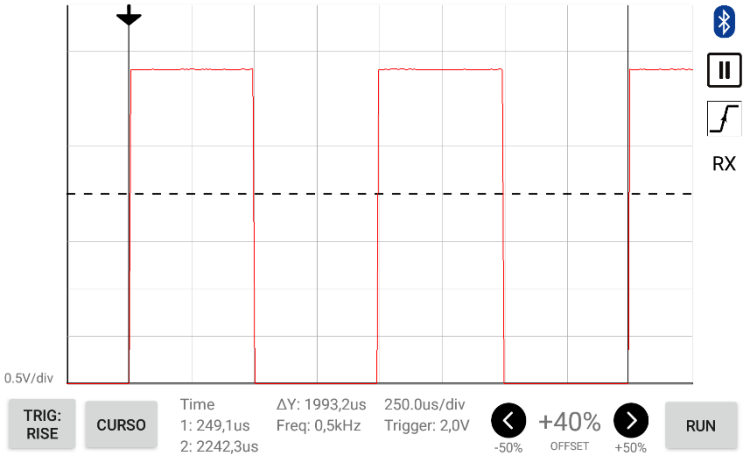

Figura 32: Onda quadrada de 1000Hz visualizada no aplicativo com escala de 250µs/divisão.

4.1.5 Frequência de 5000Hz

A Figura 33 mostra a leitura de uma onda quadrada de 5000Hz no osciloscópio.

A Figura 34 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250µs por divisão, com os cursores de tempo aferindo o período de cinco ciclos desta. O erro obtido nesta escala com esta onda foi de -0,51 % no eixo do tempo.

A Figura 35 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 100µs por divisão, com os cursores de tempo aferindo o período de dois ciclos desta. O erro obtido nesta escala com esta onda foi de 0,84% no eixo do tempo.

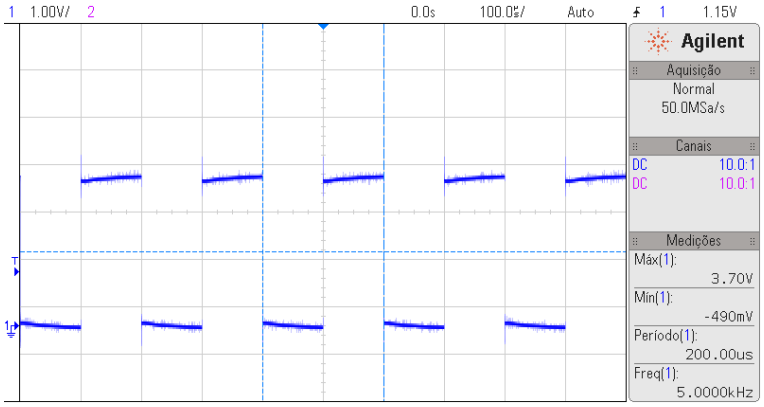

Figura 33: Onda quadrada com frequência de 5000Hz visualizada no osciloscópio.

Figura 34: Onda quadrada de 5000Hz visualizada no aplicativo com escala de 250µs/divisão.

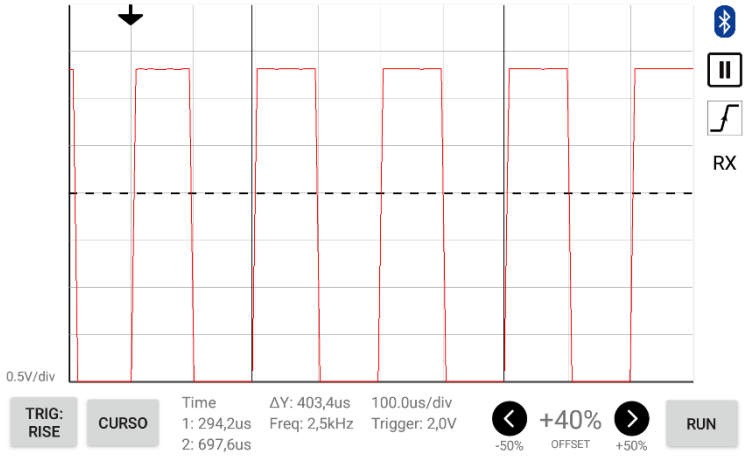

Figura 35: Onda quadrada de 5000Hz visualizada no aplicativo com escala de 100µs/divisão.

4.2 Onda dente de serra

O sinal utilizado foi uma onda do tipo dente de serra, variando de 509mV a 2,51V. Foi utilizado o gerador de função do próprio osciloscópio.

4.2.1 Frequência de 500Hz

A Figura 36 mostra a leitura de uma onda dente de serra de 500Hz no osciloscópio.

A Figura 37 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 1ms por divisão, com os cursores de tensão aferindo a amplitude desta onda, através dos seus pontos máximo e mínimo. O erro obtido nesta escala com esta onda foi de 0,02V (0,40%) no cursor superior e 0,0V (0,0%) no cursor inferior.

A Figura 38 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 1ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,24% no eixo do tempo.

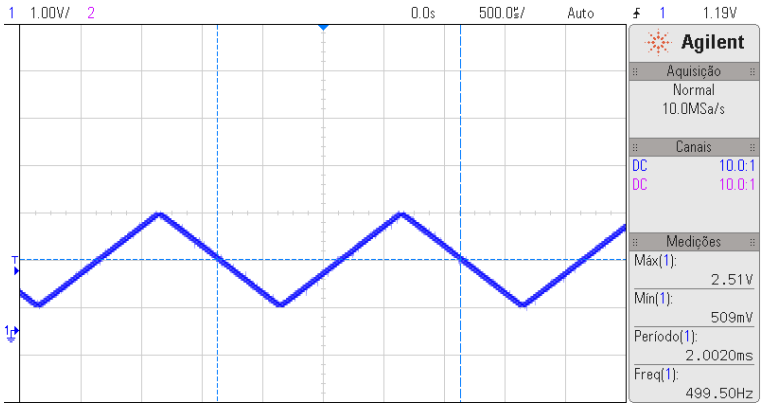

Figura 36: Onda dente de serra com frequência de 500Hz visualizada no osciloscópio.

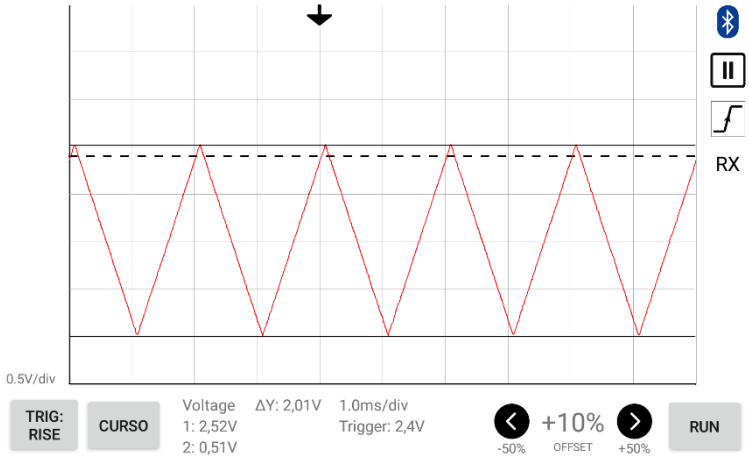

Figura 37: Onda dente de serra de 500Hz visualizada no aplicativo com escala de 1ms/divisão.

