UNIVERSIDADE FEDERAL DE UBERLÂNDIA
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

PAULO RICARDO DE ALMEIDA SANTOS

IMPLEMENTAÇÃO DE UM *DATALOGGER* UTILIZANDO DMA E UART

UBERLÂNDIA
2024

PAULO RICARDO DE ALMEIDA SANTOS

IMPLEMENTAÇÃO DE UM *DATALOGGER* UTILIZANDO DMA E UART

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia como requisito para a obtenção do diploma de graduação em Engenharia Eletrônica e de Telecomunicações.

Orientador: Prof. Dr. Josué Silva de Morais

UBERLÂNDIA

2024

PAULO RICARDO DE ALMEIDA SANTOS

IMPLEMENTAÇÃO DE UM *DATALOGGER* UTILIZANDO DMA E UART

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia como requisito para a obtenção do diploma de graduação em Engenharia Eletrônica e de Telecomunicações.
Orientador: Prof. Dr. Josué Silva de Morais

UBERLÂNDIA - MG, 18 de novembro de 2024

Banca Examinadora

Prof. Dr. Josué Silva de Morais – FEELT/UFU (Orientador)

Prof. Me. Éder Alves de Moura – FEELT/UFU (Membro 1)

Prof. Me. Luís Ricardo Cândido Côrtes – FEELT/UFU (Membro 2)

RESUMO

Este trabalho apresenta o desenvolvimento de um *datalogger* de baixo custo utilizando os microcontroladores STM32 e ESP32, integrados a uma interface gráfica em Python. Com o objetivo de oferecer uma alternativa acessível ao uso de osciloscópios de bancada, o projeto visa capturar e analisar sinais elétricos em tempo real, contribuindo para a formação prática de estudantes em Engenharia Elétrica e Eletrônica. A implementação incluiu um filtro *anti-aliasing* para eliminar componentes de alta frequência, garantindo a fidelidade do sinal no processo de conversão analógica-digital. Utilizando o DMA para uma transmissão eficiente dos dados, o sistema alcançou a taxa de amostragem necessária, e a comunicação entre os microcontroladores via UART permitiu a transferência e visualização dos sinais capturados em uma interface gráfica. Os resultados demonstraram que o *datalogger* cumpre os requisitos de precisão e desempenho, destacando-se como uma ferramenta educacional acessível para análise de sinais em ambientes acadêmicos.

Palavras-chave: *Datalogger*, STM32, ESP32, DMA, UART.

ABSTRACT

This work presents the development of a low-cost datalogger using STM32 and ESP32 microcontrollers, integrated with a Python graphical interface. Aimed at providing an affordable alternative to traditional bench oscilloscopes, the project seeks to capture and analyze electrical signals in real time, contributing to the practical training of students in Electrical and Electronic Engineering. The implementation included an anti-aliasing filter to eliminate high-frequency components, ensuring signal fidelity during the analog-to-digital conversion process. By using DMA for efficient data transmission, the system achieved the required sampling rate, and the communication between the microcontrollers via UART allowed for the transfer and visualization of the captured signals in a graphical interface. The results demonstrated that the datalogger meets precision and performance requirements, standing out as an accessible educational tool for signal analysis in academic settings.

Keywords: Datalogger, STM32, ESP32, DMA, UART.

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES

ADC – *Analog-to-Digital Converter*

AO – Amplificador Operacional

CPU – *Central Processing Unit*

DIWO – *Do It With Others*

DIY – *Do It Yourself*

DMA – *Direct Memory Access*

GPIO – *General-Purpose Input/Output*

IDE – *Integrated Development Environment*

IoT – *Internet of Things*

MCU – *Microcontroller Unit*

MISO – *Master Input Slave Output*

MOSI – *Master Output Slave Input*

PWM – *Pulse Width Modulation*

RAM – *Random Access Memory*

SAR – *Successive Approximation Register*

SDK – *Software Development Kit*

SPI – *Serial Peripheral Interface*

SS ou CS – *Chip Select*

UART – *Universal Asynchronous Receiver/Transmitter*

USB – *Universal Serial Bus*

Sumário

1. Introdução

No ramo das engenharias elétrica e eletrônica, um equipamento amplamente utilizado nos estudos é o osciloscópio. O osciloscópio é um instrumento de medição de sinais elétricos, capaz de visualizá-los no domínio do tempo e também de representá-los na forma de ondas.

Com base nesses conceitos, é crucial que os discentes dos cursos de engenharia, especialmente nas áreas elétrica e eletrônica, tenham acesso a esses aparelhos para consolidar os aprendizados adquiridos em sala de aula. No entanto, os osciloscópios de bancada utilizados nos laboratórios têm custos elevados, tanto para a maioria dos estudantes quanto para as instituições de ensino.

Uma forma de equilibrar essa situação é utilizar a técnica DIY (sigla do inglês *Do It Yourself*, que significa "faça você mesmo"), que permite a construção de protótipos de baixo custo, além de aprimorar os conhecimentos teóricos e práticos dos alunos. Devido aos avanços tecnológicos, as técnicas DIY e DIWO (sigla de *Do It With Others*) deram origem ao movimento *Maker*. Esse movimento tecnológico parte do princípio de praticar o que foi aprendido, com o intuito de estimular qualquer pessoa a modificar, consertar ou fabricar seus próprios objetos (CORDEIRO; GUÉRIOS; PAZ, 2019).

Compreendendo a importância do osciloscópio para o aprendizado dos discentes e sabendo que é possível implementar a técnica DIY, surgiu a ideia de desenvolver um osciloscópio utilizando as plataformas STM32 e ESP32 para a captura de sinais elétricos e seu envio para uma interface gráfica no computador ou notebook.

1.1. Motivação

O ensino de engenharia elétrica exige a aplicação prática dos conceitos teóricos aprendidos em sala de aula, e a construção de instrumentos de medição, como o osciloscópio, oferece uma excelente oportunidade de integrar teoria e prática de maneira significativa. Um osciloscópio é uma ferramenta fundamental no campo da eletrônica e engenharia elétrica, sendo utilizado para visualizar e analisar sinais elétricos em tempo real, o que o torna indispensável para a realização de projetos e experimentos avançados na área. No entanto, seu elevado custo tem dificultado sua

aquisição tanto por parte dos alunos quanto, em maior escala, pelas instituições de ensino.

Com o objetivo de solucionar esse problema, este trabalho propõe a construção de um equipamento de baixo custo semelhante a um osciloscópio. Projetos semelhantes já foram desenvolvidos, como o de Shou (2011), que criou um osciloscópio digital portátil de baixo custo. A vantagem desse projeto foi o baixo custo envolvido, mas sua desvantagem foi a conclusão do trabalho apenas com simulações.

Outro exemplo é o trabalho de Cardoso (2016), que desenvolveu um osciloscópio de baixo custo utilizando a plataforma Arduino. No entanto, uma limitação desse projeto foi o uso do Arduino, uma plataforma de cunho didático, mas com barreiras técnicas para um ambiente acadêmico e profissional, onde tecnologias como STM32 ou ESP32 seriam mais adequadas.

Além disso, há o estudo de Predebon e Becker (2018), que desenvolveram um osciloscópio portátil de baixo custo com comunicação Wi-Fi, denominado WifiScope. Nesse projeto foram utilizadas das tecnologias STM32, ESP8266 e a linguagem Python para o desenvolvimento da interface gráfica. Entretanto a plataforma ESP32 utilizada neste presente trabalho é a versão atualizada em relação a plataforma ESP8266.

Com base nesses trabalhos, optou-se pela utilização dos microcontroladores (MCU – *Microcrontroller Unit*) STM32 e ESP32, por serem tecnologias mais recentes e amplamente utilizadas no meio acadêmico. Este projeto também servirá como manual de instruções para que os alunos da graduação do curso de engenharia possam desenvolver seu próprio osciloscópio. Além disso, espera-se que o equipamento desenvolvido com essas plataformas pode ser acoplado ao kit didático (Figura 1), desenvolvido por Soares (2023) e pelo professor doutor Josué Silva de Morais.

Figura 1 - Kit didático

Fonte: Soares (2023).

1.2. Objetivos

1.2.1. Objetivo geral

Este trabalho visa desenvolver um *datalogger* capaz de realizar leituras de sinais de até 40kHz no computador ou notebook, por meio da captação de sinais com as plataformas dos microcontroladores STM32 e ESP32. A proposta é instruir a construção desse *datalogger* aos leitores, preferencialmente aos discentes do curso de engenharia, para que possam desenvolver um osciloscópio de baixo custo e poder analisar os sinais elétricos e suas características. Por fim, esse equipamento será acoplado ao kit didático de instrumentação industrial, conforme mencionado anteriormente.

1.2.2. Objetivos específicos

- Explorar as características e funcionalidades do filtro ativo passa-baixa.
- Explorar as capacidades de desenvolvimento e integração do microcontrolador ESP32, com ênfase no estudo da utilização do protocolo UART e sua conexão com a interface gráfica.
- Explorar as capacidades de desenvolvimento e integração do microcontrolador STM32, com ênfase no estudo da utilização do ADC, DMA, protocolo UART e sua comunicação com a ESP32.
- Explorar o *datalogger* das plataformas STM32 e ESP32.
- Criar um código na linguagem Python para a plotagem de gráficos e resultados.

2. Desenvolvimento teórico

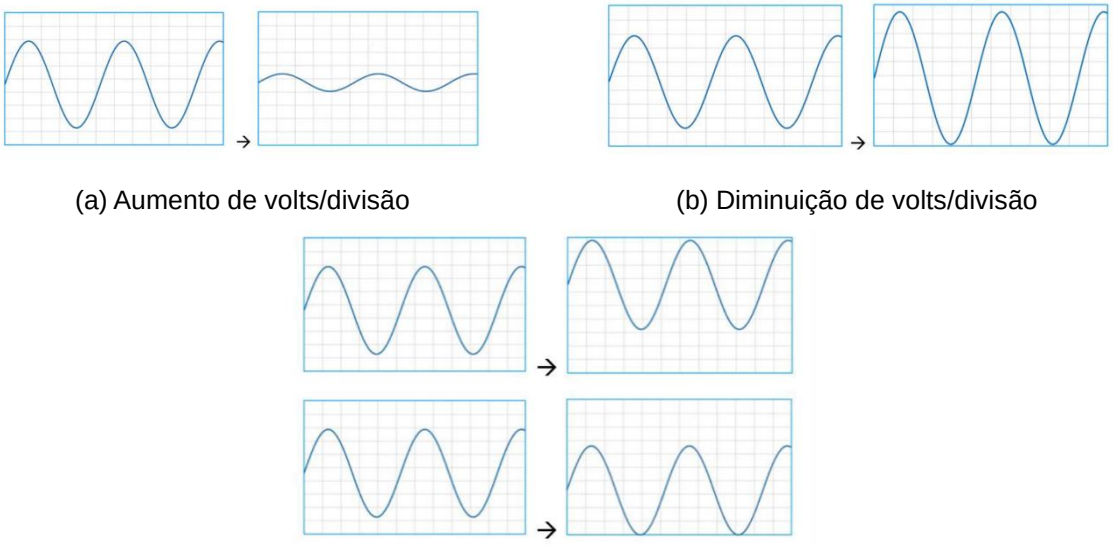
2.1. Osciloscópio

O osciloscópio é um equipamento amplamente utilizado para visualizar graficamente as formas de onda dos sinais eletrônicos e estudá-los. Diversos parâmetros de um sinal elétrico podem ser analisados por meio desse aparelho, como a frequência, amplitude, forma de onda e tensão máxima (MATTEDE, 2024). Osciloscópios modernos possuem funcionalidades adicionais, como a representação de sinais no domínio da frequência (similar a um analisador de espectro), a medição automática de parâmetros como frequência e tensão pico-a-pico, além da análise de barramentos seriais, entre outros (ROHDE & SCHWARZ, 2024).

Esses aparelhos mostram o desenvolvimento da tensão ao longo do tempo, tanto em sinais periódicos quanto em sinais não periódicos, nos modelos mais recentes. Na maioria dos osciloscópios, o eixo horizontal (x) representa o tempo, enquanto o eixo vertical (y) indica a amplitude do sinal, ou seja, o valor da tensão. O sinal exibido pode ser ajustado por três sistemas principais: o sistema horizontal, o sistema vertical e o sistema de gatilhamento (*trigger*) (ROHDE & SCHWARZ, 2024).

O sistema vertical é responsável pela exibição da tensão em função do tempo, além de permitir o posicionamento vertical do sinal e o ajuste da sua visualização por meio do controle de volts/divisão (Figura 2).

Figura 2 - Sistema vertical



(a) Aumento de volts/divisão

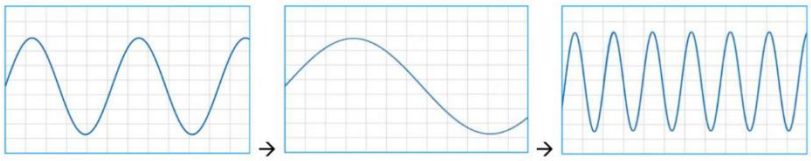
(b) Diminuição de volts/divisão

(c) Mudança do eixo de referência vertical

Fonte: ROHDE & SCHWARZ (2024).

O sistema horizontal, por sua vez, controla a exibição da forma de onda ao longo do tempo, possibilitando ajustar, por meio do controle de segundos/divisão, quantos ciclos serão visualizados na tela do osciloscópio (Figura 3). Além disso, este sistema realiza a amostragem do sinal, tema que será abordado posteriormente.

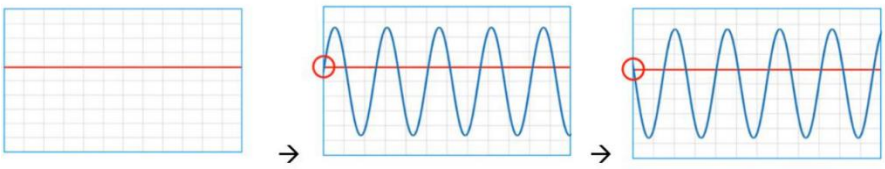
Figura 3 - Sistema horizontal



Fonte: ROHDE & SCHWARZ (2024).

Por fim, o sistema de *trigger* estabiliza o gráfico na tela do osciloscópio. O modo de *trigger* mais comum é o disparo pela borda, que ocorre quando a tensão ultrapassa um limite pré-determinado, seja na borda ascendente ou descendente de uma forma de onda (Figura 4). Alguns osciloscópios oferecem diferentes modos de trigger, como o modo automático, que auxilia na visualização da forma de onda antes da configuração do *trigger*, e o modo normal, no qual o sistema só adquire a forma de onda quando as condições de *trigger* são atendidas (ROHDE & SCHWARZ, 2024).

Figura 4 - Sistema de trigger



Fonte: ROHDE & SCHWARZ (2024).

2.2. Amostragem e *Aliasing*

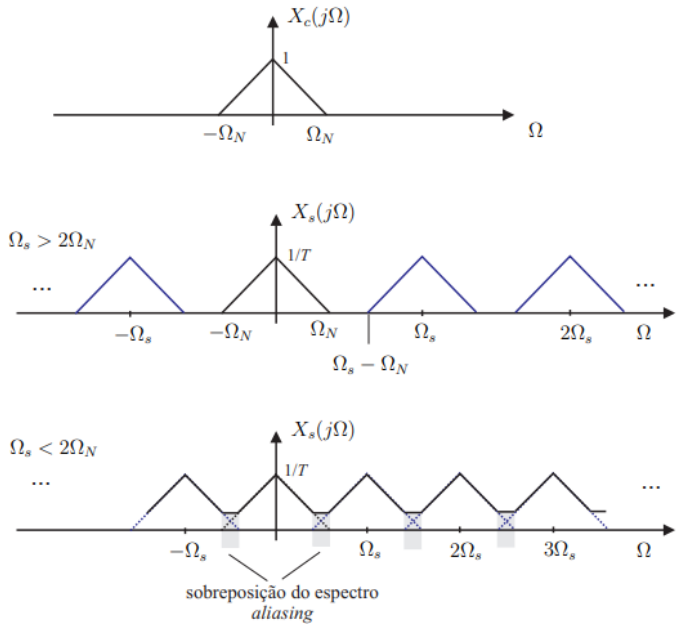
Amostragem é o processo de converter um sinal contínuo em um sinal discreto ao medir o valor do sinal contínuo em intervalos de tempo regulares. O conceito de amostragem é crucial na análise e no processamento de sinais, principalmente na conversão sinais analógicos para digitais. A amostragem é descrita pelo Teorema da Amostragem de Nyquist, em que afirma que para um sinal

a ser adequadamente amostrado e reconstruído sem perda de informação, deve ser amostrado a uma taxa mínima (frequência de amostragem f_s) que seja o dobro da maior frequência f presente no sinal original (LATHI; DING, 2012).

$$f_s > 2f$$

Quando o sinal é amostrado a uma taxa inferior à frequência de Nyquist, ocorre o efeito de *aliasing*. O *aliasing* é o fenômeno em que há sobreposição dos espectros de frequência do sinal, o que significa que o sinal analógico foi amostrado a uma taxa insuficiente, ou seja, abaixo da frequência de Nyquist. Quando o *aliasing* ocorre, torna-se impossível recuperar o sinal original de forma precisa. Uma representação gráfica do efeito de *aliasing* pode ser observada na Figura 5, onde Ω_N representa a frequência máxima do sinal e Ω_S a frequência de amostragem. É perceptível que o não cumprimento do teorema de Nyquist leva à sobreposição do sinal (*aliasing*).