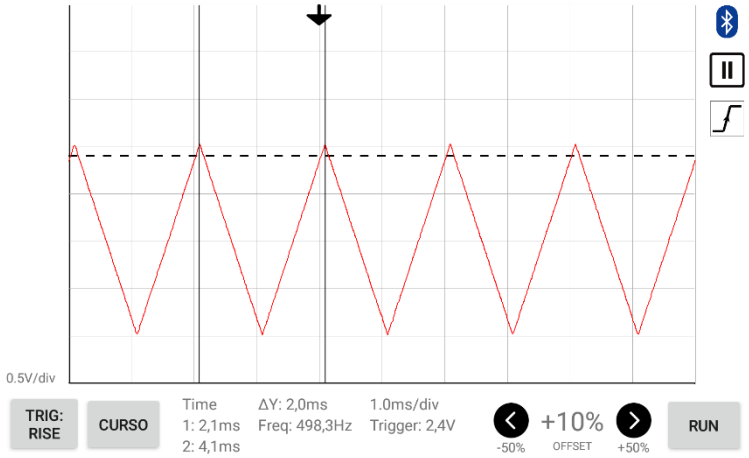

Figura 38: Onda dente de serra de 500Hz visualizada no aplicativo com escala de 1ms/divisão.

4.3 Onda senoidal

O sinal utilizado foi uma onda senoidal, com apenas uma harmônica, variando de 0V a 3,3V. Foi utilizado o gerador de função do próprio osciloscópio.

4.3.1 Frequência de 5Hz

A Figura 39 mostra a leitura de uma onda senoidal de 5Hz no osciloscópio.

A Figura 40 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,06% no eixo do tempo.

A Figura 41 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 100ms por divisão, com os cursores de tempo aferindo o período de dois ciclos desta. O erro obtido nesta escala com esta onda foi de -0,06% no eixo do tempo.

A Figura 42 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 50ms por divisão, com os cursores de tempo aferindo o período de cinco ciclos desta. O erro obtido nesta escala com esta onda foi de -0,06% no eixo do tempo.

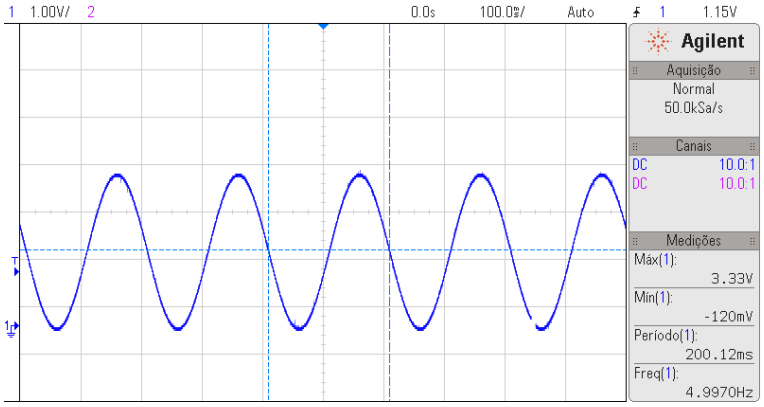

Figura 39: Onda senoidal de 5Hz visualizada no osciloscópio.

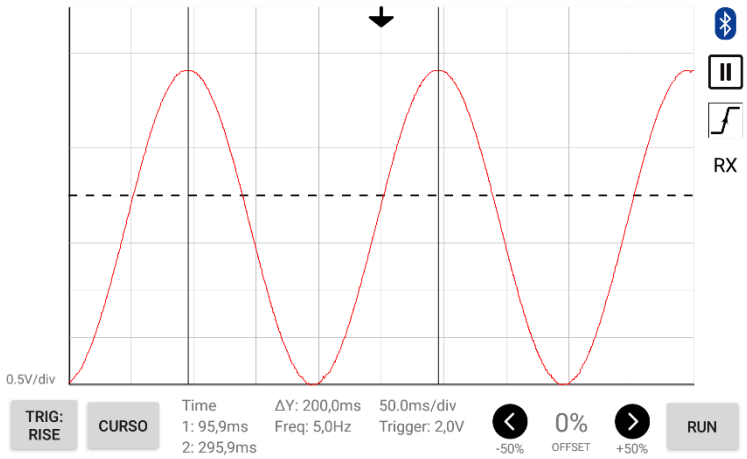

Figura 40: Onda senoidal de 5Hz visualizada no aplicativo com escala de 50ms/divisão.

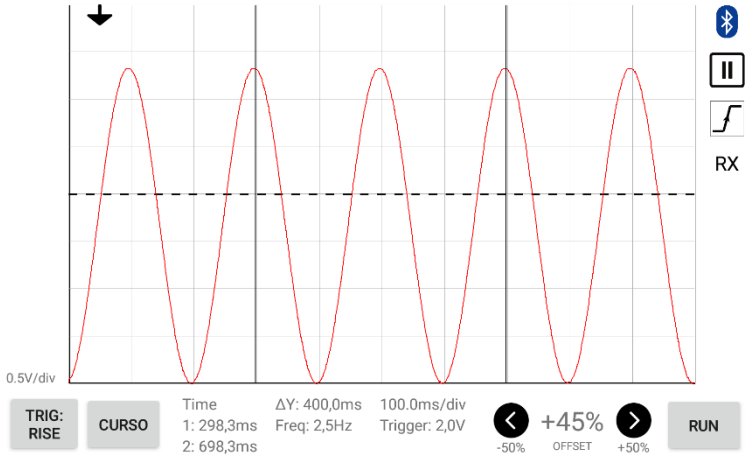

Figura 41: Onda senoidal de 5Hz visualizada no aplicativo com escala de 100ms/divisão.

Figura 42: Onda senoidal de 5Hz visualizada no aplicativo com escala de 250ms/divisão.

4.3.2 Frequência de 50Hz

A Figura 43 mostra a leitura de uma onda senoidal de 50Hz no osciloscópio.

A Figura 44 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 10ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,34% no eixo do tempo.

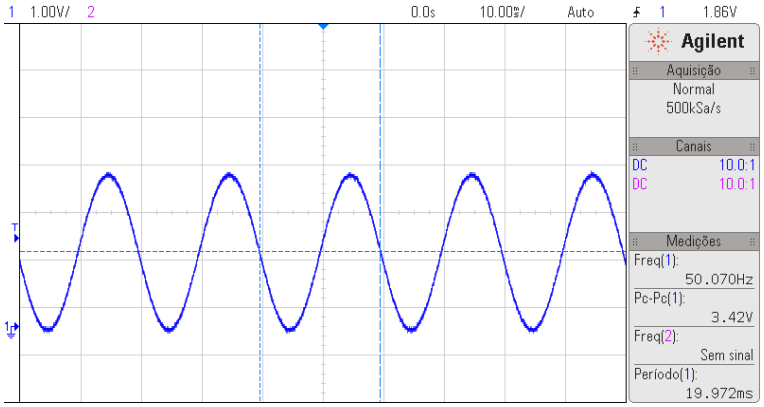

Figura 43: Onda senoidal de 50Hz visualizada no osciloscópio.

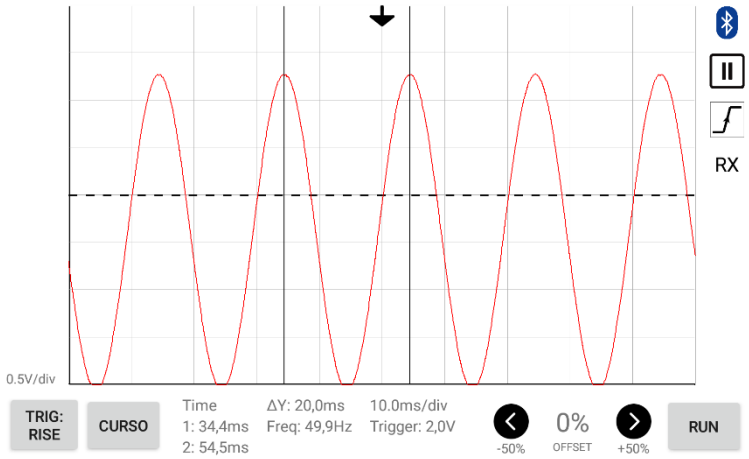

Figura 44: Onda senoidal de 50Hz visualizada no aplicativo com escala de 10ms/divisão.

4.3.3 Frequência de 100Hz

A Figura 45 mostra a leitura de uma onda senoidal de 100Hz no osciloscópio.

A Figura 46 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 5ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,14% no eixo do tempo.