Figura 5 - Efeito Aliasing

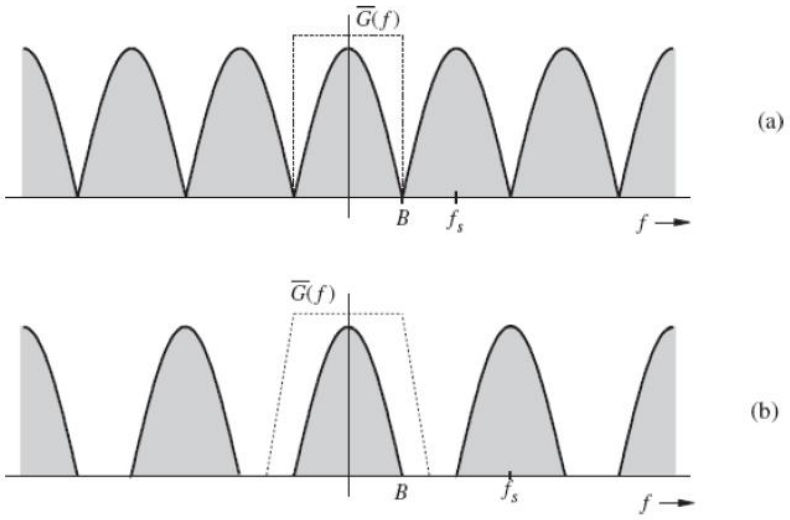


Fonte: HIGUTI.

Para evitar o fenômeno *aliasing*, normalmente é utilizado filtros passa-baixa (*anti-aliasing*) antes do processo de amostragem, removendo os componentes de frequência acima da metade da taxa de amostragem. Esse filtro garante que apenas as frequências dentro do limite de Nyquist sejam capturadas corretamente, preservando a integridade do sinal durante sua conversão analógico para digital

(LATHI; DING, 2012). A Figura 6 (a) representa a filtragem (linha pontilhada) à taxa de Nyquist e Figura 6 (b) à uma taxa superior a de Nyquist.

Figura 6 - Filtragem das frequências acima da frequência de amostragem

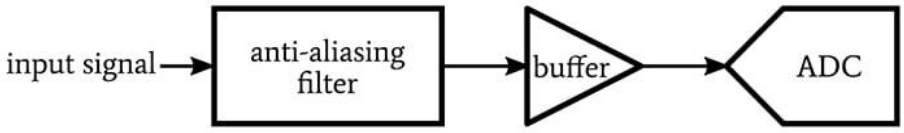


Fonte: Lathi; Ding (2012).

2.3. Filtro ativo passa-baixa (*anti-aliasing*)

Na prática, para evitar o efeito *aliasing* nos circuitos eletrônicos é preciso implementar um filtro passa-baixa antes da amostragem, isso é, antes do processo de conversão analógica-digital, com o propósito de limitar o sinal por banda (Figura 7). Ou seja, ao passar por um filtro passa-baixa, atenua-se o conteúdo espectral acima de uma frequência específica (frequência de corte) e, dessa forma, cria-se um limite de frequência superior (KEIM, 2020).

Figura 7 - Diagrama de blocos do filtro anti-aliasing

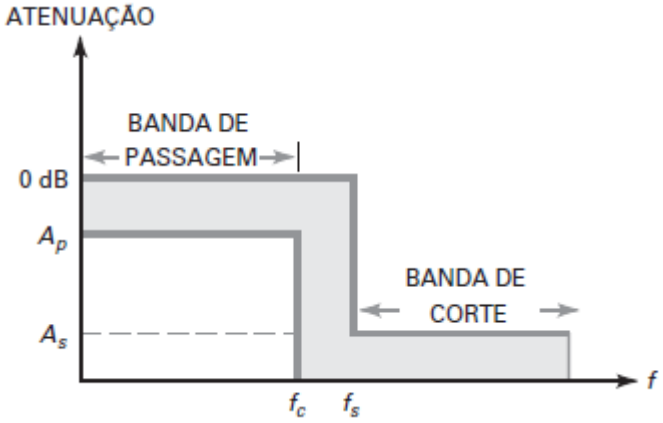


Fonte: Keim (2020).

Existe a configuração de filtros passivos e ativos, porém o foco deste estudo é utilizar o filtro ativo, por causa do filtro passivo possuir algumas deficiências, como a limitação da função de transferência. A maioria dos filtros ativos empregam amplificadores operacionais que permitem a simplificação dos modelos dos

fundamentos eletrônicos (RAZAVI, 2017). É possível analisar a resposta dos filtros ativos seguindo como base a Figura 8 que classifica as regiões do gráfico por banda.

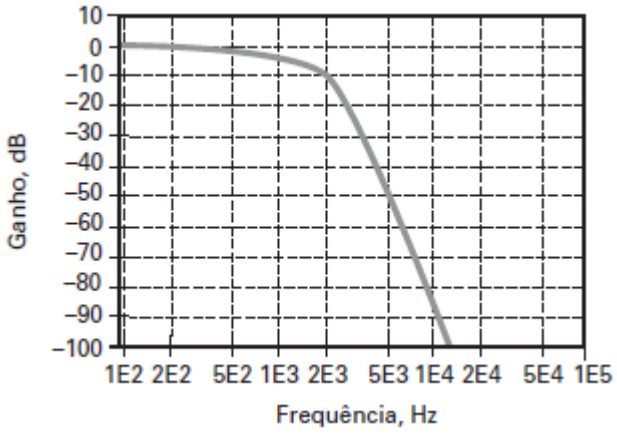
Figura 8 – Análise da resposta gráfica de um filtro ativo passa-baixa



Fonte: Malvino (2016).

Existem tipos diferentes de respostas de filtros ativos passa-baixa, como o *Butterworth* e *Chebyshev*. A resposta *Butterworth* do filtro passa-baixa pode ser representado pela Figura 9, em que é caracterizada pela ausência de ondulações nas bandas passante e rejeição. Quanto maior a ordem do filtro, mais a resposta se torna abrupta na banda de transição e maior a planura na banda passante.

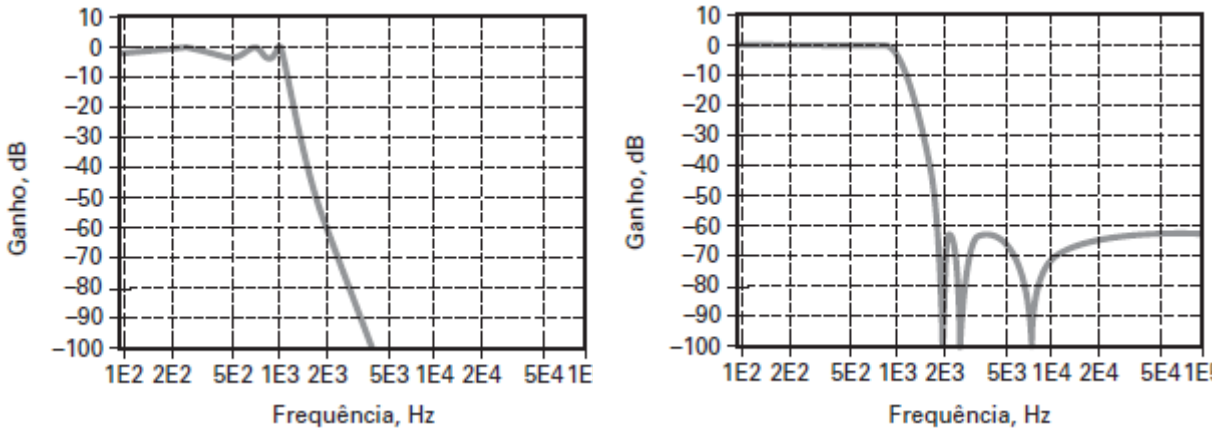
Figura 9 - Resposta de um filtro passa-baixa Butterworth



Fonte: Malvino (2016).

Já a resposta *Chebyschev* possui como característica o comportamento de “ondulação constante” (*equiripple*) na banda passante, conforme ilustra a Figura 10.

Figura 10 - Resposta do filtro passa-baixa da aproximação Chebyschev

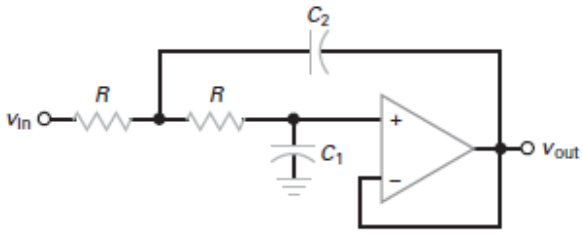


(a) Aproximação *Chebyschev* (b) Aproximação *Chebyschev* inversa

Fonte: Malvino (2016).

Como o objetivo deste estudo é apenas filtrar as frequências acima da metade da frequência de amostragem, isso é, atenuar a magnitude das frequências superiores a frequência de corte, a resposta *Butterworth* atende esse requisito e será estudado neste presente trabalho. Eletronicamente, a Figura 11 representa um modelo de um filtro passa-baixas de segunda ordem *Sallen-Key* para a aproximação *Butterworth* (MALVINO, 2016).

Figura 11 - Configuração eletrônica do filtro passa-baixa Butterworth de segunda ordem Sallen-Key



Fonte: Malvino (2016).

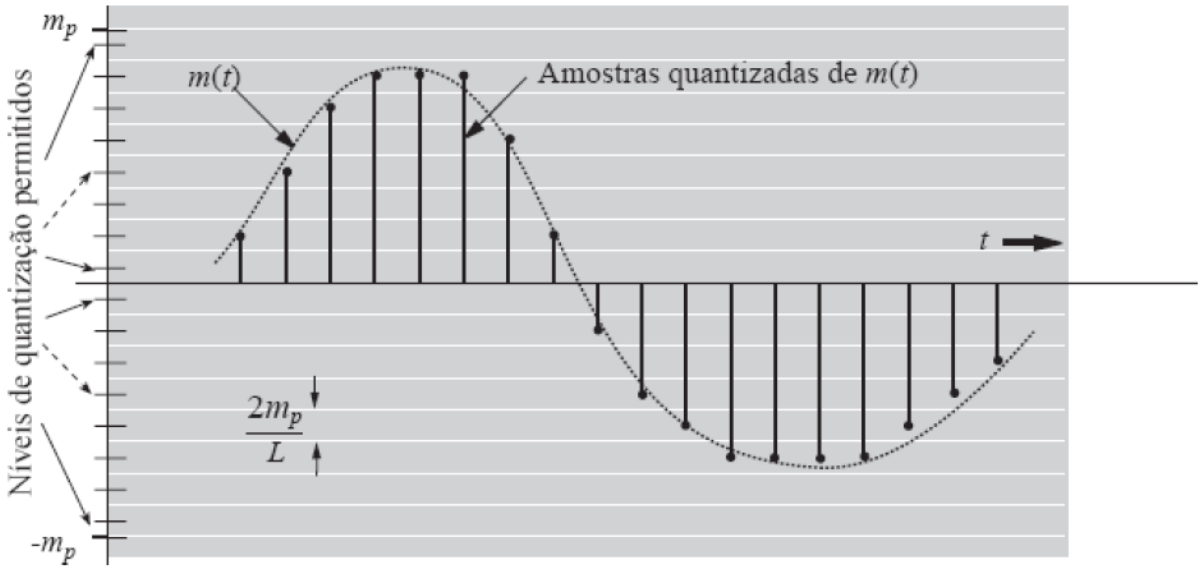
2.4. ADC (*Analog-to-Digital Converter*)

Compreendida a filtragem do sinal, pode ser estudado o processo de conversão analógico-digital. O conversor analógico digital ADC é responsável pela conversão de dados analógicos condicionados em um fluxo de dados digitais. Os sinais analógicos são contínuos no tempo e podem assumir qualquer amplitude em um intervalo de valores. Os dados digitais, por sua vez, existem em momentos

discretos no tempo e assumem somente um número finito de valores. (LATHI; DING, 2012)

A conversão A/D é construída em duas etapas: primeiro, um sinal em tempo contínuo é amostrado para produzir um sinal em tempo discreto, cujas amplitudes são contínuas; segundo, esse sinal é quantizado em níveis discretos de sinal. Ou seja, o processo de amostragem é realizado, em conformidade com o teorema de Nyquist e, por fim, o sinal passa por um processo denominado de quantização, em que cada amostra é aproximada ao nível de quantização mais próximo, conforme ilustra a Figura 12 (LATHI; DING, 2012).

Figura 12 - Ilustração do processo de amostragem



Fonte: Lathi; Ding (2012).

Vale ressaltar que como o sinal quantizado é uma aproximação do sinal original e que a percepção humana não exige uma precisão infinita, a conversão A/D pode, efetivamente, obter a informação necessária de uma fonte analógica para a transmissão do sinal digital (LATHI; DING, 2012).

Existem tipos de conversores digitais, como o Aproximação Sucessiva (SAR), e *Delta-Sigma* os ADCs *flash*. Resumidamente, os do tipo aproximação sucessiva são conversores que possuem um equilíbrio de velocidade e resolução, lidam com uma gama de variedade de sinais, são de baixo custo, não possuem nenhum tipo de filtro *anti-aliasing* e sua resolução é limitada do eixo de amplitude. Os do tipo ADC *Delta-Sigma* são utilizados em aplicações de alta dinâmica, como ruído, áudio,

choque e vibração, balanceamento e processamento senoidal. Esses possuem um filtro passa-baixa implementado que elimina o ruído de quantização. Já os do tipo ADC *flash* são conhecidos pelas suas taxas de amostragem extremamente altas, mas com resolução do eixo de amplitude baixa (DEWESOFT, 2024).

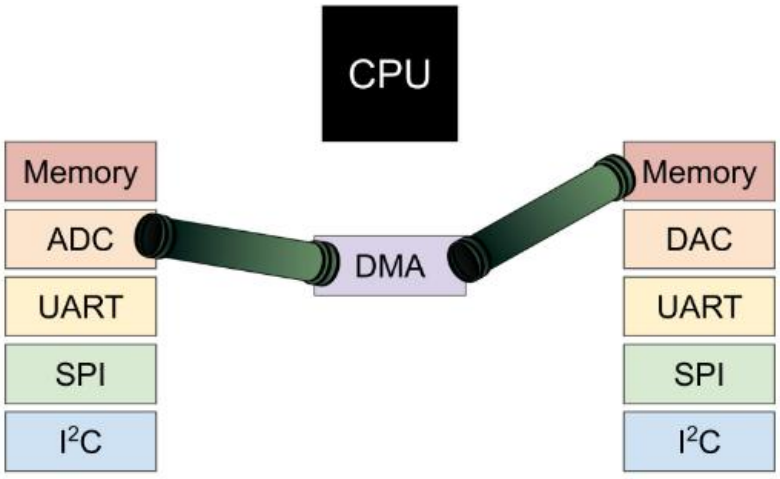
2.5. DMA (*Direct Memory Access*)

O DMA (sigla do inglês *Direct Memory Access*, que significa Acesso Direto a Memória) é um método que possibilita os dispositivos de entrada e saída (periféricos) a terem acesso direto à memória RAM (*Random Access Memory*), sem sobrecarregar o processador ou unidade central de processamento (CPU) e acelera as operações que envolvem a memória (ADRENALINE, 2024).

Normalmente, temos vários periféricos que são controlados pela CPU e quando deseja enviar dados de um periférico para outro, a CPU deve ler os dados da memória e enviá-los para outro periférico (ou memória). Quando se trata de uma grande quantidade de dados, pode sobrecarregar a CPU e prejudicar o seu desempenho (DIGIKEY, 2019).

Muitos microcontroladores modernos, inclusive os da plataforma STM32, possuem um controlador DMA, em que envia dados automaticamente de um periférico para outro sem o uso da CPU. Isso torna a CPU disponível para outras atividades, isso é, aumenta o desempenho do projeto. Na Figura 13, é possível visualizar esse efeito (DIGIKEY, 2019).

Figura 13 - Ilustração do funcionamento do DMA



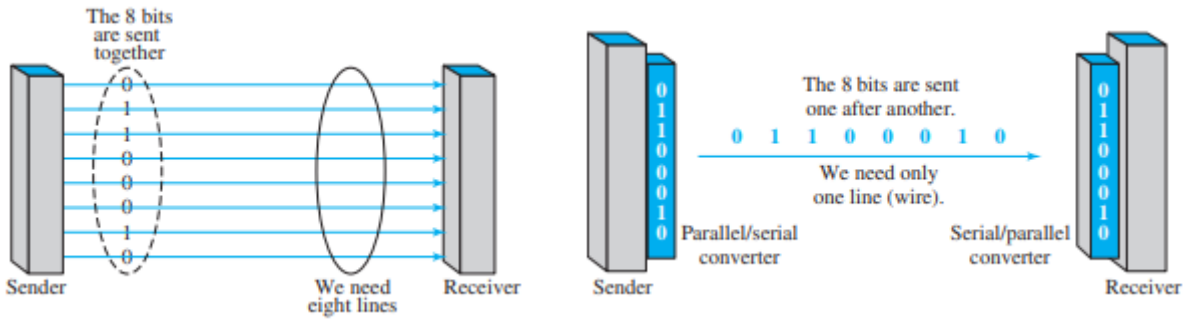
Fonte: DIGIKEY (2024).

Portanto, para casos em que é preciso mover fluxos longos e contínuos de dados, como coletar dados do ADC e armazená-los em um *buffer*, é recomendado o uso de DMA.

2.6. Protocolos de comunicação

Para a transmissão de dados digitais, existem dois tipos de comunicação: paralela e serial. A comunicação paralela é definida quando se envia vários bits a cada *tick* de *clock*. No modo serial, envia-se uma sequência de bit, ou seja, 1 bit é enviado a cada *tick* de *clock*. (FOROUZAN, 2012) A Figura 14 ilustra a definição de cada modo de comunicação.

Figura 14 - Funcionamento da comunicação paralela e serial



(a) Comunicação paralela (b) Comunicação serial

Fonte: Forouzan (2012).

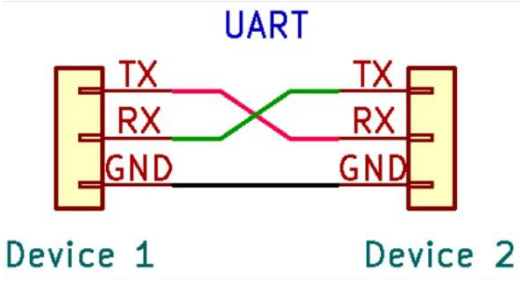
Em uma comunicação em que se deseja enviar uma informação de 8 bits, a transmissão paralela apresenta como vantagem sua alta velocidade, pois, a cada *tick* de *clock*, é possível enviar 8 bits simultaneamente, ou seja, o dado completo. Já na transmissão serial, seriam necessários 8 *ticks* de *clock* para enviar a mesma informação, o que demandaria mais tempo de transmissão. No entanto, a comunicação paralela exige um fio ou pino (no contexto de microcontroladores) para a transmissão de cada bit, o que, nesse exemplo, utilizaria 8 pinos/fios para o envio do dado. Em contraste, a comunicação serial utiliza apenas um único fio ou pino para transmitir todos os bits da informação.

Neste trabalho, será utilizado apenas o modo de transmissão serial, o qual será estudado com mais detalhes. A transmissão serial pode ser dividida em três tipos: assíncrona, síncrona e isócrona. A transmissão assíncrona é caracterizada pela ausência de sincronização por *clock*, ou seja, os dados são transmitidos sem

um sinal de *clock* comum entre o transmissor e o receptor. Por outro lado, a transmissão síncrona utiliza um *clock* para sincronizar a transmissão de dados entre os dispositivos. Já a transmissão isócrona é definida pela garantia de uma taxa de entrega fixa de dados, sendo essencial em aplicações que demandam tempo real. Cada um desses modos de comunicação serial possui protocolos específicos, como, por exemplo, UART (assíncrona), SPI (síncrona) e USB (isócrona).

O UART (*Universal Asynchronous Receiver Transmitter*) é um protocolo de comunicação assíncrona de fácil implementação e amplamente utilizado na comunicação entre microcontroladores e periféricos. Trata-se de um protocolo full-duplex, ou seja, é capaz de transmitir e receber dados simultaneamente. Ele utiliza duas conexões principais: RX (recepção) e TX (transmissão), onde o RX é responsável por receber os dados e o TX por transmiti-los. A configuração da conexão entre os dispositivos pode ser visualizada na Figura 15.

Figura 15 - Conexão UART entre os dispositivos



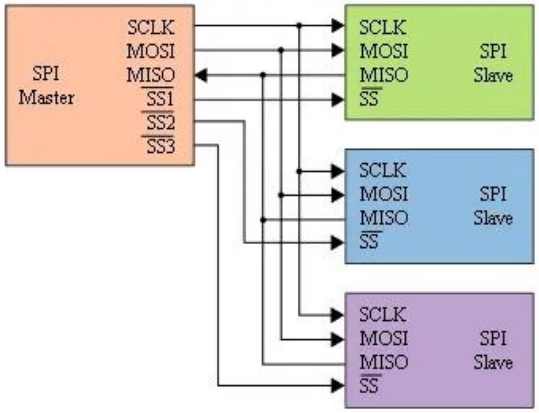
Fonte: Phillips (2023).

O SPI (*Serial Peripheral Interface*) é um protocolo de comunicação síncrona full-duplex que utiliza uma conexão de *clock* para sincronizar a transmissão de dados. Ele é especialmente indicado para situações em que há múltiplos periféricos conectados a um único dispositivo, sendo organizado em uma arquitetura mestre-escravo, na qual há um dispositivo mestre e um ou mais escravos. Cada escravo possui uma linha de seleção (SS ou CS - *chip select*) dedicada, que permite ao mestre ativá-lo para enviar ou receber informações. O mestre, por sua vez, controla todas as linhas de seleção dos escravos (SACCO, 2014).

Tanto o mestre quanto os escravos possuem conexões para o *clock* (SCLK), além dos pinos MISO (*Master Input Slave Output*) e MOSI (*Master Output Slave*

Input). O pino MISO permite que o mestre receba dados enviados pelo escravo, enquanto o pino MOSI é utilizado para o mestre enviar informações ao escravo. Em resumo, o funcionamento do protocolo SPI é ilustrado na Figura 16.

Figura 16 - Conexão SPI entre os dispositivos



Fonte: Sacco (2014).

O protocolo USB (*Universal Serial Bus*) é um tipo de comunicação serial isócrona que garante a transmissão de dados a uma taxa fixa. Esse protocolo assegura a transferência de novos dados em intervalos de tempo regulares e é amplamente utilizado em streaming de câmera e microfone. (Learning about Eletronics, 2024)

2.7. Microcontroladores

2.7.1. ESP32

Para acoplar a função de osciloscópio ao kit didático elaborado por Soares (2023), é necessário utilizar, como microcontrolador principal, o mesmo adotado no projeto: o ESP32-WROOM-32, mais conhecido como ESP32. Este microcontrolador, desenvolvido pela Espressif Systems ® e lançado em 2016, é amplamente empregado em projetos de Internet das Coisas (IoT), devido à sua conectividade Wi-Fi e Bluetooth. Além dessas características, o ESP32 destaca-se por oferecer um bom desempenho de processamento e ser uma solução de baixo custo (PEREIRA; CHAARI; DAROGE, 2023).

Existem diferentes módulos desse microcontrolador disponíveis no mercado para facilitar as conexões com periféricos e sua programação. O módulo utilizado

neste trabalho é o ESP32 DEVKIT V1 com 30 pinos, conforme ilustrado na Figura 17. Apesar de existirem outras versões, como a de 38 pinos, a maioria desses módulos compartilha a mesma configuração e funcionalidade de pinagem (Last Minute Engineers, 2024).

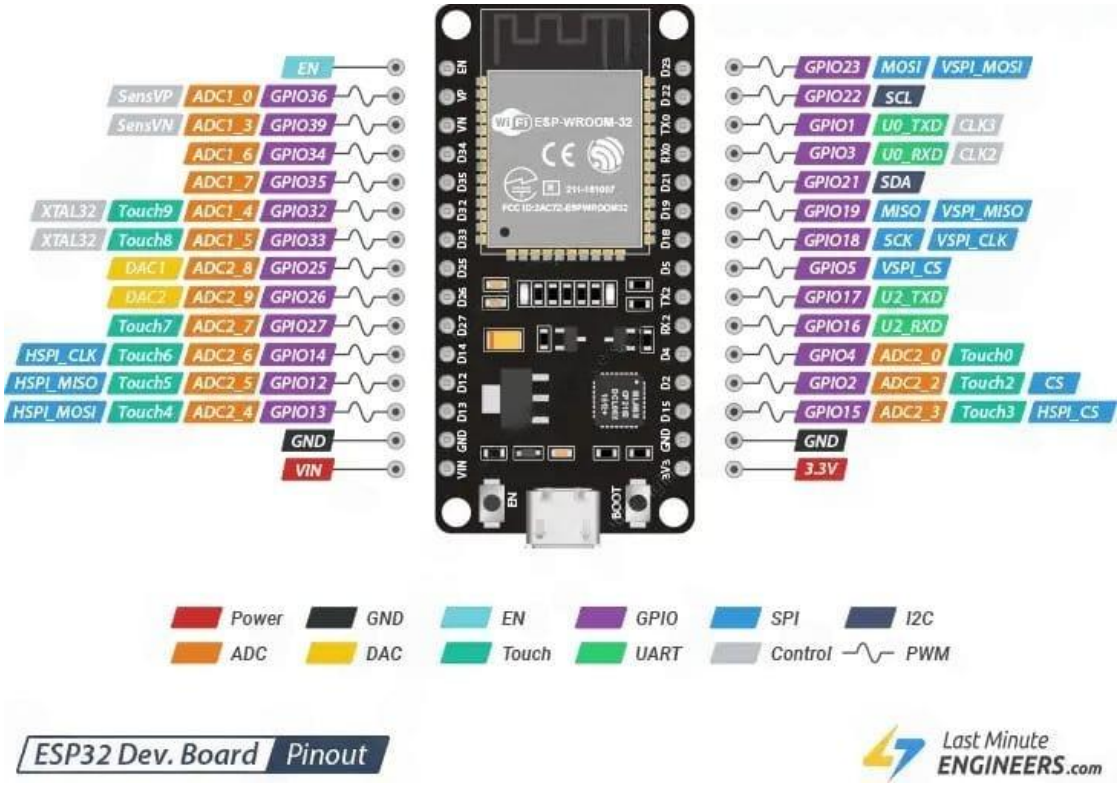
Figura 17 - Microcontrolador ESP32 DEVKIT V1

Fonte: Last Minute Engineers (2024).

A pinagem do ESP32 DEVKIT V1 é detalhada na Figura 18, onde pode-se destacar algumas características, como:

- 25 pinos GPIO (*General Purpose Input/Output*) programáveis, que podem ser configurados para interagir com periféricos, operando como entradas ou saídas e habilitados como portas digitais ou analógicas;
- 25 saídas PWM (*Pulse Widht Modulation*);
- 15 canais ADC;
- 2 interfaces para comunicação UART, Interface 0 (RX0 - GPIO3 e TX0 - GPIO1) e interface 2 (RX0 – GPIO16 e TX0 - GPIO17)

Figura 18 - Pinagem do ESP32 DEVKIT V1



Fonte: Last Minute Engineers (2024).

2.7.2. Ambiente de Desenvolvimento Integrado da ESP32

Para a programação do microcontrolador ESP32 podem ser utilizadas dois ambientes de desenvolvimento integrado (IDE – *Integrated Development Environment*), o ESP-IDF® ou o Arduino IDE ®.

O ESP-IDF® foi desenvolvido pela Espressif ® para programar os hardwares fornecidos por ela. Ele oferece um SDK (*Software Development Kit* – Kit de Desenvolvimento de Software) autossuficiente para o desenvolvimento de aplicativos genéricos das séries ESP32, ESP32-S, ESP32-C e ESP32-H de SoCs, utilizando as linguagens C ou C++. Trata-se de uma plataforma de código aberto, disponibilizada gratuitamente para os usuários que desejam utilizá-la (ESPRESSIF, 2024).

Por outro lado, o Arduino IDE ® foi desenvolvido pela empresa Arduino ® e também permite a programação dos microcontroladores ESP32, desde que instalado e configurado adequadamente. Assim como o ESP-IDF, é uma plataforma de código aberto que utiliza as linguagens C ou C++.

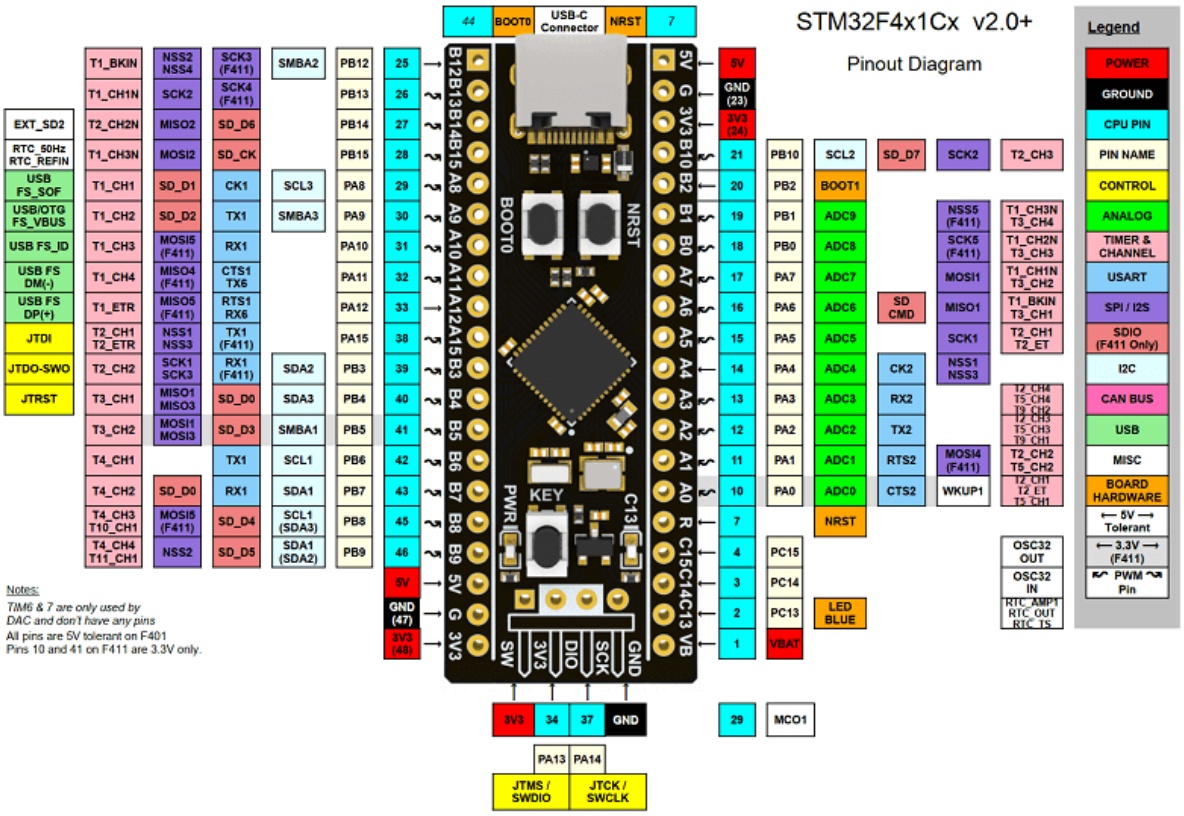
2.7.3. STM32F411

Outro microcontrolador utilizado neste estudo é o STM32F411, fabricado pela STMicroelectronics ®. Esse microcontrolador é equipado com um núcleo Cortex-M4 com suporte a ponto flutuante, operando a 100 MHz, o que garante um desempenho eficiente. Além disso, destaca-se pela eficiência energética, com consumo de energia significativamente baixo em comparação a outros modelos (STMicroelectronis, 2024).