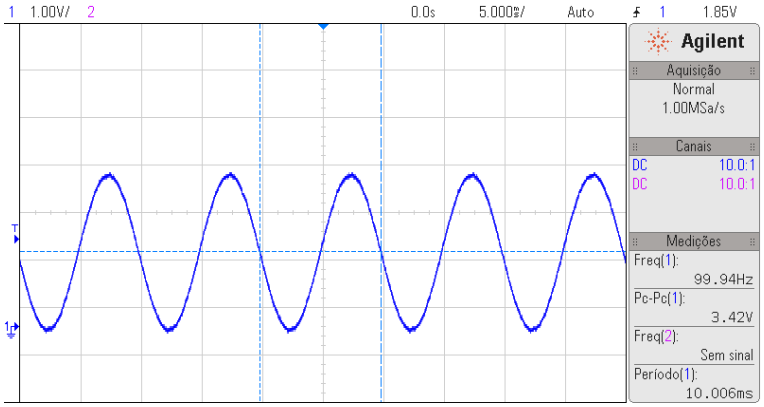

Figura 45: Onda senoidal de 100Hz visualizada no osciloscópio.

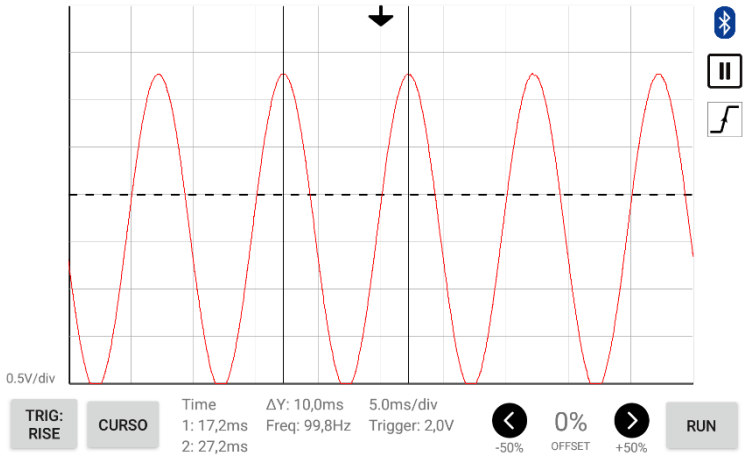

Figura 46: Onda senoidal de 100Hz visualizada no aplicativo com escala de 5ms/divisão.

4.3.4 Frequência de 500Hz

A Figura 47 mostra a leitura de uma onda senoidal de 500Hz no osciloscópio.

A Figura 48 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 2,5ms por divisão, com os cursores de tempo aferindo o período de dois ciclos desta. O erro obtido nesta escala com esta onda foi de 0,09% no eixo do tempo.

A Figura 49 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 1ms por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de 0,19% no eixo do tempo.

A Figura 50 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 500µs por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,15% no eixo do tempo.

A Figura 51 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250µs por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,49% no eixo do tempo.

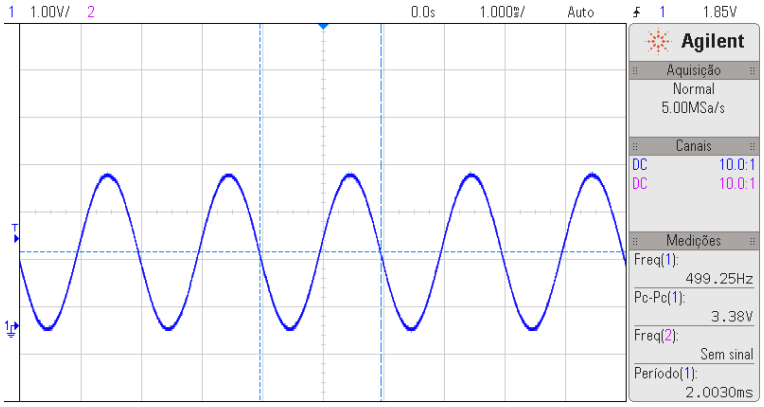

Figura 47: Onda senoidal de 500Hz visualizada no osciloscópio.

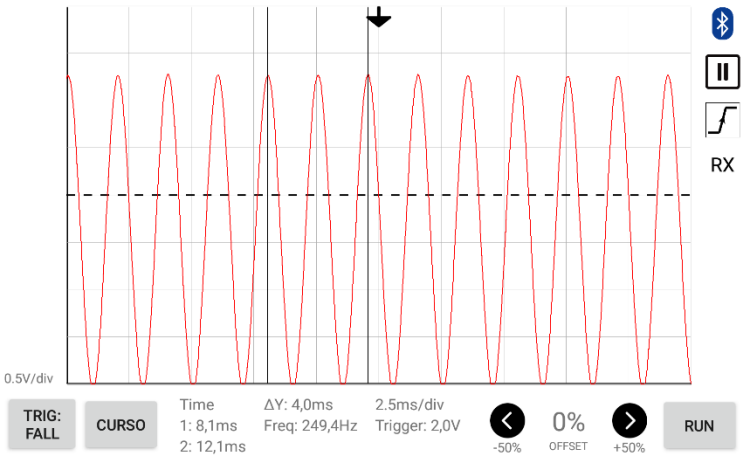

Figura 48: Onda senoidal de 500Hz visualizada no aplicativo com escala de 2,5ms/divisão.

Figura 49: Onda senoidal de 500Hz visualizada no aplicativo com escala de 1ms/divisão.

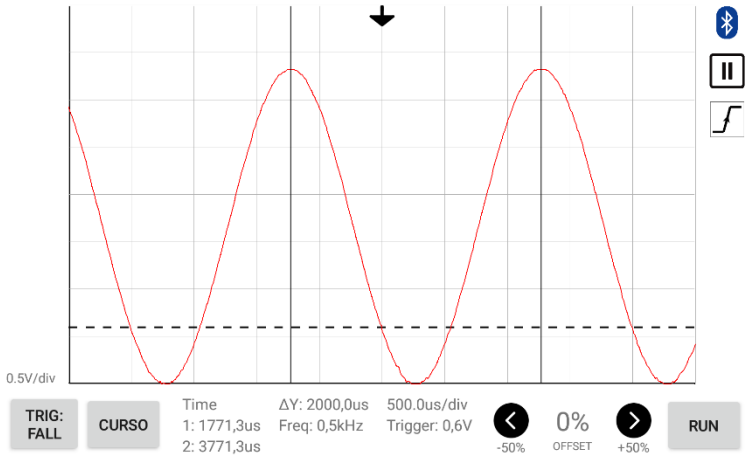

Figura 50: Onda senoidal de 500Hz visualizada no aplicativo com escala de 500µs/divisão.

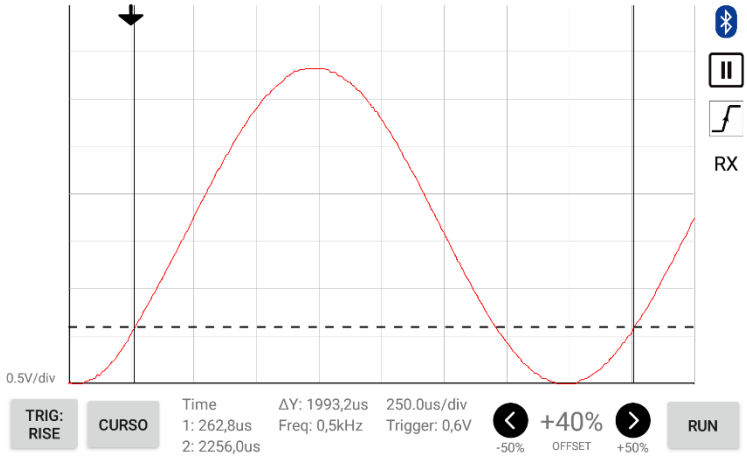

Figura 51: Onda senoidal de 500Hz visualizada no aplicativo com escala de 250µs/divisão.

4.3.5 Frequência de 1000Hz

A Figura 52 mostra a leitura de uma onda senoidal de 1000Hz no osciloscópio.