A pinagem do STM32F411 é ilustrada na Figura 19, destacando as seguintes características principais:

- Interfaces de comunicação:
  - 3 I2Cs até 1Mbps;
  - 5 SPIs até 50Mbits/s
  - 3 USARTs rodando a até 12,5 Mbit/s (USART1, USART2 e USART6), é importante ressaltar que esses podem ser configurados como UART;
- ADC de 12 bits atingindo 2,4 MSPS;
- 11 temporizadores, de 16 e 32 bits, operando a até 100 MHz

Figura 19 - Pinagem do STM32F4x1Cx



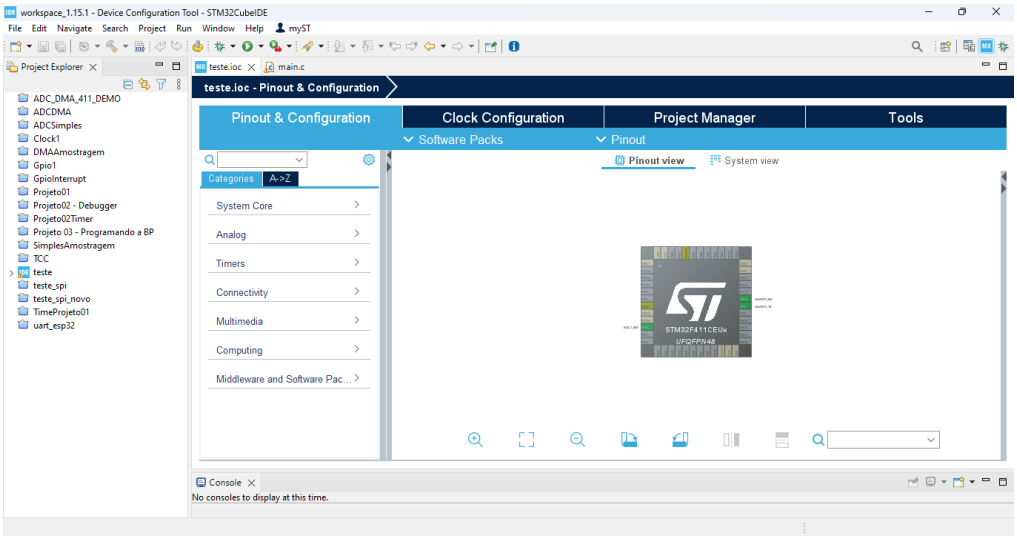
Fonte: Mouser Electronics (2021).

2.7.4. Ambiente de desenvolvimento integrado da STM32

Para a programação dos hardwares da plataforma STM32, utiliza-se o STM32CubeIDE, uma IDE baseada no framework Eclipse ®, que oferece os recursos necessários para o desenvolvimento e a depuração do código-fonte em C/C++ (STMicroelectronis, 2024).

Ao iniciar o ambiente de desenvolvimento, o usuário pode selecionar o MCU desejado para configurar e programar adequadamente. Após essa etapa, a interface gráfica do STM32CubeIDE, ilustrada na Figura 20, permite configurar as funções e os modos de operação dos pinos do microcontrolador selecionado. As abas mais utilizadas nesse processo são: *Pinout & Configuration* e *Clock Configuration*.

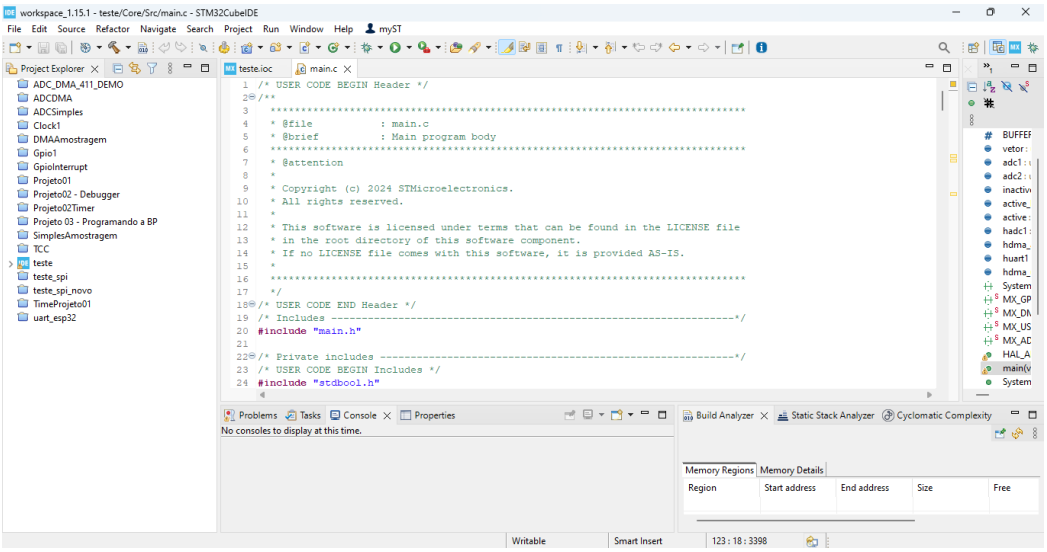
Figura 20 – Interface gráfica do STM32CubeIDE



Fonte: O autor (2024).

Após salvar as configurações, a plataforma gera o arquivo main.c, onde será desenvolvido o código principal do microcontrolador. A perspectiva da interface é automaticamente ajustada para tornar as ferramentas de compilação e depuração mais acessíveis, facilitando o desenvolvimento do projeto. A interface dessa perspectiva é apresentada na Figura 21.

Figura 21 - Interface STM32CubeIDE para o desenvolvimento do código

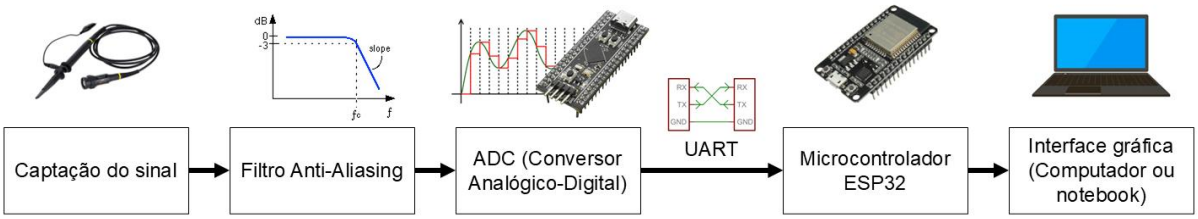


Fonte: O autor (2024).

3. Materiais e métodos

O esquema proposto para a implementação da pesquisa está representado na Figura 22. O sinal de entrada passou por um filtro passa-baixa (*anti-aliasing*) para eliminar as frequências acima da metade da frequência de amostragem, conforme o teorema de Nyquist. Após a filtragem, o sinal foi convertido de analógico para digital pelo microcontrolador STM32, possibilitando o processamento dos dados em formato digital. Esses dados foram transmitidos via protocolo de comunicação UART para o microcontrolador ESP32, onde foram processados para a geração e plotagem dos resultados e gráficos correspondentes ao sinal capturado.

Figura 22 – Esquema do funcionamento do datalogger



Fonte: O autor (2024).

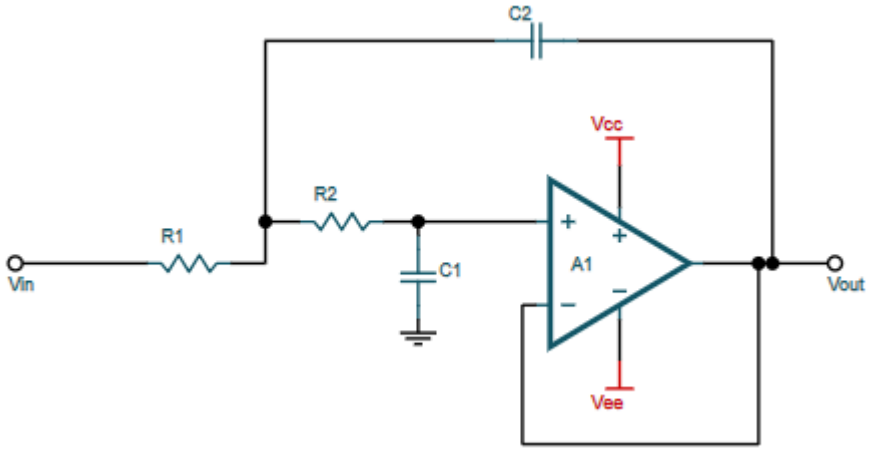
3.1. Filtro *Anti-Aliasing*

O filtro passa-baixa (*anti-aliasing*) *Butterworth* foi escolhido neste trabalho para realizar a filtragem das frequências acima da metade da frequência de amostragem. A escolha pelo filtro passa-baixa *Butterworth* se deu pela resposta de amplitude suave, transição suave e fácil implementação. A resposta de amplitude na faixa passante desse filtro é plana, diferente de outros filtros, como o *Chebyshev*, que possui ondulações (*ripples*) no sinal. A transição entre a banda passante e a banda de rejeição do filtro *Butterworth* é gradual, o que minimiza a distorção do sinal.

A configuração de um filtro passa-baixa *Butterworth* utilizada neste estudo pode ser visualizada na Figura 23. A criação do filtro foi realizada utilizando o site da Texas Instruments ®. Para isso, foi necessário informar a frequência de corte desejada, que foi definida em 350 kHz. Esse valor foi escolhido devido à frequência de amostragem ser de 694 kHz, um aspecto que será detalhado na Seção 3.2. ADC

(STM32F411), onde são abordados os conceitos de amostragem e suas características.

Figura 23 - Configuração do filtro passa-baixa Butterworth



Fonte: Texas Instruments (2024).

Os valores obtidos pela plataforma foram

$$R1 = 2{,}74k\Omega$$

$$R2 = 3{,}65k\Omega$$

$$C1 = 100pF$$

$$C2 = 205pF$$

Para obter os valores práticos foi realizada uma aproximação dos valores teóricos. Os valores comerciais foram

$$R1 = 2{,}7k\Omega$$

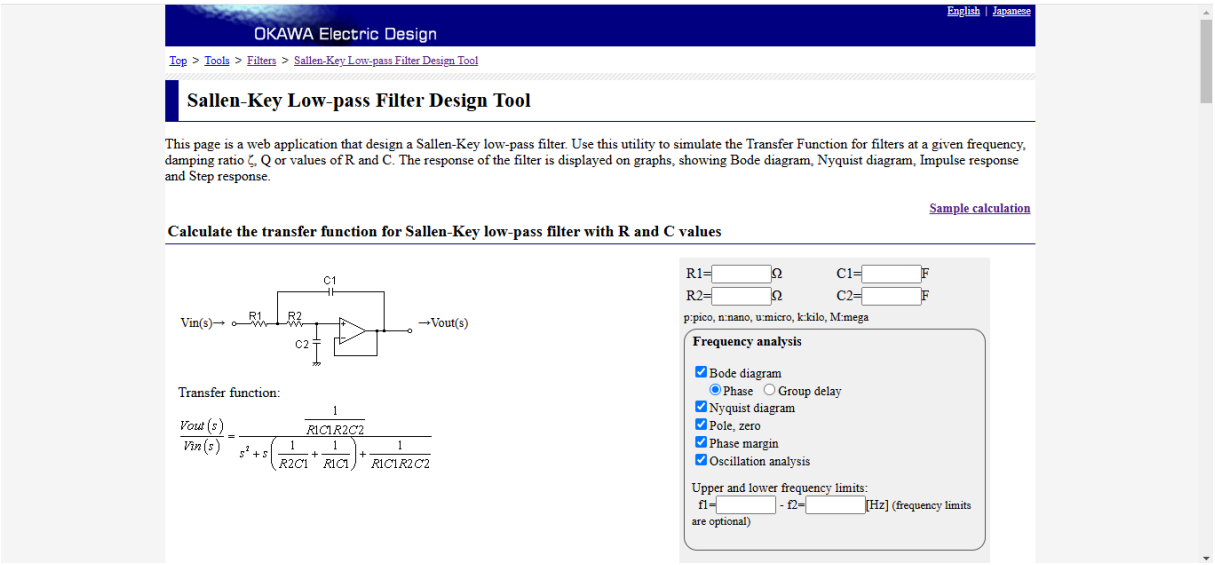
$$R2 = 3{,}6k\Omega$$

$$C1 = 100pF$$

$$C2 = 220pF$$

Com os valores comerciais foi preciso verificar o valor da frequência de corte desse filtro. Para isso, foi utilizado um site de análises de filtros conhecido como Okawa (OKAWA Electric Design, 2024) em que é possível escolher o filtro de interesse, fornecer os valores dos componentes passivos e o site informa a frequência de corte, conforme ilustra a Figura 24.

Figura 24 - Interface da plataforma Okawa



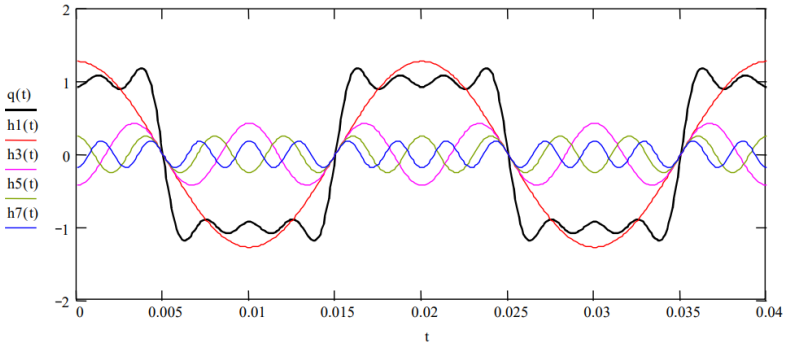
Fonte: OKAWA Electric Design (2024).

Após de informar os valores do filtro e clicar no botão "*Calculate*", o site informou uma frequência de corte $f_c = 356541.48471829(Hz)$.

Houve uma discrepância do valor teórico do valor prático da frequência de corte, isso devido a variação dos valores dos componentes passivos que foram aproximados para os valores disponíveis no mercado.

É importante destacar que, para a análise dos resultados, utilizou-se uma onda quadrada como sinal de entrada de referência. A onda quadrada é composta por harmônicas ímpares (n = 1, 3, 5, 7, 9 ...), e quanto mais harmônicas são somadas, mais evidente se torna sua forma quadrada, conforme ilustrado na Figura 25. Nessa figura, $q(t)$ representa o somatório das harmônicas $h1(t)$, $h3(t)$, $h5(t)$ e $h7(t)$.

Figura 25 - Composição da onda quadrada a partir de suas harmônicas



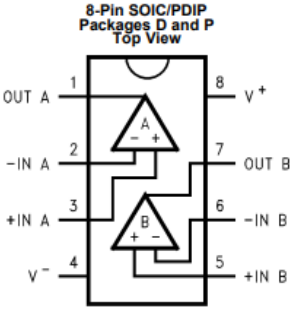
Fonte: Pomilio (2013).

Para garantir uma boa resposta do filtro e preservar a característica de onda quadrada, a frequência do sinal de entrada deve ser cerca de 10 vezes menor que a frequência de corte, ou aproximadamente 35 kHz. Isso assegura que, pelo menos, as 11 primeiras harmônicas componham a onda quadrada, mantendo sua forma característica. Para realizar testes e analisar o comportamento do filtro, foram observadas suas respostas nas frequências de 10 kHz, 20 kHz e 40 kHz.

Como a proposta é desenvolver um *datalogger* compatível com o kit didático de Soares (2023), cuja a fonte de alimentação disponível é de 0 a 24V, optou-se por um amplificador operacional (AO) que pudesse operar nessa faixa de tensão de forma assimétrica. Para atender essa necessidade, foi escolhido o LM6132BIM, um amplificador capaz de operar com $V^+ = +24V$ e $V^- = 0V$. Esse AO possui uma largura de banda de ganho unitário de 10MHz, o que permite sua operação estável até essa frequência. A Figura 26 (a) representa o AO e a Figura 26 (b) as suas respectivas pinagens, obtidas no datasheet do componente.

Figura 26 - Amplificador Operacional LM6132BIM

(a) Visão geral do AO

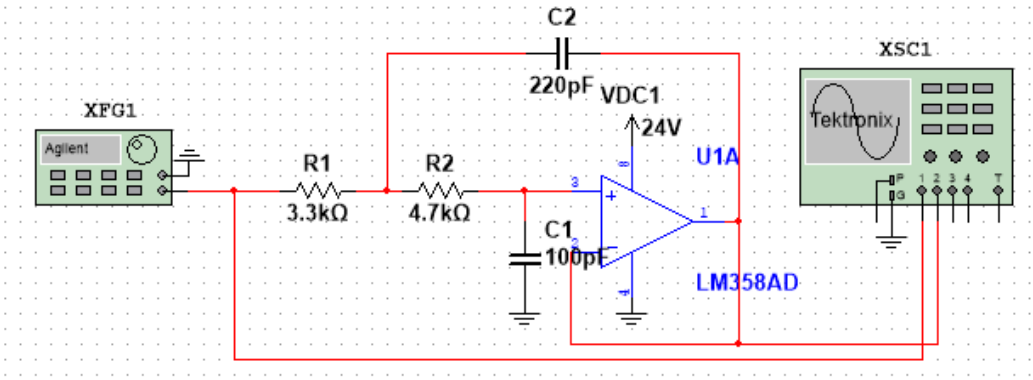


(b) Pinagens do AO

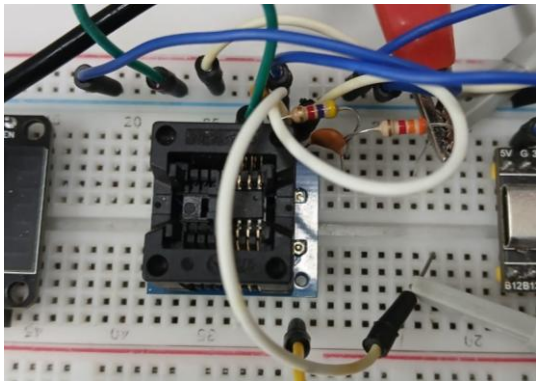
Fonte: Adaptado do Datasheet LM6132BIM.

Dessa forma, foi possível a montagem da simulação no Multisim ® do filtro representada pela Figura 27 (a) e a montagem na *protoboard* representada pela Figura 27 (b).

Figura 27 - Montagem do filtro anti-aliasing



(a) Simulação



(b) *Protoboard*

Fonte: O autor (2024).

Realizada a montagem do circuito do filtro no simulador Multisim ® e na *protoboard*, foi possível obter os resultados simulados e práticos, esses foram analisados na Seção 4.

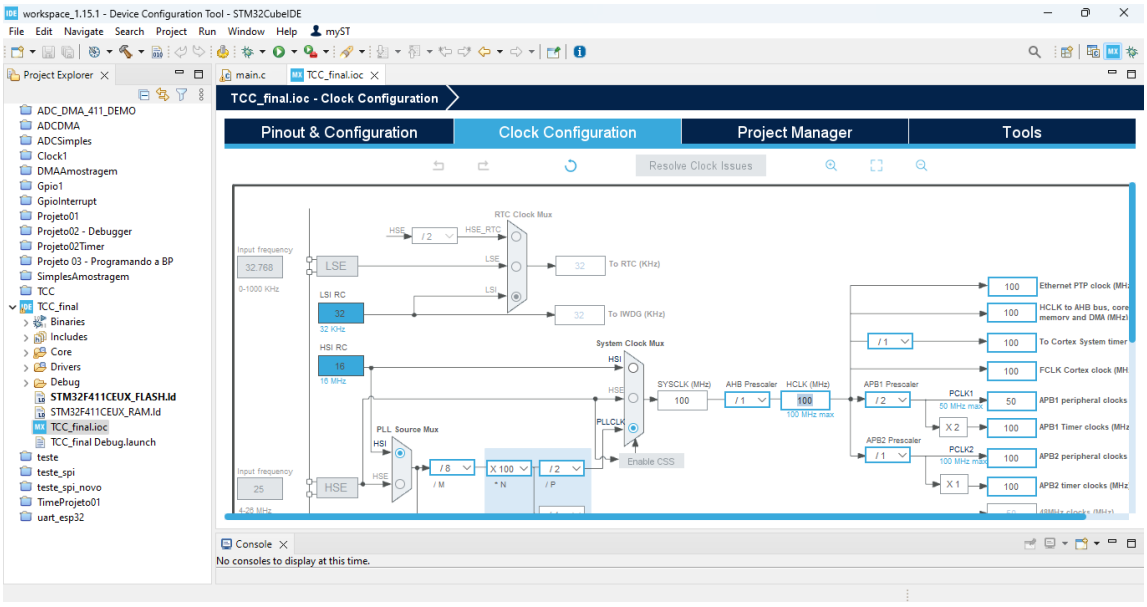
3.2. ADC e DMA (STM32F411)

O datasheet do microcontrolador ESP32 informa que seu ADC possui 6% de diferença de medição, inclusive a própria fabricante Espressif ® fornece métodos de calibração para a medição do ADC. (Datasheet ESP32, 2024) Visto isso, concluiu-se que o ADC da ESP32 não é o mais recomendado para uma boa medição de valores. Já os microcontroladores da STMicroelectronics Company ® são bons

equipamentos utilizados para diversas aplicações, especialmente quando é preciso converter um sinal analógico para um sinal digital. (STMicroelectronics, 2024)

Nesta perspectiva, utilizou-se o microcontrolador STM32F411 da ST como conversor analógico-digital do projeto. Inicialmente, é preciso configurar o ADC da ST pela plataforma de desenvolvimento STM32CubeIDE. Para isso, criou-se um projeto com o microcontrolador em questão. Como o objetivo é obter a máxima frequência de amostragem do ADC, foi preciso configurar a frequência do *clock* como máxima. Para isso, na aba *Clock Configuration* foi ajustado o termo HCLK na máxima frequência permitida que, nesse microcontrolador, é 100MHz, conforme ilustra a Figura 28.

Figura 28 - Configuração Clock Configuration



Fonte: O autor (2024).

Pelo datasheet, o ADC está ligado ao barramento APB2 que, nesta configuração, o *APB2 peripheral clock* está a 100MHz, conhecida como a frequência do *clock* do ADC f_{clock}. Portanto, a frequência de operação do microcontrolador foi de $f_{clock} = 100\,MHz$.

Na aba *Pinout & Configuration*, na configuração *Analog* é possível selecionar o ADC1 e selecionar o modo (*Mode*) IN0, por exemplo. Assim, o ADC1 é habilitado e é possível ajustar os parâmetros do ADC na configuração *Parameter Settings* para obter a sua frequência de amostragem. No parâmetro *Clock Prescaler* selecionou a

menor divisão disponível que é *PLCK2 divided by 4*, ou seja, a frequência do *clock* será dividida por 4, essa é a frequência f_{ADC} . Portanto, a frequência do ADC $f_{ADC} = \frac{100MHz}{4} = 25MHz$.

Com a frequência f_{ADC} definida, foi necessário escolher a resolução do ADC (*Resolution*). Para simplificar e facilitar o entendimento do código, optou-se por configurar a resolução do ADC em 8 bits, pois a UART transmite dados em blocos desse tamanho. Essa configuração permite transmitir os dados lidos diretamente e plotá-los em um gráfico de forma mais eficiente, sem necessidade de manipular bits mais ou menos significativos. Além disso, a escolha de 8 bits se mostrou ideal, pois, se a resolução fosse configurada em 12 bits, o ADC reservaria 16 bits para cada amostra, exigindo dois blocos de 8 bits para transmitir cada valor. Isso aumentaria o tempo de transmissão e demandaria mais espaço de memória nos microcontroladores.

Foi preciso habilitar o parâmetro *Continuos Conversion Mode* para que o ADC opere de modo contínuo de conversão. Para finalizar, foi necessário determinar quantos ciclos que o ADC irá executar para amostrar o valor convertido. Para isso, foi preciso expandir o parâmetro Rank que está dentro de ADC_Regular_ConversionMode e selecionar a quantidade de 28 ciclos. Experimentalmente, 3 e 15 ciclos não obteve resultados devido a rápida conversão do ADC, logo os valores convertidos ficaram desconfigurados.

Pelo datasheet da STM32F411, a fórmula do tempo de conversão de cada amostra é dada por

$$t_c = \frac{ciclos\ +\ n\,bits\,resolution}{f_{ADC}}$$

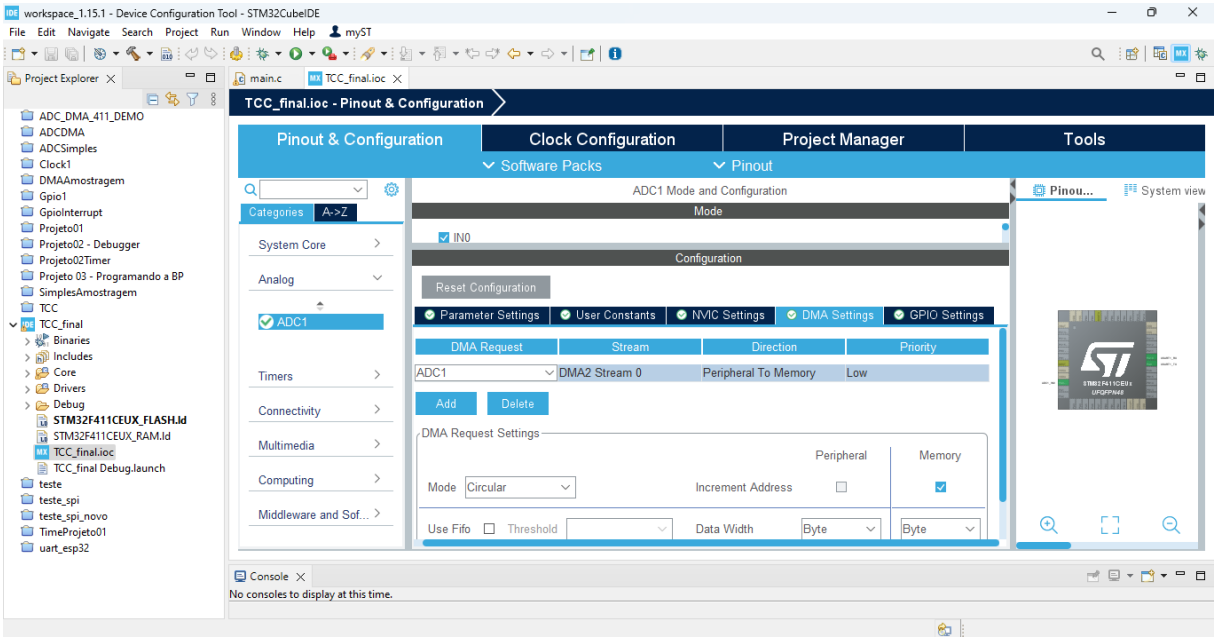
$$t_c = \frac{28 + 8}{25MHz} = 1{,}44\mu s$$

Logo o tempo de conversão de uma amostra para outra foi de $1{,}44\mu s$, isto é uma frequência de amostragem de $694444Hz$. De acordo com o teorema de Nyquist e o estudo do filtro passa-baixa, a frequência de corte do filtro deve ser a metade da frequência de amostragem, logo foi determinada que a frequência de corte do filtro seria de 350kHz, aproximadamente. Portanto, a frequência de amostragem do ADC é de 694kHz, aproximadamente.

Para um melhor desempenho, a conversão analógica-digital foi configurada utilizando o acesso direto à memória DMA para deixar a CPU livre dessa tarefa. Para

ativar o DMA do ADC foi preciso ir na configuração *DMA Settings* e adicionar o ADC1. Ao adicionar o ADC1, foi selecionado o modo circular e a opção *Byte* na lista de seleção *Data Width do Peripheral*. Automaticamente, a memória também foi ajustada para Byte. Essa configuração é necessária para poder converter os dados em bytes e tornar possível a comunicação UART. A configuração do DMA é ilustrada na Figura 29.

Figura 29 - Configuração do DMA no STM32CubeIDE

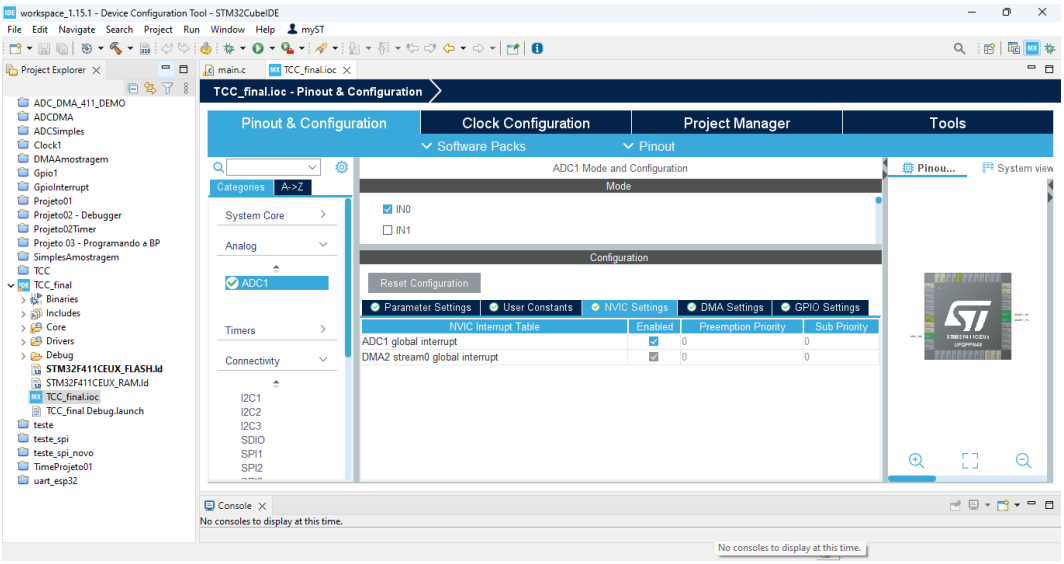


Fonte: O autor (2024).

Para habilitar o DMA do ADC, foi necessário acessar a configuração *Parameter Settings* e ativar o parâmetro *DMA Continuous Requests*. Com isso, o ADC foi configurado para operar a uma frequência de amostragem de 694 kHz, utilizando o DMA para melhorar a eficiência e o desempenho do microcontrolador.

Por fim, como o ADC irá operar em modo de interrupção foi necessário ativar a interrupção do ADC. Para isso, na configuração NVIC Settings habilitou-se a opção *ADC1 global interrupt*, conforme a ilustração da Figura 30.