A Figura 53 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 500µs por divisão, com os cursores de tempo aferindo o período de dois ciclos desta. O erro obtido nesta escala com esta onda foi de -0,09% no eixo do tempo.

A Figura 54 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250µs por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,09% no eixo do tempo.

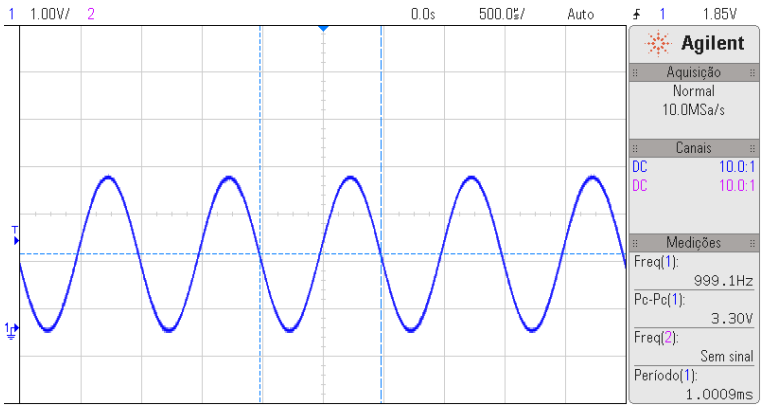

Figura 52: Onda senoidal de 1000Hz visualizada no osciloscópio.

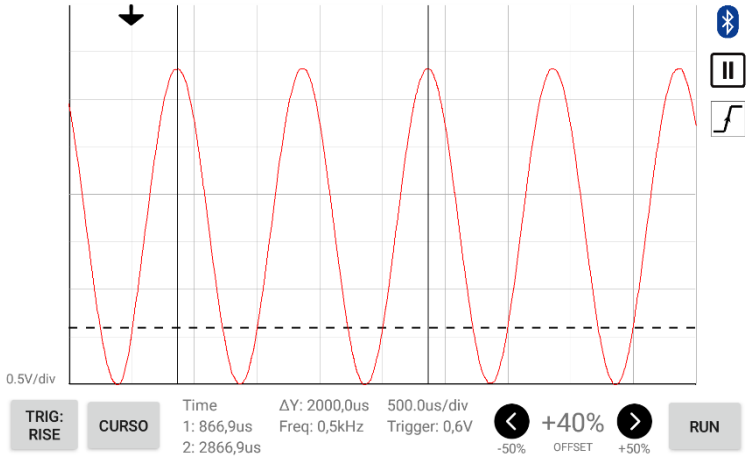

Figura 53: Onda senoidal de 1000Hz visualizada no aplicativo com escala de 500µs/divisão.

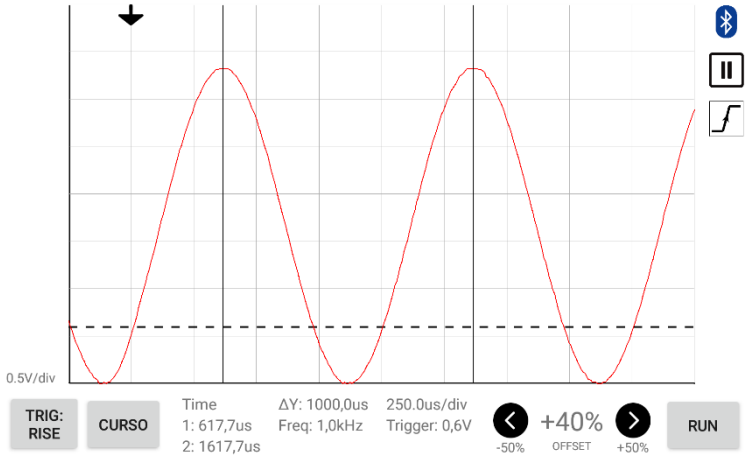

Figura 54: Onda senoidal de 1000Hz visualizada no aplicativo com escala de 250µs/divisão.

4.3.6 Frequência de 2000Hz

A Figura 55 mostra a leitura de uma onda senoidal de 2000Hz no osciloscópio.

A Figura 56 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 100µs por divisão, com os cursores de tempo aferindo o período de um ciclo desta. O erro obtido nesta escala com esta onda foi de -0,08% no eixo do tempo.

A Figura 57 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250µs por divisão, com os cursores de tempo aferindo o período de dois ciclos desta. O erro obtido nesta escala com esta onda foi de -0,02% no eixo do tempo.

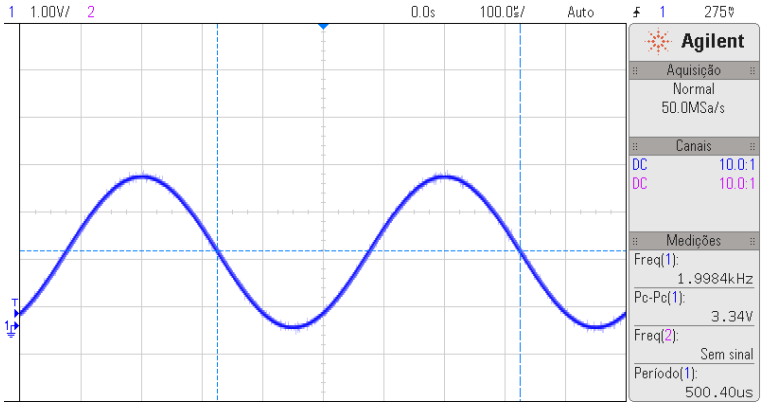

Figura 55: Onda senoidal de 2000Hz visualizada no osciloscópio.

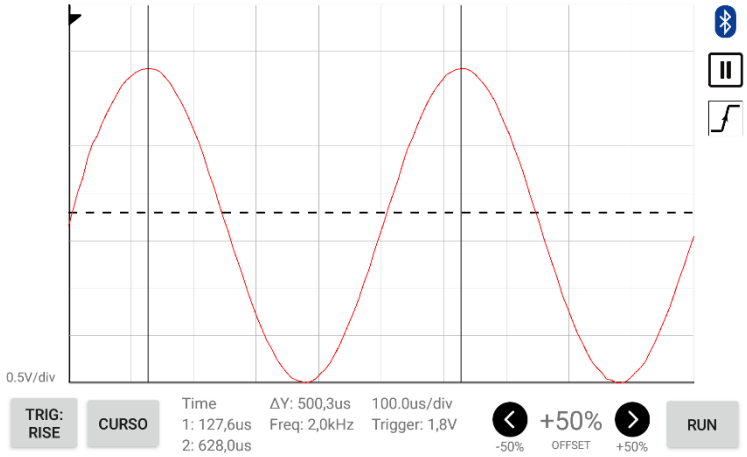

Figura 56: Onda senoidal de 2000Hz visualizada no aplicativo com escala de 100µs/divisão.

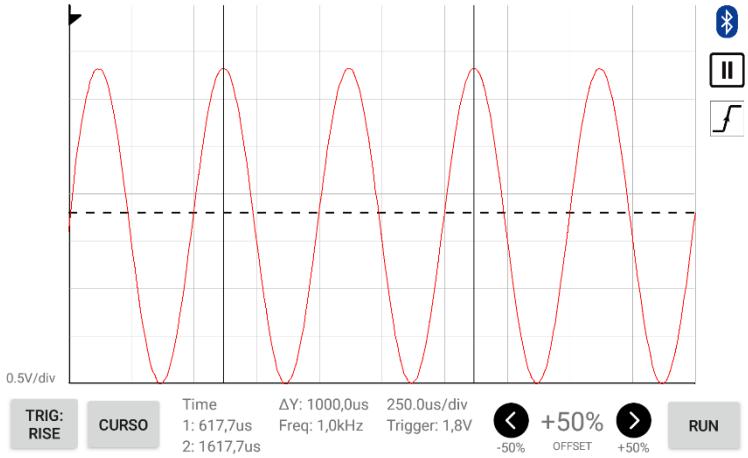

Figura 57: Onda senoidal de 2000Hz visualizada no aplicativo com escala de 250µs/divisão.