Figura 30 - Configuração da interrupção do ADC no STM32CubeIDE

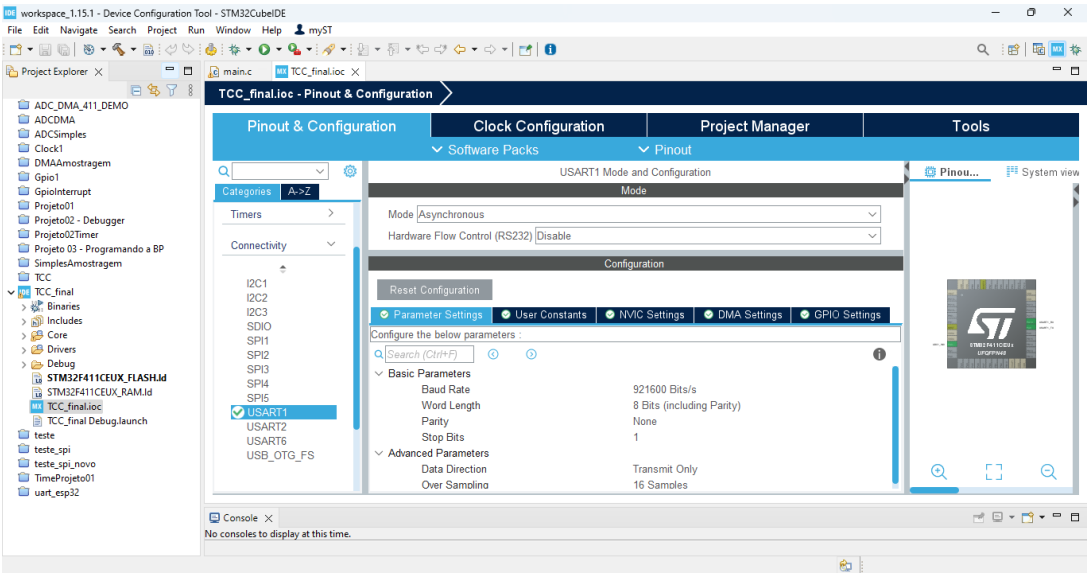


Fonte: O autor (2024).

3.3. UART (STM32F411 e ESP32)

Primeiramente, foi preciso configurar a UART da STM32. Para isso, na aba *Pinout & Configuration*, na categoria *Connectivity*, selecionou-se a opção USART1. Na configuração, foi selecionada a opção modo Assíncrono (*Asynchronous*). Na configuração *Parameter Settings*, foi definida como velocidade *Baud Rate* 921600 bits/s e *Word Length* como 8 bits. A configuração final da UART é representada pela Figura 31.

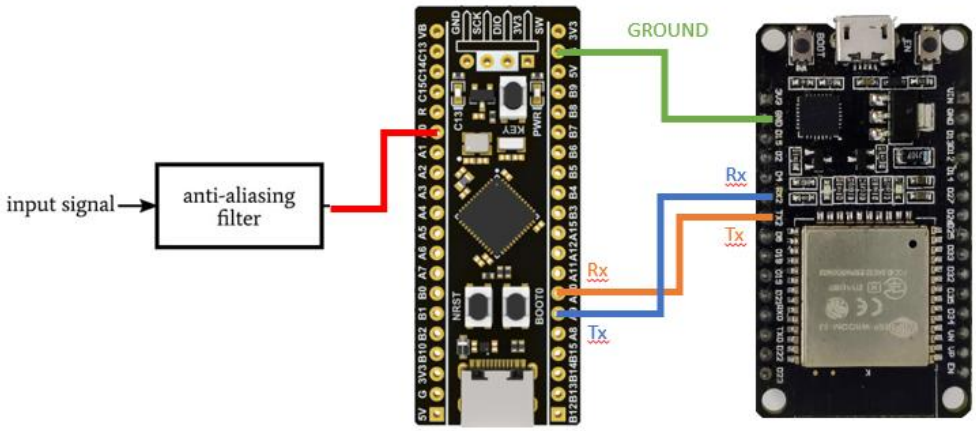
Figura 31 - Configuração final da UART no STM32CubeIDE



Fonte: O autor (2024).

Já na ESP32, a UART2 (pinos RX2 e TX2) foi selecionada para realizar a comunicação com o microcontrolador STM32F411. Assim, foi possível realizar a conexão dos pinos da UART entre a STM32 e ESP32, a ilustração dessa conexão é representada pela Figura 32.

Figura 32 - Esquemático das conexões entre o filtro, STM32F411 e ESP32



Fonte: O autor (2024).

3.4. Programação STM32F411

‘ O propósito do microcontrolador da ST é converter os valores analógicos para digital por DMA e transmiti-los para a ESP32 via UART, para isso foi desenvolvido o código com esse propósito (Apêndice A).

Primeiramente, é ativada a comunicação UART (HAL_UART_Init(&huart1)) e, posteriormente, é realizada a primeira conversão do ADC por DMA (HAL_ADC_Start_DMA(&huart1, (uint32_t*)adc2, BUFFER_SIZE)) para poder ativar a interrupção (HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)). Na interrupção é realizada a troca de *buffer* circular e é transmitido o *buffer* salvo, enquanto o outro *buffer* é preenchido.

3.5. Programação ESP32

A programação do ESP32 foi realizada na plataforma PlatformIO, integrada ao Visual Studio Code, que oferece as mesmas funcionalidades que a ESP-IDF. No arquivo de configuração do microcontrolador, platformio.ini, foram ajustados os parâmetros conforme a Figura 33.

Figura 33 - Ajustes dos parâmetros ESP32

```
(env:esp32dev)
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 921600
upload_speed = 921600
upload_port = COM3
```

Fonte: O autor (2024).

Já o desenvolvimento do código (arquivo main.cpp) é ilustrado na Figura 34.

Figura 34 - Código da ESP32

```
#include <HardwareSerial.h>
#define VECTOR_SIZE 1000  // Tamanho do vetor
HardwareSerial mySerial(2); // Usa a UART1
uint8_t vetor(VECTOR_SIZE); // Vetor para armazenar os inteiros recebidos
uint64_t count=0;
void setup() {
    Serial.begin(921600);                    // Serial para depuração
    mySerial.begin(921600, SERIAL_8N1, 16, 17); // Defina os pinos de RX e TX
}
void loop() {
    Serial.print(count++); // Eixo X
    Serial.print(",");     // Separador
    Serial.println(mySerial.read());  // Eixo Y
}
```

Fonte: O autor (2024).

3.6. Programação Interface Gráfica (Python)

Para a interface gráfica, foi desenvolvido um código em Python (conforme detalhado no Apêndice B), utilizando a biblioteca matplotlib.h para a plotagem de gráficos. Essa biblioteca possibilitou a criação de animações que exibem a evolução do sinal ao longo do tempo. Foi definido um número específico de amostras a serem plotadas, a fim de proporcionar uma visualização mais clara do período da onda. Para a frequência de 10 kHz, foram necessárias 100 amostras para uma visualização adequada da forma da onda. Já para as frequências de 20 kHz e 40 kHz, optou-se por 41 amostras, o que se mostrou mais eficaz para a análise visual.

4. Resultados

Nesta seção foram abordados os resultados obtidos pela simulação e prática da metodologia aplicada, desde a passagem pelo filtro passa-baixa até a geração dos gráficos pela interface gráfica. Adicionalmente, serão discutidos os aspectos relacionados ao desempenho do filtro e a análise dos gráficos obtidos, como validação da frequência, tempo de conversão e amplitude. Por meio dessa análise crítica, será possível compreender a efetividade da captação dos sinais analógicos para a construção de um osciloscópio.

4.1. Filtro *anti-aliasing*

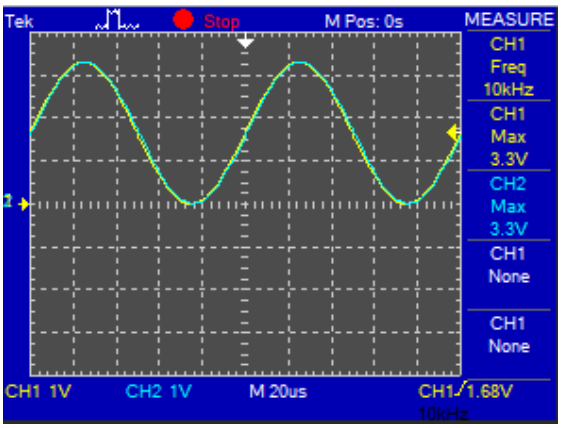
A atenuação das altas frequências provenientes de meios eletromagnéticos e de outros sistemas visa evitar o efeito de *aliasing* no ADC. Se o ADC capturar frequências indesejadas como se fossem parte do sinal original, ocasionará o efeito de *aliasing* no sinal e a resposta ficará comprometida. O filtro passa-baixa impede que frequências acima do limite do ADC interfiram no sinal amostrado, o que garante que o *datalogger* capture um sinal fiel ao original.

Logo, a importância dessa aplicação é prevenir e garantir que o sinal a ser estudado não sofra variações e, assim, prejudicando a sua leitura. Dessa forma, o sinal a ser lido passou pelo filtro *anti-aliasing*, resultando em duas respostas diferentes: uma obtida por meio da simulação e outra na prática, utilizando a *protoboard*.

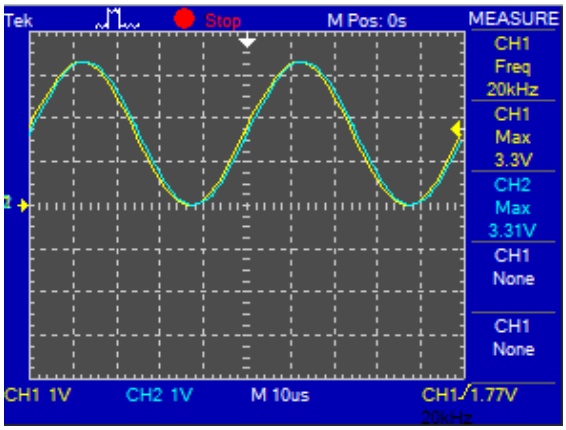
4.1.1. Simulação

Primeiramente, foi analisado o caso de sinais de entrada com formas de ondas senoidais, com offset de 1.65 V e amplitude pico-a-pico de 3.3V. As frequências utilizadas neste estudo foram as frequências de 10kHz, 20kHz e 40kHz. Os resultados da simulação são ilustrados pela Figura 35 nas suas respectivas frequências. Os sinais de entradas são representados pela cor amarela e os sinais de saída do filtro pela cor azul.

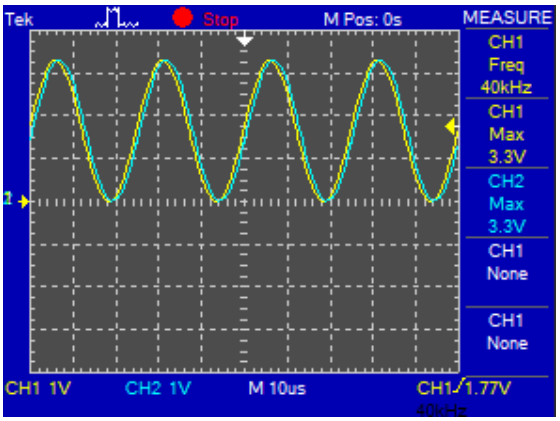
Figura 35 - Resultados da simulação do filtro anti-aliasing para a entrada de uma onda senoidal



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



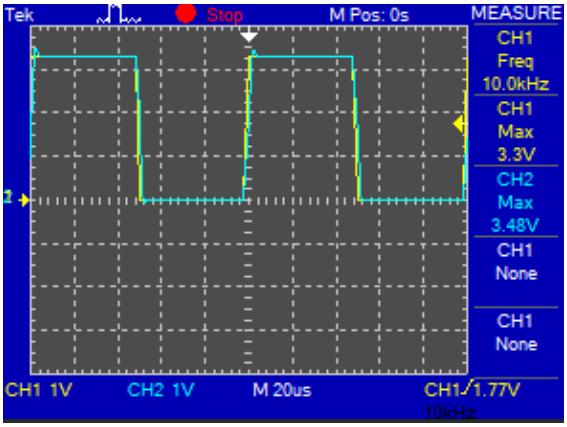
(c) Sinal de entrada de 40kHz

Fonte: O autor (2024).

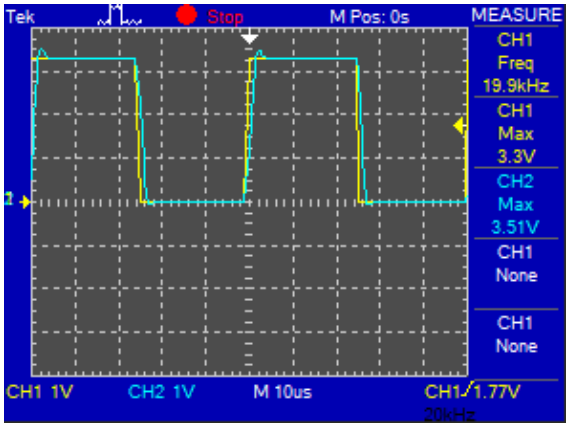
Foi observado que quanto maior a frequência, mais defasado ficou a resposta do filtro. Porém as características como amplitude, frequência e forma de ondas permaneceram as mesmas.

Partindo para a forma de onda quadrada, a resposta do filtro é ilustrada na Figura 36.

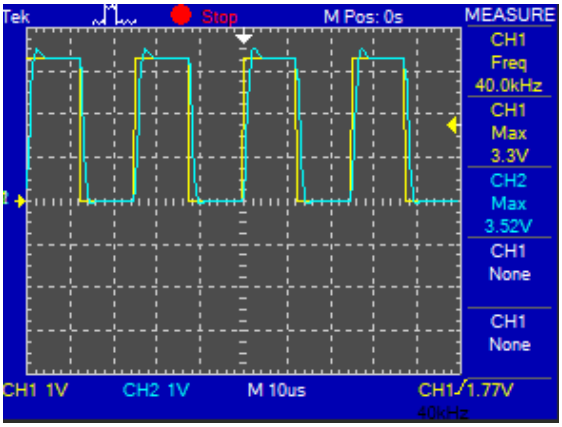
Figura 36 - Resultados da simulação do filtro anti-aliasing para a entrada de uma onda quadrada



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



(d) Sinal de entrada de 40kHz

Fonte: O autor (2024).

Foram observados “*ripples*” (oscilações) nas bordas de subida e descida da onda, causados pela atenuação das frequências mais altas pelo filtro passa-baixa. As bordas também ficaram mais suaves em comparação com o sinal original de onda quadrada. Na frequência de 40 kHz, os “*ripples*” e a suavidade das bordas foram ainda mais acentuados devido à atenuação de mais harmônicas em comparação com as demais frequências.

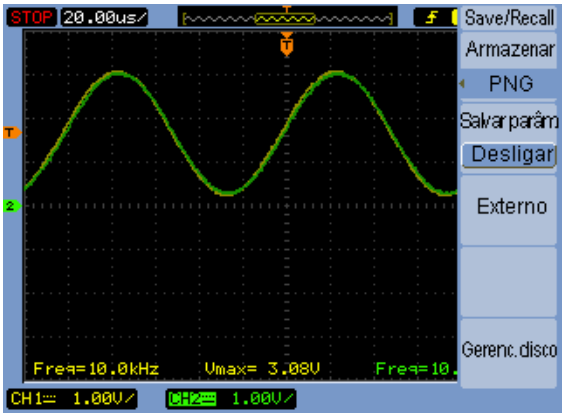
É importante destacar que a diferença no comportamento entre a onda senoidal e a onda quadrada está na composição de harmônicas. A onda senoidal representa apenas a frequência fundamental, enquanto a onda quadrada é composta por uma série de harmônicas ímpares, que juntas conferem sua forma característica. Quando a onda quadrada possui menos harmônicas em sua composição, são mais evidentes as oscilações e a suavidade nas bordas. Já na

onda senoidal, esse efeito não foi observado, pois ela consiste em uma única frequência fundamental, sem a contribuição de harmônicas.

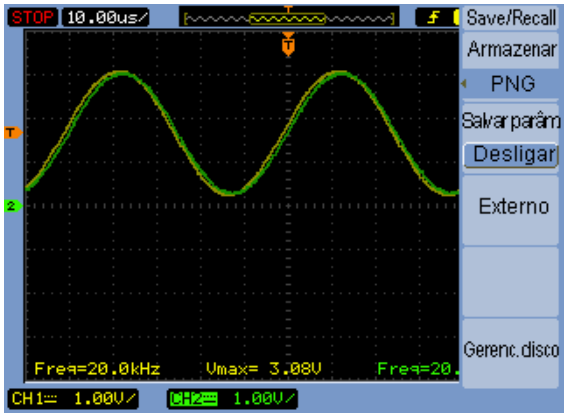
4.1.2. Prática (*Protoboard*)

Os resultados práticos foram obtidos através da aplicação de uma onda senoidal e também de uma onda quadrada de um gerador de funções, configurado com um offset de 1,65V e amplitude pico-a-pico de 3,3V nas frequências de 10kHz, 20kHz e 40kHz. Foi realizada a montagem do circuito na *protoboard*, conforme a Figura 37. Foi utilizada uma fonte de tensão contínua para alimentar o AO na tensão de $V^+ = 24V$ e $V^- = 0V$, simulando a fonte do kit didático de Soares (SOARES, 2024). Os resultados do filtro montado da onda senoidal são ilustrados na Figura 37. O sinal de entrada é representado pela cor amarela e o sinal de saída pela cor verde.