4.3.7 Frequência de 5000Hz

A Figura 58 mostra a leitura de uma onda senoidal de 2000Hz no osciloscópio.

A Figura 59 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 250µs por divisão, com os cursores de tempo aferindo o período de cinco ciclos desta. O erro obtido nesta escala com esta onda foi de 0,06% no eixo do tempo.

A Figura 60 mostra esta onda visualizada no aplicativo Android do projeto, na escala de 100µs por divisão, com os cursores de tempo aferindo o período de três ciclos desta. O erro obtido nesta escala com esta onda foi de 0,39% no eixo do tempo.

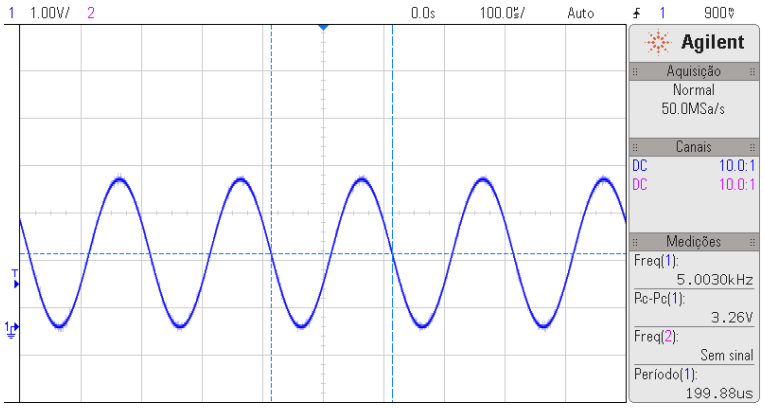

Figura 58: Onda senoidal de 5000Hz visualizada no osciloscópio.

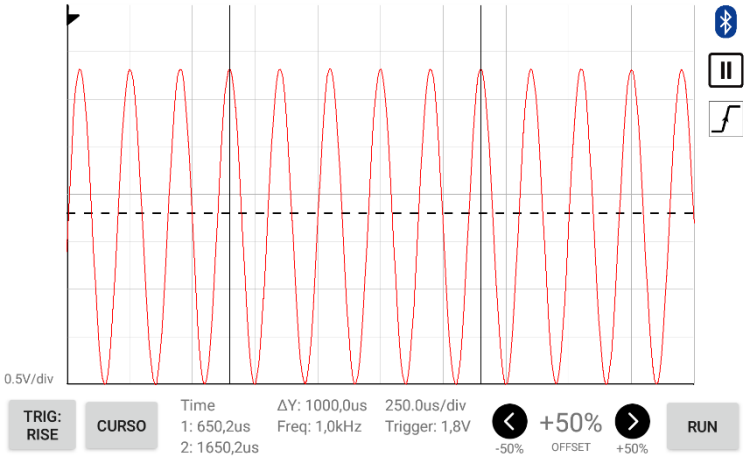

Figura 59: Onda senoidal de 5000Hz visualizada no aplicativo com escala de 250µs/divisão.

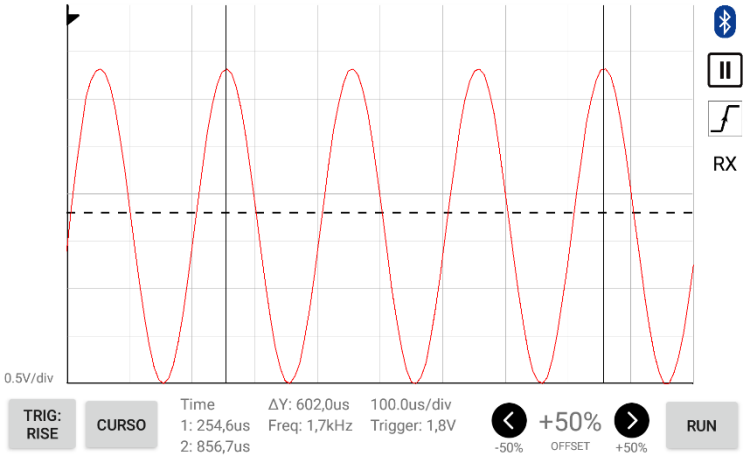

Figura 60: Onda senoidal de 5000Hz visualizada no aplicativo com escala de 100µs/divisão.

4.4 Carregamento de um capacitor

O sinal capturado foi a curva da tensão de um capacitor em um circuito RC ao ser alimentado, estando previamente descarregado. O circuito utilizado pode ser visto na Figura 61.

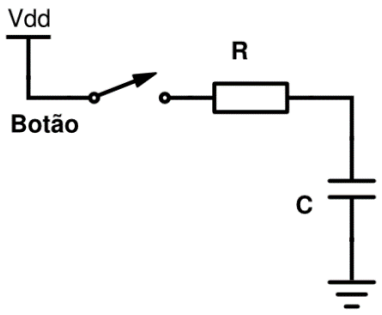

Figura 61: Circuito RC com botão utilizado para capturar o carregamento do capacitor.

Para obter uma prévia do tempo de carregamento do capacitor, pode-se utilizar a formula:

$$V_C = V_S(1 - e^{\frac{-t}{RC}})$$

Onde:

- V_C é a tensão no capacitor. Neste caso será considerada a tensão de 3V;
- V_S é a tensão de alimentação. Neste caso foi utilizada a tensão de alimentação do microcontrolador, 3,3V;
- t é o tempo desde a aplicação da tensão de alimentação;
- RC é a constante de tempo do circuito. Neste caso utilizamos um resistor de resistência R=1kΩ e um capacitor com capacitância C=1µF.

Assim, resolvendo a equação para t, tem-se que o tempo gasto para a tensão no capacitor chegar a 3V é de 2,40ms. A Figura 62 mostra o sinal obtido no projeto e em um osciloscópio, onde o tempo mensurado foi de 2,64ms. Este erro de 10% para mais pode ser atrelado ao fato de terem sido utilizados componentes de baixa precisão no circuito.

A Figura 63 mostra a curva da tensão do capacitor sendo carregado visualizada no aplicativo Android do projeto, na escala de 1ms por divisão, com os cursores de tempo aferindo o tempo para esta atingir 3V. Pode ser visto que o tempo de subida da tensão do capacitor de 0V para 3V mensurado pelo dispositivo desenvolvido foi de 2,6ms. Considerando o tempo mensurado pelo osciloscópio, 2,64ms, tem-se um erro de 1,5% para menos.

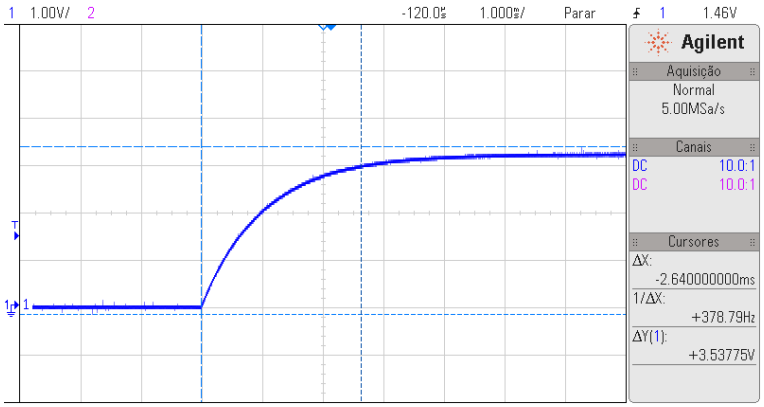

Figura 62: Curva da tensão do capacitor sendo carregado visualizada no osciloscópio.