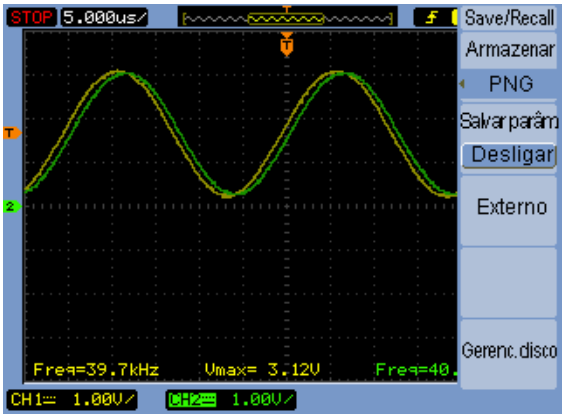
Figura 37 - Resultados da montagem prática do filtro anti-aliasing para a entrada de uma onda senoidal



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



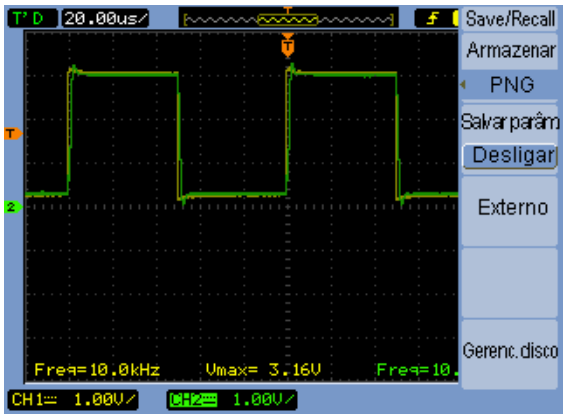
(c) Sinal de entrada de 40kHz

Fonte: O autor (2024).

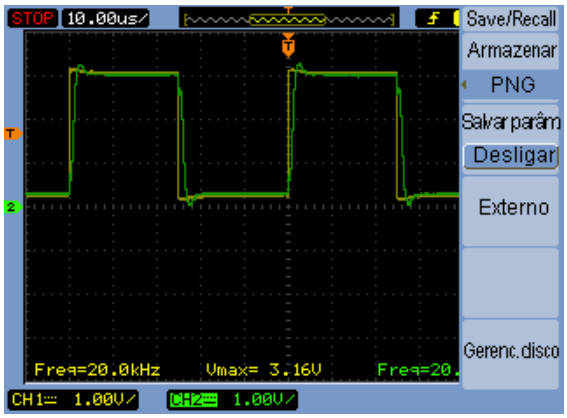
Conforme observado na simulação, verificou-se que, à medida que a frequência de operação aumentava, o sinal de saída apresentava uma defasagem crescente em relação ao sinal de entrada. No entanto, a amplitude, a frequência e o formato da onda de saída permaneceram consistentes com os do sinal original. Assim, o filtro demonstrou ser eficiente, preservando fielmente o sinal dentro da faixa de interesse, sem causar perdas significativas de amplitude ou distorções no formato da onda.

Já na aplicação da onda quadrada, foram obtidos os resultados ilustrados da Figura 38.

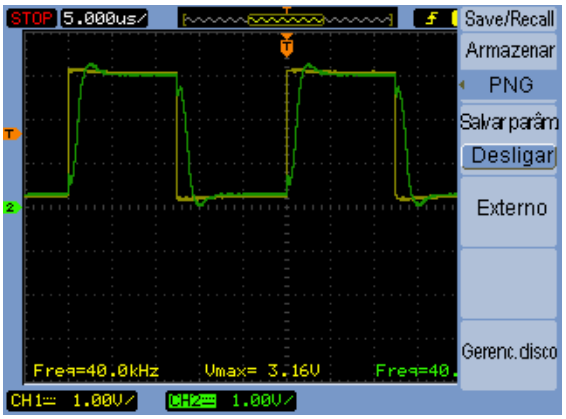
Figura 38 - Resultados da montagem prática do filtro anti-aliasing para a entrada de uma onda quadrada



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



(c) Sinal de entrada de 40kHz

Fonte: O autor (2024).

Semelhantemente aos resultados da simulação, foi possível visualizar o surgimento de *"ripples"* na subida e descida da onda quadrada, além da suavidade nas bordas. Esse comportamento é mais notável no caso da frequência de 40kHz

em que filtrou mais harmônicas impares em comparação no das frequências de 10kHz e 20kHz. Os resultados obtidos na simulação e prática foram consideravelmente próximos.

Assim como observado na simulação, o surgimento de oscilações e suavização nas bordas da onda quadrada ocorreu devido à filtragem das harmônicas de ordem superior que compunham o sinal. Isso evidenciou que a resposta do filtro atenua as harmônicas ímpares de maior frequência, resultando nesse comportamento característico.

4.2. Interface gráfica

Ao aplicar uma determinada frequência na entrada do sistema projetado e executar o script em Python, foram gerados gráficos dos sinais processados. Como o foco deste estudo é o desenvolvimento de um *datalogger* para sinais analógicos, a estrutura da interface gráfica foi mantida básica. Para obter novos gráficos, foi necessário fechar o gráfico gerado, alterar a frequência do sinal de entrada e executar o script novamente para captar o sinal atualizado.

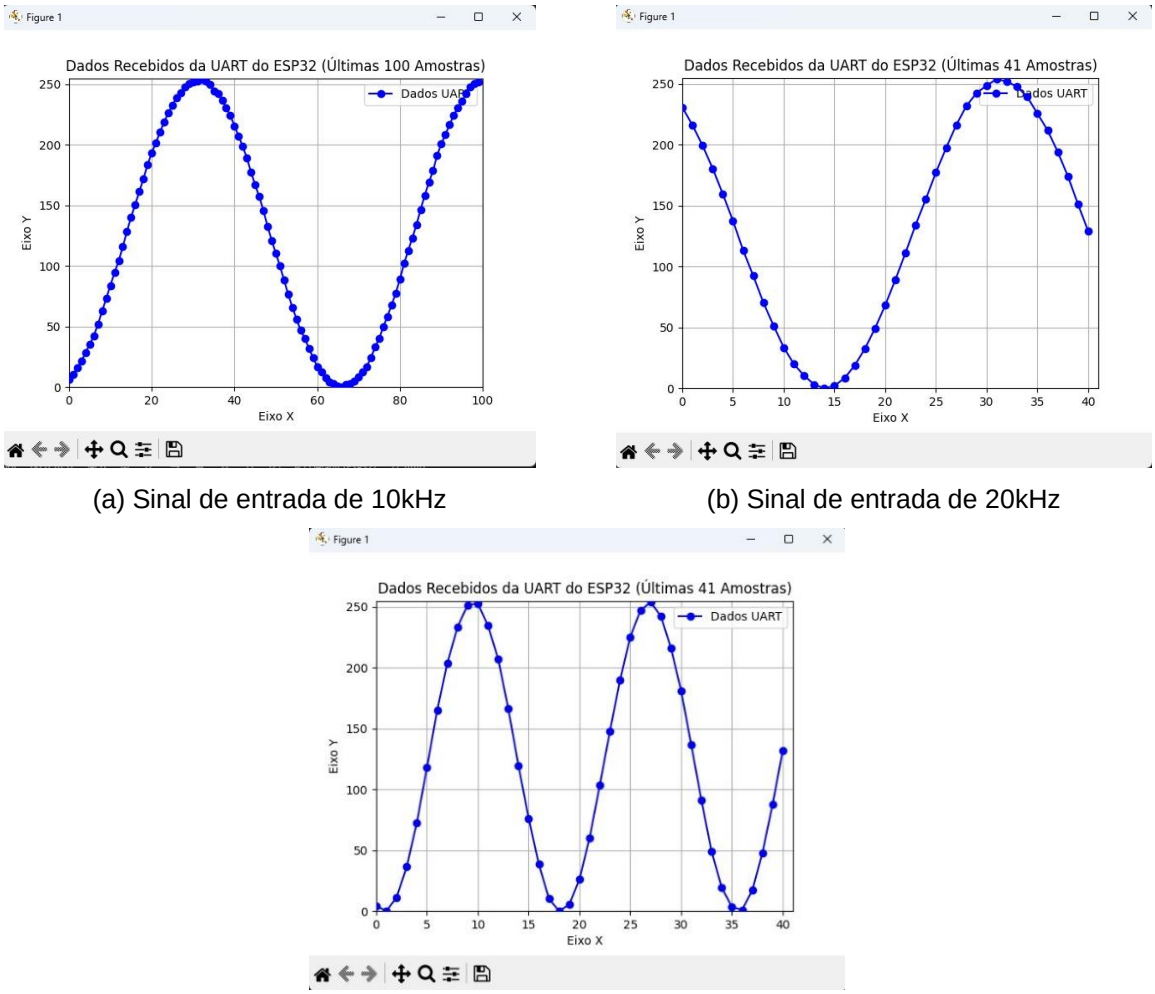
O eixo vertical diz a respeito da amplitude do sinal que foi quantificada em uma escala digital, expressa em bits, devido à conversão do sinal analógico para digital realizada pelo ADC (Conversor Analógico-Digital). Como o ADC foi configurado com uma resolução de 8 bits, ele divide a faixa de tensão de entrada (0 V a 3,3 V) em 256 níveis discretos, que vão de 0 a 255. Dessa forma, cada incremento de um nível representa um pequeno aumento na amplitude do sinal, aproximadamente 0,0129 V (3,3 V / 256). Essa quantização permite que a amplitude do sinal seja representada digitalmente, facilitando o processamento e a análise dos dados no microcontrolador e na interface gráfica.

Já o eixo horizontal foi determinado o tempo de plotagem de cada amostra. O tempo de conversão, calculado anteriormente, corresponde ao espaçamento entre cada amostra, que é de 1,44µs. Nessa análise, foram abordadas duas fases dos resultados: uma relacionada à entrada do sinal sem o filtro passa-baixa e outra com o filtro aplicado.

Com a devida montagem do filtro na *protoboard* e feita a sua ligação nos microcontroladores, conforme ilustrado na Figura 32. Primeiramente, foram inseridos os sinais senoidais e, em seguida, os sinais de onda quadrada nas frequências de

10 kHz, 20 kHz e 40 kHz. A Figura 39 ilustra os resultados obtidos com o filtro *anti-aliasing* aplicado ao sinal de entrada senoidal.

Figura 39 - Resultados do script Python da onda senoidal



(a) Sinal de entrada de 10kHz

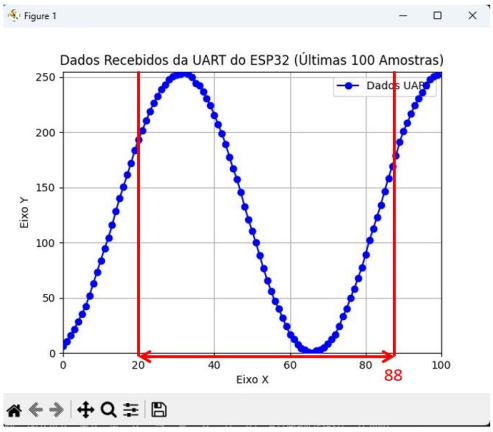
(b) Sinal de entrada de 20kHz

(c) Sinal de entrada de 40kHz

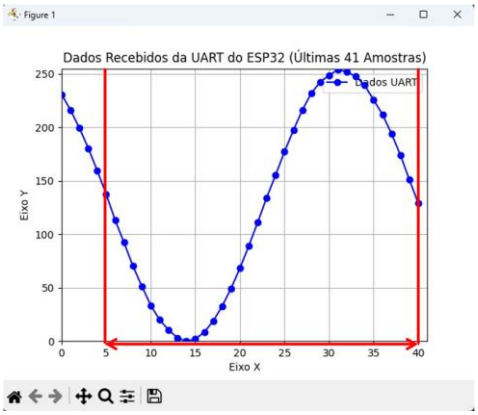
Fonte: O autor (2024).

Os resultados foram semelhantes aos dos sinais de entrada, pois, na saída do filtro, a onda senoidal não apresentou alterações na amplitude ou na forma de onda. Com esses resultados e de maneira análoga ao cálculo da frequência realizado no experimento anterior, foi possível avaliar se a frequência do sinal de saída também se manteve inalterada. Essa análise está representada na Figura 40, a partir da qual foi calculada a frequência do sinal de saída.

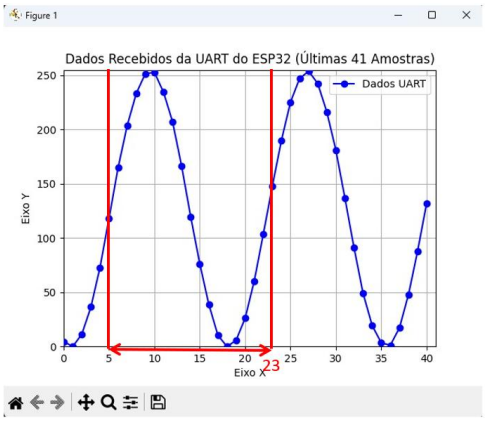
Figura 40 - Análise da frequência dos resultados da interface gráfica da onda senoidal



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



(c) Sinal de entrada de 40kHz

Fonte: O autor (2024).

Para os sinais de entrada com filtro de 10kHz, 20kHz e 40kHz, as frequências dos sinais de saída foram de, respectivamente:

$$f_{10kHz} = \frac{1}{1{,}44\mu s \cdot (88 - 20)} = 10{,}212kHz$$

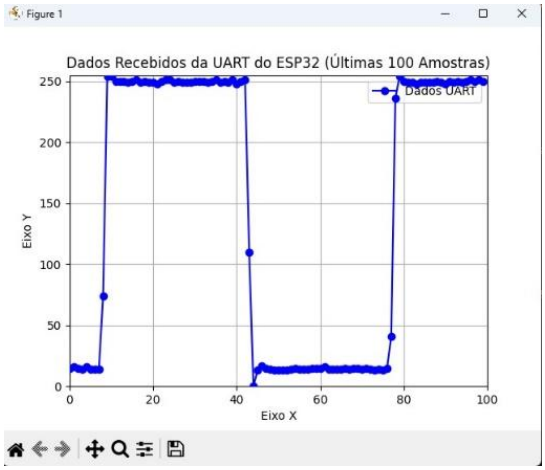
$$f_{20kHz} = \frac{1}{1{,}44\mu s \cdot (40 - 5)} = 19{,}841kHz$$

$$f_{40kHz} = \frac{1}{1{,}44\mu s \cdot (23 - 5)} = 38{,}58kHz$$

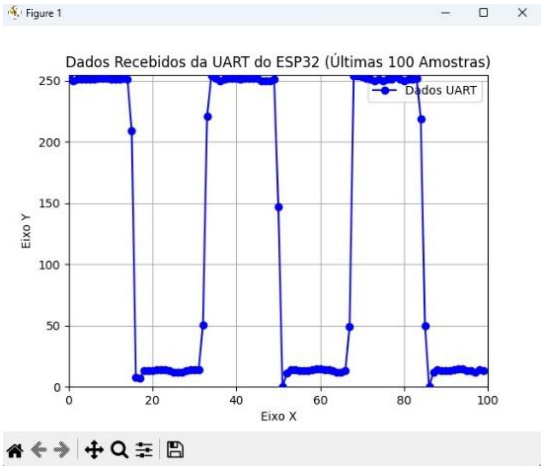
Portanto, as frequências dos sinais de saída mostraram-se próximas às frequências dos sinais de entrada.

Ao aplicar uma onda quadrada como sinal de entrada, obtiveram-se os resultados ilustrados na Figura 41.