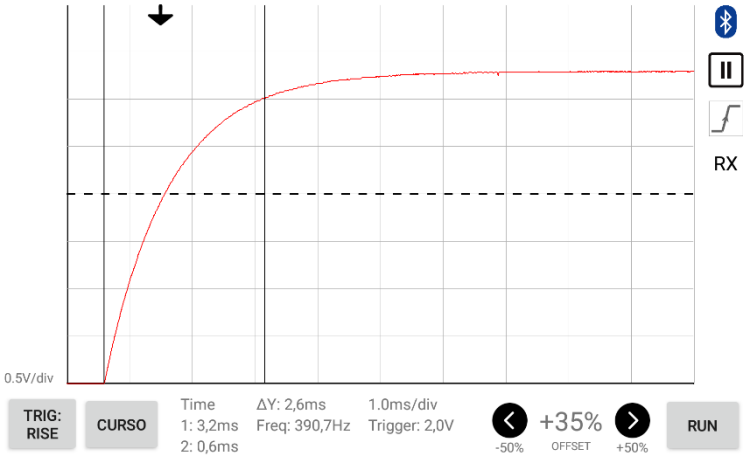

Figura 63: Curva da tensão do capacitor sendo carregado visualizada no aplicativo.

4.5 *Trem de pulsos*

O sinal capturado foi um trem de pulsos gerado por um outro microcontrolador igual ao utilizado no projeto. O sinal simula um contador binário de 8 bits, onde o primeiro pulso representa o bit mais significativo e o oitavo pulso representa o menos significativo. Com este sinal será possível testar também a variação do período de *holdoff*.

Os quatro bits mais significativos foram fixados em nível alto (“1111”) para facilitar a visualização, a duração de cada pulso é de 1ms e o espaço de tempo entre um bit menos significativo e o bit mais significativo do próximo número é de 20ms.

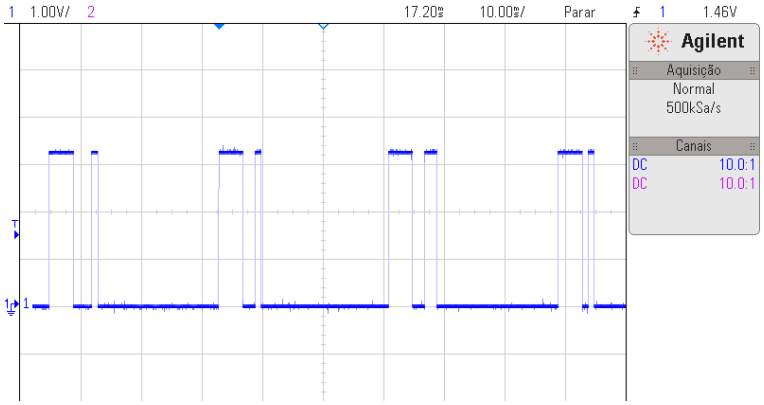

Figura 64: Trem de pulsos visualizado no osciloscópio.

Quando conectou-se os cabos do gerador de onda no dispositivo, ocorreu o efeito descrito na seção 3.2.5 - Variação do período de *holdoff*, onde o evento de *trigger* não ocorria sempre no mesmo bit do sinal, como pode ser visto nas figuras Figura 65 e Figura 66. Pode ser visto que naa primeira o *trigger* ocorre no primeiro bit do sinal e na segunda este ocorre em seu último bit.

Após ajustar o controle de *holdoff* para um valor acima de 5/8, o evento de *trigger* sempre ocorreu no primeiro bit do sinal, demonstrando assim a utilidade deste controle.

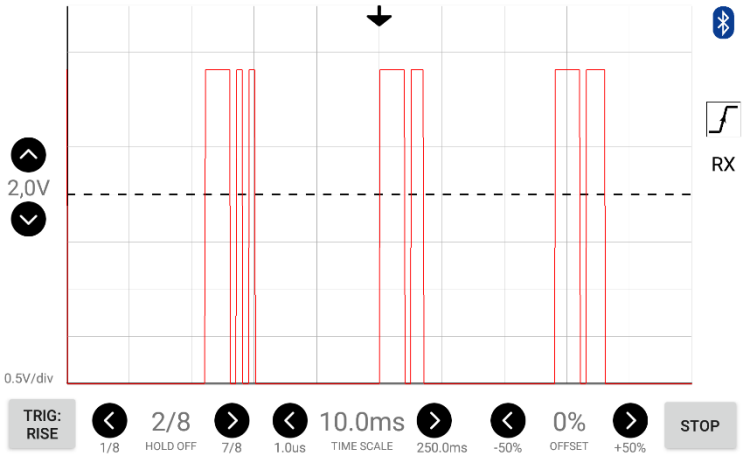

Figura 65: Sinal do trem de pulsos visualizado no aplicativo.

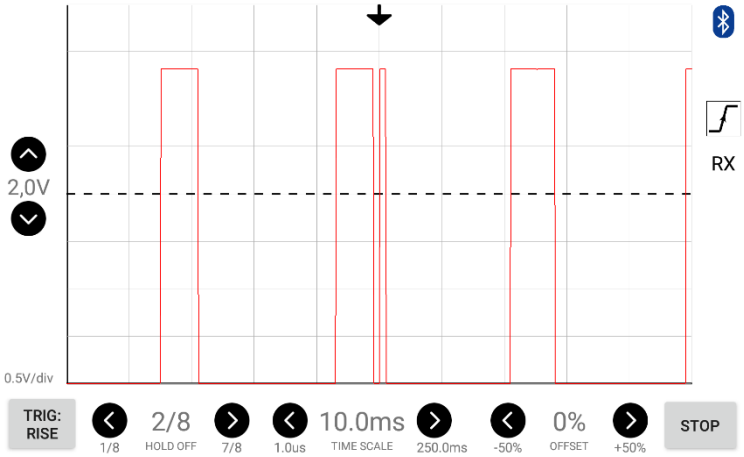

Figura 66: Sinal do trem de pulsos visualizado no aplicativo, em um momento diferente.

4.6 Resumo e análise dos resultados

A Tabela 6 agrupa todos os resultados obtidos nas leituras de frequências das ondas quadrada, dente de serra e senoidal.

Tabela 6: Resultados obtidos nas leituras de frequência.

Forma de onda	**Frequência nominal (Hz)**	**Escala (por divisão)**	**Frequência lida no osciloscópio (Hz)**	**Frequência mensurada (Hz)**	**Erro**
Quadrada	50	10ms	50,00	50,20	-0,40%
		5ms	50,00	50,20	-0,40%
		2,5ms	50,00	50,00	0,00%
	100	5ms	100,00	100,00	0,00%
		2,5ms	100,00	100,00	0,00%
	500	1ms	500,05	500,00	0,01%
		500µs	500,05	500,00	0,01%
	1000	250µs	999,80	1003,41	-0,36%
	5000	250µs	5000,00	5025,63	-0,51%
		100µs	5000,00	5130,84	-0,84%
Dente de serra	500	1ms	499,50	498,30	0,24%
Senoidal	5	50ms	5,00	5,00	-0,06%
		100ms	5,00	5,00	-0,06%
		250ms	5,00	5,00	-0,06%
	50	10ms	50,07	49,90	0,34%
	100	5ms	99,94	99,80	0,14%
	500	2,5ms	499,25	498,80	0,09%
		1ms	499,25	498,30	0,19%
		500µs	499,25	500,00	-0,15%
		250µs	499,25	501,71	-0,49%
	1000	500µs	999,10	1000,00	-0,09%
		250µs	999,10	1000,00	-0,09%
	2000	250µs	1998,40	1998,80	-0,02%
		100µs	1998,40	2000,00	-0,08%
	5000	250µs	5003,00	5000,00	0,06%
		100µs	5003,00	4983,39	0,39%
				Máximo (módulo)	**0,84%**
				Médio (módulo)	**0,20%**

O erro médio obtido na escala de tempo foi de 0,20%. Comparando este valor com a precisão do osciloscópio comercial utilizado, ±0,16%, podemos considerar este

resultado adequado para a utilização do projeto. O erro obtido é em grande parte decorrente do posicionamento manual dos cursores utilizando a interface *touchscreen* da tela.