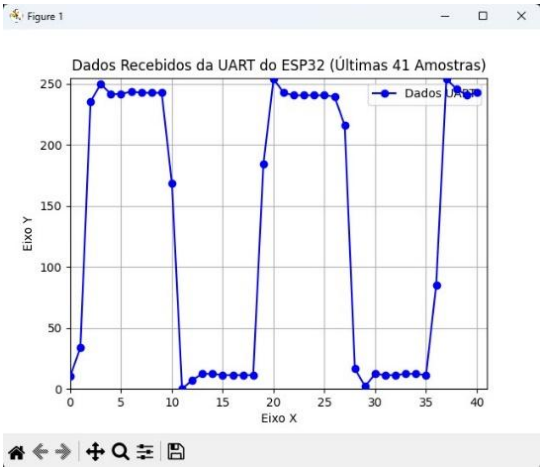
Figura 41 - Resultados do script Python da onda quadrada



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



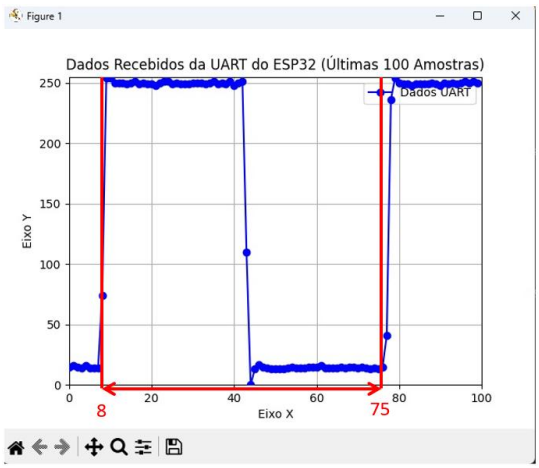
(c) Sinal de entrada de 40kHz

Fonte: O autor (2024).

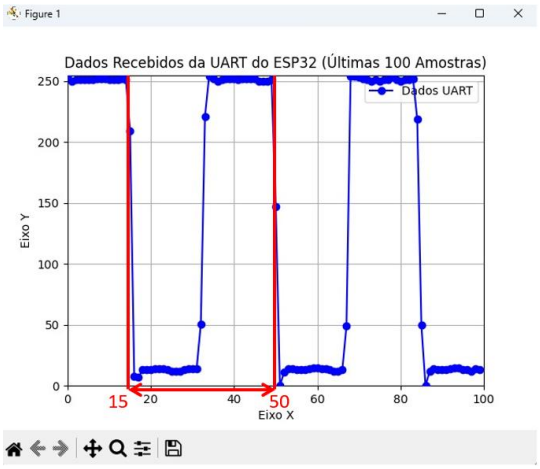
Nos resultados obtidos com o filtro passa-baixa, as harmônicas ímpares superiores foram atenuadas, resultando em uma suavização das bordas de subida e descida. Esse comportamento foi evidente, uma vez que, nas bordas de subida e descida, foram registradas amostras que não estariam presentes sem o filtro, conforme observado anteriormente. Além disso, nos resultados com o filtro, é notável a presença de *ripples* no sinal de saída.

Para validar os valores da frequência do sinal de saída em comparação com os do sinal de entrada, foi realizada a análise apresentada na Figura 42.

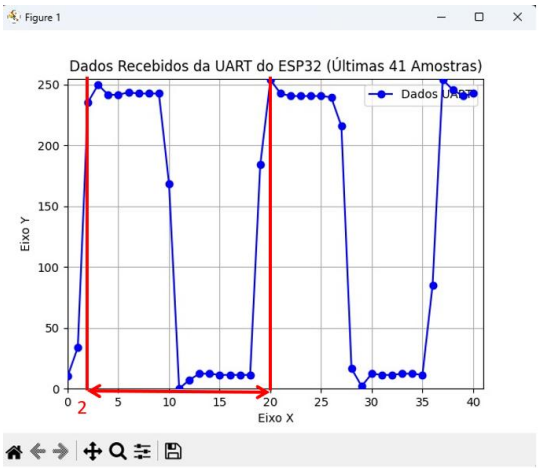
Figura 42 - Análise da frequência dos resultados da interface gráfica da onda quadrada



(a) Sinal de entrada de 10kHz



(b) Sinal de entrada de 20kHz



(c) Sinal de entrada de 40kHz

Fonte: O autor (2024).

Para os sinais de entrada de 10kHz, 20kHz e 40kHz, a respectiva frequência do sinal de saída foi de:

$$f_{10kHz} = \frac{1}{1{,}44\mu s \cdot (75 - 8)} = 10{,}364kHz$$

$$f_{20kHz} = \frac{1}{1{,}44\mu s \cdot (50 - 15)} = 19{,}841kHz$$

$$f_{40kHz} = \frac{1}{1{,}44\mu s \cdot (20 - 2)} = 38{,}58kHz$$

Os resultados da frequência do sinal de saída mostraram-se próximos aos valores do sinal de entrada aplicados.

5. Conclusão

O projeto desenvolvido implementou um *datalogger* de baixo custo para capturar e analisar sinais elétricos em tempo real, integrando os microcontroladores STM32 e ESP32 com uma interface gráfica em Python. Essa solução oferece uma alternativa acessível aos osciloscópios de bancada, que são fundamentais, mas custosos, para estudantes e profissionais de Engenharia Elétrica e Eletrônica.

A inclusão de um filtro *anti-aliasing* passa-baixa *Butterworth* mostrou-se essencial para manter a integridade do sinal, filtrando componentes de alta frequência que poderiam distorcer a conversão analógica-digital. Tanto nos testes simulados quanto na montagem prática, o filtro apresentou desempenho consistente, atenuando algumas harmônicas e preservando a amplitude e a forma do sinal dentro da faixa de interesse.

O uso do DMA associado ao ADC do STM32 foi crucial para atingir a frequência de amostragem necessária, otimizando o processamento e permitindo que a CPU se concentrasse em outras tarefas, o que elevou a eficiência do sistema. A transmissão de dados para o ESP32 via UART foi bem-sucedida, permitindo a análise e visualização do sinal na interface gráfica em Python.

Os sinais capturados puderam ser visualizados na interface gráfica, o que evidenciou o correto funcionamento do sistema de aquisição de dados e a precisão das leituras de sinal, mantendo suas características originais.

Em resumo, o *datalogger* desenvolvido atende às exigências de um equipamento de baixo custo, fácil implementação e funcionalidade para o contexto acadêmico. Este projeto fornece uma ferramenta prática para o ensino em laboratórios de Engenharia Elétrica e Eletrônica, incentivando o aprendizado dos alunos na construção e análise de sistemas de aquisição e processamento de sinais.

Para aprimorar o desempenho do *datalogger*, futuros trabalhos poderão focar em otimizar a saída do filtro, reduzindo a atenuação das harmônicas da onda quadrada e melhorando a resposta do sinal. Também será relevante avaliar o comportamento do sistema em uma gama mais ampla de frequências, testando sua robustez e versatilidade em diferentes aplicações. Além disso, o desenvolvimento de um software em Python com uma interface mais interativa e automatizada poderá melhorar a experiência do usuário, facilitando a análise e visualização dos dados.

Referências

ADRENALINE. **DMA: o que é Direct Memory Access e como funciona?** 2024. Disponível em: https://www.adrenaline.com.br/artigos/dma-o-que-e-direct-memory-access-e-como-funciona/. Acesso em: 8 out. 2024.

CARDOSO, L. F. **Osciloscópio de baixo custo utilizando a plataforma Arduino**. 2016

CORDEIRO, L. F.; GUÉRIOS, S. C.; PAZ, D. P. **Movimento Maker e a Educação: A tecnologia a favor da construção do conhecimento**. 2019. Disponível em: https://revistas.ifpr.edu.br/index.php/mundisociais/article/view/720/556. Acesso em: 8 out. 2024.

Datasheet ESP32. **ESP32 Series**. Espressif. Disponível em: https://www.ti.com/lit/ds/symlink/lm6134.pdf?ts=1728454081897&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em: 12 out. 2024

Datasheet LM6132BIM. **Datasheet LM6132, LM6134**. Texas Instruments. Disponível em: https://www.ti.com/lit/ds/symlink/lm6134.pdf?ts=1728454081897&ref_url=https%253A%252F%252Fwww.google.com%252F. Acesso em: 15 ago. 2024

DEWESOFT. **O que é o Conversor ADC (Conversor Analógico-Digital)**. Disponível em: https://dewesoft.com/pt/blog/o-que-e-conversor-adc. Acesso em: 8 out. 2024.

DIGIKEY. **Getting Started with STM32 - Working with ADC and DMA**. 2019. Disponível em: https://www.digikey.com.br/en/maker/projects/getting-started-with-stm32-working-with-adc-and-dma/f5009db3a3ed4370acaf545a3370c30c. Acesso em: 12 out. 2024.

ESPRESSIF. **ESP-IDF**. 2024. Disponível em: https://www.espressif.com/en/products/sdks/esp-idf. Acesso em: 13 out. 2024

FOROUZAN, Behrouz A. **Data communications and networking**. 5th ed. Boston: McGraw-Hill, 2012.

HIGUTI, R. T. **Amostragem e Reconstrução de Sinais**. UNESP. Disponível em: https://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/ele1095_3_amostragem.pdf. Acesso em: 6 out. 2024.

KEIM, R. **Anti-aliasing filters: applying sampling theory to ADC design**. 2020. Disponível em: https://www.allaboutcircuits.com/technical-articles/anti-aliasing-filters-applying-sampling-theory-to-adc-design/. Acesso em: 7 out. 2024.

Last Minute Engineers. **ESP32 Pinout Reference**. Disponível em: https://lastminuteengineers.com/esp32-pinout-reference/. Acesso em: 12 out. 2024

LATHI, B. P.; DING, Zhi. **Sistemas de Comunicações Analógicos e Digitais Modernos** - 4ª edição, LTC, 2012

Learning about Eletronics. **USB Transfer Types (Endpoint Types)- Explained**. Disponível em: https://www.learningaboutelectronics.com/Articles/USB-transfer-types.php. Acesso em: 9 out. 2024

MALVINO, Albert. **Eletrônica** [recurso eletrônico] / Albert Malvino, David J. Bates; tradução: Antonio Pertence Jr. – 8. ed. – Porto Alegre: AMGH, 2016.

MATTEDE, H. **O que é um osciloscópio e para que serve?** 2024. Disponível em: https://www.mundodaeletrica.com.br/o-que-e-osciloscopio-e-para-que-serve/. Acesso em: 10 out. 2024.

Mouser Electronics. **STM32F411 BlackPill Development Board – DFRobot**. 2021. Disponível em: https://br.mouser.com/new/dfrobot/dfrobot-stm32f411-blackpill-board/. Acesso em: 13 out. 2024

OKAWA Electric Design. **Filter Design and Analysis**. Disponível em: http://sim.okawa-denshi.jp/en/Fkeisan.htm. Acesso em: 6 set. 2024

PEREIRA, G. P.; CHAARI, M. Z.; DAROGE, F. **IoT-Enabled Smart Drip Irrigation System Using ESP32**. IoT, v. 4, n. 3, p. 221–243, 2023. Disponível em: https://doi.org/10.3390/iot4030012. Acesso em: 10 nov. 2024.

PHILLIPS, Travis. **Hardware Hacking: Interfacing to UART with Your Computer**. 2023. Disponível em: https://www.secureideas.com/blog/hardware-hacking-interfacing-to-uart-with-your-computer. Acesso em: 9 out. 2024

POMILIO, J. A. **Distorções da Forma de Onda**. 2013. Disponível em: https://www.dsce.fee.unicamp.br/~antenor/pdffiles/QEE%20Aero/Modulo3.pdf. Acesso em: 7 nov. 2024

PREDEBON, J. V.; BECKER, W. M. **WIFISCOPE 1.0**. 2018.

RAZAVI, Behzad. **Fundamentos de microeletrônica**. Tradução: J. R. Souza. 2. ed. Rio de Janeiro: LTC, 2017.

ROHDE & SCHWARZ. **Compreender a operação básica de osciloscópio**. 2024. Disponível em: https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/essentials-test-equipment/digital-oscilloscopes/compreender-operacao-basica-de-osciloscopios_254512.html#gallery-7. Acesso em: 12 out. 2024

SACCO, Francesco. **Comunicação SPI – Parte 1**. 2014. Disponível em: https://embarcados.com.br/spi-parte-1/. Acesso em: 12 out. 2024

SHOU, B. M. **Osciloscópio Digital Portátil de Baixo Custo**. 2011.

SOARES, R. S. **Kit Didático para as Disciplinas de Instrumentação Industrial I e II**. 2023.

STMicroelectronics. **STM32F411**. Disponível em: https://www.st.com/en/microcontrollers-microprocessors/stm32f411.html. Acesso em: 26 out. 2024

Texas Instruments. **Filter Design Tool**. Disponível em: https://webench.ti.com/filter-design-tool/filter-type. Acesso em: 16 set. 2024

APÊNDICE A – Código da STM32F411

```
#include "main.h"

#define BUFFER_SIZE 1000

uint8_t inicio;

uint8_t adc1(BUFFER_SIZE);

uint8_t adc2(BUFFER_SIZE);

uint8_t* inactive_buffer = adc1;

volatile uint8_t active = 1;

uint8_t count = 0;


void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)

{

  HAL_ADC_Stop_DMA(&hadc1);

  if(active){

            active = 0;

            inactive_buffer = adc2;

            HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc1, BUFFER_SIZE);

  } else{

            active = 1;

            inactive_buffer = adc1;

            HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc2, BUFFER_SIZE);

  }

  HAL_UART_Transmit(&huart1, (uint8_t*)inactive_buffer, BUFFER_SIZE,1);

}

int main(void)

{

  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();

  MX_DMA_Init();

  MX_ADC1_Init();

  MX_USART1_UART_Init();

  HAL_UART_Init(&huart1);

  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc2, BUFFER_SIZE);

  while (1)

  {

  }

}
```

APÊNDICE B – Código da interface gráfica em Python

Arquivo: plot.py

```
import serial

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

# Configuração da comunicação UART (ajuste o nome da porta e baud rate)

ser = serial.Serial('COM3', 921600, timeout=1)  # Ajuste para sua porta correta (ex: /dev/ttyUSB0 no Linux)

# Inicializar listas para armazenar os dados

x_data = ()

y_data = ()

# Definir o número máximo de amostras

max_samples = 100

# Inicializar a figura do gráfico

fig, ax = plt.subplots()

line, = ax.plot((), (), 'bo-', label='Dados UART')  # 'bo-' indica linha azul com marcadores circulares

# Limites iniciais dos eixos

ax.set_xlim(0, 100)

ax.set_ylim(0, 255)

# Configurações do gráfico

plt.xlabel('Eixo X')

plt.ylabel('Eixo Y')

plt.title('Dados Recebidos da UART do ESP32 (Últimas 100 Amostras)')

plt.grid(True)

plt.legend()

# Função para inicializar o gráfico

def init():

    line.set_data((), ())

    return line,

# Função para atualizar os dados continuamente

def update(frame):

    # Ler uma linha de dados via UART (esperando formato "x,y")

    line_data = ser.readline().decode('utf-8', errors='ignore').strip()

    if line_data:

        try:

            # Separar os valores de x e y recebidos

            x, y = map(float, line_data.split(','))

            # Adicionar os novos valores às listas

            x_data.append(x)

            y_data.append(y)

            # Limitar os dados para manter apenas os últimos 100 pontos

            if len(x_data) > max_samples:

                x_data.pop(0)  # Remove o primeiro elemento (mais antigo)

                y_data.pop(0)  # Remove o primeiro elemento (mais antigo)

            # Atualizar os limites dos eixos para incluir os novos dados

            ax.set_xlim(min(x_data), max(x_data) + 1)
```

```
            ax.set_ylim(min(y_data), max(y_data) + 1)
            # Definir os novos dados para a linha do gráfico
            line.set_data(x_data, y_data)
        except ValueError:
            pass  # Ignorar dados mal formatados
    return line,
# Animação para atualizar o gráfico continuamente
ani = FuncAnimation(fig, update, init_func=init, blit=True, interval=1)
# Mostrar o gráfico
plt.show()
# Fechar a conexão UART ao final
ser.close()
```