Quanto à escala da tensão, obteve-se um erro médio de 0,02V nas medidas feitas na onda dente de serra. Pode-se atrelar grande parte deste erro ao posicionamento manual dos cursores utilizando a interface touchscreen da tela. Quando comparado com o erro do osciloscópio comercial utilizado, que é de 0,01V na escala de 1V por divisão, podemos considerar o erro obtido no projeto como satisfatório.

5 CONCLUSÕES

Sabe-se que o preço dos osciloscópios são altos o suficiente para não serem um equipamento comum entre estudantes ou profissionais com seus projetos pessoais. Sabendo que uma grande parcela da população possui um *smartphone* ou *tablet* Android, há espaço para criar um produto com funcionalidades baseadas nas de um osciloscópio, podendo até possuir uma taxa de aquisição inferior, que, utilizando-se da tela e bateria do aparelho Android, possa cobrir esta necessidade tendo um baixo custo.

Um piloto deste produto pode ser desenvolvido neste projeto, mostrando que é possível desenvolvê-lo utilizando um microcontrolador, que possui baixo custo e que conecta-se a um dispositivo Android, o qual muitas pessoas já possuem.

O custo de materiais deste projeto, de US$35, pode ser reduzido se feita uma produção em série deste. É válido, entretanto, ressaltar que este ainda não é considerado um produto final e alguns incrementos ainda são necessários para tal, sendo alguns listados na seção de trabalhos futuros a seguir.

O dispositivo desenvolvido pode ser utilizado para visualizar sinais com uma taxa de erro médio de 0,2% no eixo do tempo e 0,02V no eixo da tensão com uma taxa máxima de aquisição de 150 mil amostras por segundos. Se comparadas à taxa de erro do osciloscópio comercial utilizado nos testes desse projeto e com a diferença de preço entre os dois, podemos considerar a taxa de erro obtida como aceitável para o uso do projeto no ambiente de aprendizado.

Com seu potencial baixo preço, se adotado como material individual de aula para estudantes de cursos técnicos e superiores de áreas que se utilizam desse tipo de instrumentação, este projeto tem um grande potencial para trazer resultados positivos ao aprendizado dos alunos, que terão uma ferramenta a mais para desenvolverem seus projeto de aula e pessoais em casa.

5.1 Trabalhos futuros

Abaixo encontra-se uma lista de funcionalidades sugeridas para uma eventual continuação de implementação do projeto:

- Implementação de circuitos de *step-up* para alterar as tensões de referência do módulo ADC e, assim, possibilitar a variação da escala de tensão para faixas diferentes da faixa de alimentação do microcontrolador, além de circuitos de proteção contra sobretensão na entrada do ADC;
- Implementação de funções matemáticas no sinal lido, como a transformada de Fourier, por exemplo;
- Projeto de uma PCI para agrupar os componentes e permitir o uso de pontas de prova, além de uma caixa de proteção;
- Adição de um módulo de comunicação WiFi e integração com uma interface Web.

6 REFERÊNCIAS BIBLIOGRÁFICAS

AGILENT. InfiniiVision 2000 X-Series Oscilloscopes Data Sheet. Disponível em <http://cp.literature.agilent.com/litweb/pdf/5990-6618EN.pdf>. Acesso em 25 de jun. de 2016.

BIAGORI, André A. Desenvolvimento de sistema para aquisição e acesso remoto a medidas elétricas por meio de dispositivos móveis e internet. Trabalho de Conclusão de Curso. Universidade de São Paulo. São Carlos, 2014.

BLUETOOTH SIG, INC. Serial Port Profile | Bluetooth Development Portal. Disponível em <https://developer.bluetooth.org/TechnologyOverview/Pages/SPP.aspx>. Acesso em 14 de fev. de 2016.

IDC. IDC Worldwide Quarterly Mobile Phone Tracker. Disponível em <http://www.idc.com/tracker/showproductinfo.jsp?prod_id=37>. Acesso em 24 de jun. de 2016.

ITEAD STUDIO. HC-05 Bluetooth to Serial Port Module. Disponível em <http://www.robotshop.com/media/files/pdf/rb-ite-12-bluetooth_hc05.pdf>. Acesso em 10 de fev. de 2016.

RADIO-ELECTRONICS. Oscilloscope Trigger | Scope Triggering Tutorial. Disponível em <http://www.radio-electronics.com/info/t_and_m/oscilloscope/oscilloscope-trigger.php>. Acesso em 16 de fev. de 2016.

Salla, Gabriel C. Telemetria sem fio com armazenamento e exportação dos dados de unidade de sensoriamento via ´´. Trabalho de Conclusão de Curso. Universidade de São Paulo. São Carlos, 2015.

TEKTRONIX. Triggering Fundamentals. Disponível em <www.tek.com/dl/55W_17291_6_0.pdf>. Acesso em 20 de fev. de 2016.

TEXAS INSTRUMENTS. Application Note: Using the Stellaris Microcontroller Analog-to-Digital Converter (ADC). 2013a. Disponível em <http://www.ti.com/lit/an/spma028/spma028.pdf>. Acesso em 16 de março de 2016.

TEXAS INSTRUMENTS. ADC Oversampling Techniques for Stellaris Family Microcontrollers. 2013b. Disponivel em <http://www.ti.com/lit/an/spma001a/spma001a.pdf>. Acesso em 16 de março de 2016.

TEXAS INSTRUMENTS. Getting Started with the Stellaris EK-LM4F120XL LaunchPad Workshop. 2013c. Disponível em <http://www.ly3ci.com/Stellaris_LaunchPad_Start_files/StellarisLaunchPadWorkbook.pdf>. Acesso em 10 de fev. de 2016.

TEXAS INSTRUMENTS. Stellaris LM4F120H5QR Microcontroller Datasheet. 2013d. Disponível em: <http://www.ti.com/tool/EK-LM4F120XL#Technical Documents>. Acesso em 28 de maio de 2016.

APÊNDICE A – Código do microcontrolador ARM Cortex LM4

O código fonte do *software* do microcontrolador pode ser encontrado no repositório GitHub do autor:

https://github.com/jeajjr/tcc-arm/

APÊNDICE B – Código do aplicativo para Android

O código fonte aplicativo para Android desenvolvido pode ser encontrado no repositório GitHub do autor:

https://github.com/jeajjr/tcc-android/

APÊNDICE C – Comandos utilizados para configurar o módulo Bluetooth HC-05

Os comandos listados na Tabela 7 foram utilizados para configurar o módulo Bluetooth HC-05. Para entrar no modo de configuração, foi necessário ligar o módulo pressionando-se o botão de configuração, que pode ser visto na Figura 17. Neste modo, a comunicação serial com o módulo para envio dos comandos é feita a 38400bps, com um bit de parada e sem paridade.

Tabela 7: Comandos utilizados para configurar o módulo Bluetooth HC-05.

Descrição	Comando enviado	Resposta
Teste de comunicação	AT	OK
Ler SSID	AT+NAME?	HC-2010-06-01 OK
Gravar SSID	AT+NAME=TCC	TCC OK
Verificar configuração da porta serial	AT+UART?	38400/8-N-1 OK
Configurar porta serial para 460800bps, um bit de parada e sem paridade	AT+UART=460800,0,0	460800/8-N-1 OK

ANEXO A – Diagrama de conexões do kit de desenvolvimento Stellaris LaunchPad

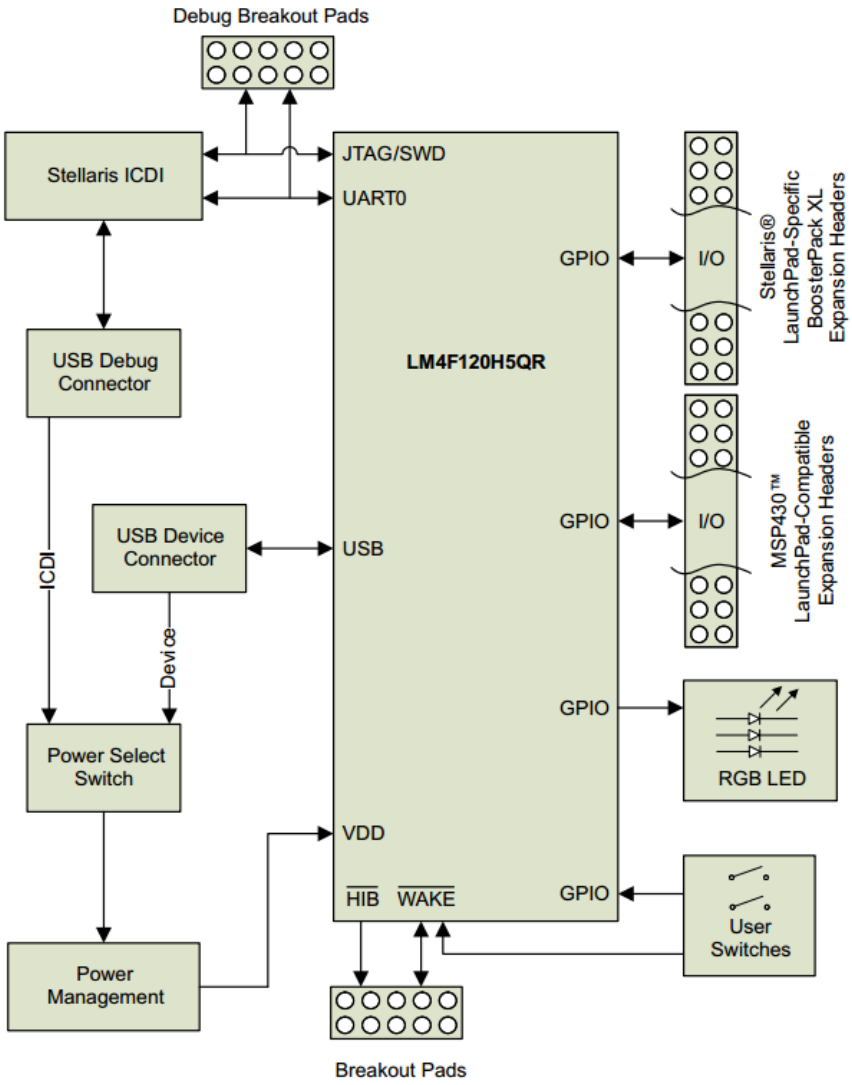

Figura 67: Diagrama de conexões do kit de desenvolvimento Stellaris LaunchPad.

Fonte: Manual do kit.

ANEXO B – Especificações técnicas do microcontrolador Texas Instruments ARM LM4F120H5QR

Tabela 8: Especificações técnicas do microcontrolador Texas Instruments ARM LM4F120H5QR contidas em seu *datasheet.*

Feature	Description
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with StellarisWare software
Communication Interfaces	
Universal Asynchronous Receivers/Transmitter (UART)	Eight UARTs
Synchronous Serial Interface (SSI)	Four SSI modules
Inter-Integrated Circuit (I2C)	Four I2C modules with four transmission speeds including high-speed mode
Controller Area Network (CAN)	CAN 2.0 A/B controllers
Universal Serial Bus (USB)	USB 2.0 Device
System Integration	
Micro Direct Memory Access (µDMA)	ARM PrimeCell 32-channel configurable µDMA controller
General-Purpose Timer (GPTM)	Six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
Watchdog Timer (WDT)	Two watchdog timers
Hibernation Module (HIB)	Low-power battery-backed Hibernation module
General-Purpose Input/Output (GPIO)	Six physical GPIO blocks
Analog Support	
Analog-to-Digital Converter (ADC)	Two 12-bit ADC modules with a maximum sample rate of one million samples/second
Analog Comparator Controller	Two independent integrated analog comparators
Digital Comparator	16 digital comparators
JTAG and Serial Wire Debug (SWD)	One JTAG module with integrated ARM SWD
Package	64-pin LQFP
Operating Range	Industrial (-40°C to 85°C) temperature range

ANEXO C – Especificações do Osciloscópio Agilent DSO-X 2002A

Tabela 9: Especificações do Osciloscópio Agilent DSO-X 2002A contidas em seu *datasheet*. [AGILENT]

Specification overview											
		2002A		**2004A**		**2012A**		**2014A**		**2022A**	**2024A**
Bandwidth [1] (–3 dB)		70 MHz				100 MHz				200 MHz	
Calculated rise time (10 to 90%)		≤ 5 ns				≤ 3.5 ns				≤ 1.75 ns	
Input channels	DSOX	2		4		2		4		2	4
	MSOX	2 + 8		4 + 8		2 + 8		4 + 8		2 + 8	4 + 8
Maximum sample rate [1]		2 GSa/s half-channel interleaved, 1 GSa/s per channel									
Maximum memory depth		100 kpts per channel (standard), 1 Mpt per channel (optional with DSOX2MEMUP)									
Display size and type		8.5-inch WVGA with 64 levels of intensity grading									
Waveform update rate		50,000 waveforms per second									
WaveGen – built-in function generator (Specifications are typical)											
Frequency	– Sine wave and ramp accuracy: – 130 ppm (frequency < 10 kHz) – 50 ppm (frequency > 10 kHz) – Square wave and pulse accuracy: – [50+frequency/200] ppm (frequency < 25 kHz) – 50 ppm (frequency ≥ 25 kHz) – Resolution: 0.1 Hz or 4 digits, whichever is larger										
Amplitude	– Range: – 20 mVpp to 5 Vpp into Hi-Z – 10 mVpp to 2.5 Vpp into 50 Ω – Resolution: 100 µV or 3 digits, whichever is larger – Accuracy: 2% (frequency = 1 kHz)										
Vertical system analog channels											
Input coupling		AC, DC									
Input sensitivity range		1 mV/div to 5 V/div [2]									
Input impedance		1 MΩ ± 2% (11 pF)									
Vertical resolution		8 bits (measurement resolution is 12 bits with averaging)									
Dynamic range		± 8 divisions from center screen									
Maximum input voltage		300 Vrms, 400 Vpk; transient overvoltage 1.6 kVpk									
		With N2862B or N2863B 10:1 probe: 300 Vrms Frequency de-rating (assumes sine wave input): 400 Vpk until 40 kHz. Then de-rates at 20 db/dec until 6 Vpk									
DC vertical accuracy		± [DC vertical gain accuracy + DC vertical offset accuracy + 0.25% full scale] [2]									
DC vertical gain accuracy [1]		± 3% full scale (≥ 10 mV/div); ± 4% full scale (< 10 mV/div) [2]									
DC vertical offset accuracy		± 0.1 div ± 2mV ± 1% of offset setting									
Channel-to-channel isolation		≥ 40 dB from DC to maximum specified bandwidth of each model									
Position/offset range	1 MΩ	1 mV to 200 mV/div: ± 2 V, > 200 mV to 5 V/div: ± 50 V									
Hardware bandwidth limits		Approximately 20 MHz (selectable)									
Horizontal system analog channels											
		2002A		**2004A**		**2012A**		**2014A**		**2022A**	**2024A**
Time base range		5 ns/div to 50 s/div								2 ns/div to 50 s/div	
Horizontal resolution		2.5 ps									
Time base accuracy [1]		25 ppm ± 5 ppm per year (aging)									
Time base delay time range	Pre-trigger	Greater of 1 screen width or 200 µs (400 µs in interleaving mode)									
	Post-trigger	1 s to 500 s									
Channel-to-channel deskew range		± 100 ns									
Δ Time accuracy (using cursors)		± (time base accuracy [1] reading) ± (0.0016 [1] screen width) ± 100 ps									

1. Denotes warranted specifications, all others are typical. Specifications are valid after a 30-minute warm-up period and from ± 10 °C firmware calibration temperature.
2. 1 mV/div and 2 mV/div is a magnification of 4 mV/div setting. For vertical accuracy calculations, use full scale of 32 mV for 1 mV/div and 2 mV/div sensitivity setting.