III MOSTRA DE CIÊNCIA, TECNOLOGIA E INOVAÇÃO
04 de dezembro de 2024
URI ERECHIM

CIÊNCIA DA COMPUTAÇÃO

PEDALEIRA ELETRÔNICA COM CONVERSÃO ANALÓGICO DIGITAL, PROCESSAMENTO DE SINAL DIGITAL EM TEMPO REAL E INTERFACE USB

Autores: Henri A. Grzegozeski, André Vítor Barbosa Schenato, Levi da Rosa Gomes, Ruan Dalla Rosa
Orientadores: Neilor Avelino Tonin, Marcos André Lucas

***RESUMO**: Este trabalho aborda o desenvolvimento de uma pedaleira eletrônica com funcionalidades de amplificação de sinal digital, conversão analógico-digital e manipulação de sinais de áudio em tempo real. O projeto teve como objetivo aplicar conceitos de arquitetura de computadores e sistemas digitais, utilizando componentes como o amplificador MCP6024, o conversor ADC MCP3008 e a microcontroladora ESP32. O sinal digitalizado foi transmitido para um computador via serial e reproduzido com um programa em Python, validando a precisão e a distinção tonal de cada corda de uma guitarra.*

***Palavras Chave:** Pedaleira Eletrônica, Conversão Analógico-Digital, ESP32*

1. INTRODUÇÃO

Este projeto integrador tem como objetivo a construção de uma pedaleira eletrônica experimental, aplicada ao processamento de sinais de áudio de instrumentos musicais de captação elétrica, com foco em consolidar os conhecimentos adquiridos nas disciplinas de Arquitetura de Computadores e Sistemas Digitais. A pedaleira foi projetada para captar, digitalizar e processar sinais sonoros, explorando as etapas de conversão analógico-digital, manipulação digital do áudio e reprodução.

Para alcançar esses objetivos, o projeto utiliza componentes de captura e conversão de sinal, como um conversor ADC e um microcontrolador, permitindo a análise e processamento do áudio digitalizado. A implementação do sistema de manipulação e transmissão dos dados digitais foi estruturada para oferecer uma base sólida para a diferenciação de frequências e para experimentos futuros com efeitos sonoros e aprimoramento de fidelidade.

2. OBJETIVOS GERAIS E ESPECÍFICOS

O objetivo geral do presente trabalho é a aplicação prática dos conceitos apresentados nas disciplinas de Arquitetura de Computadores e Sistemas Digitais através de um estudo orientado à construção de uma pedaleira eletrônica.

Os objetivos específicos incluem:

- Mapeamento das tecnologias envolvidas para implementação
- Seleção e aprofundamento de conceitos e ferramentas
- Prototipação através de software para simulação de funcionamento
- Prototipação física para integração de componentes

- Desenvolver o software de controle para a ESP32 e um programa em Python para captura, transmissão e reprodução em tempo real do sinal digitalizado.

3. MATERIAIS E MÉTODOS

Para a implementação deste projeto, foi realizada uma pesquisa inicial em fóruns e comunidades especializadas em elétrica e eletrônica, com o intuito de identificar as melhores abordagens para o desenvolvimento da pedaleira. Durante essa fase, diferentes possibilidades de construção foram analisadas, levando em consideração aspectos como viabilidade financeira, acessibilidade dos componentes e a complexidade teórica envolvida. Esses fatores foram cruciais para a seleção dos materiais mais adequados ao projeto.

Durante a fase de prototipação, várias ferramentas de simulação foram avaliadas para validar os circuitos e a manipulação dos sinais digitais e analógicos. O Logisim foi inicialmente utilizado para o estudo dos sinais digitais, devido à sua interface intuitiva e foco em lógica digital. Para os circuitos analógicos, o CircuitJS se mostrou mais adequado ao fornecer uma visualização clara do comportamento elétrico, como corrente e voltagem em tempo real. No entanto, o LTspice destacou-se pela precisão nas medições de tensão, já que oferece modelos detalhados de componentes específicos, como chips e amplificadores operacionais. Além disso, sua simplicidade e eficiência o tornaram uma escolha ideal em comparação com outras ferramentas mais complexas, como Proteus e KiCad.

A implementação física do protótipo ocorreu de forma incremental. Cada componente foi testado individualmente antes de ser integrado ao sistema geral, garantindo assim a funcionalidade de cada módulo e facilitando o diagnóstico de possíveis falhas. A preparação da microcontroladora não foi simulada por software, devido à sua complexidade, sendo testada diretamente no protótipo físico. O fluxo de dados foi configurado para a correta manipulação e processamento do sinal da pedaleira.

Os componentes utilizados para a montagem do protótipo final foram:

- 1 Protoboard
- 1 Fonte de alimentação 9V
- 1 Conector P10 fêmea
- 2 Amplificadores operacionais (LM358, MCP6024)
- 1 Conversor analógico-digital MCP3008
- 1 Microcontrolador ESP32
- 1 Resistor de 10kΩ
- 1 Resistor de 100kΩ
- Cabos e conectores diversos

As fases subsequentes do projeto incluíram o desenvolvimento de código para a microcontroladora, responsável pela aquisição e manipulação dos sinais digitais convertidos, bem como o envio dos dados processados via interface de comunicação ao computador.

4. REFERENCIAL TEÓRICO

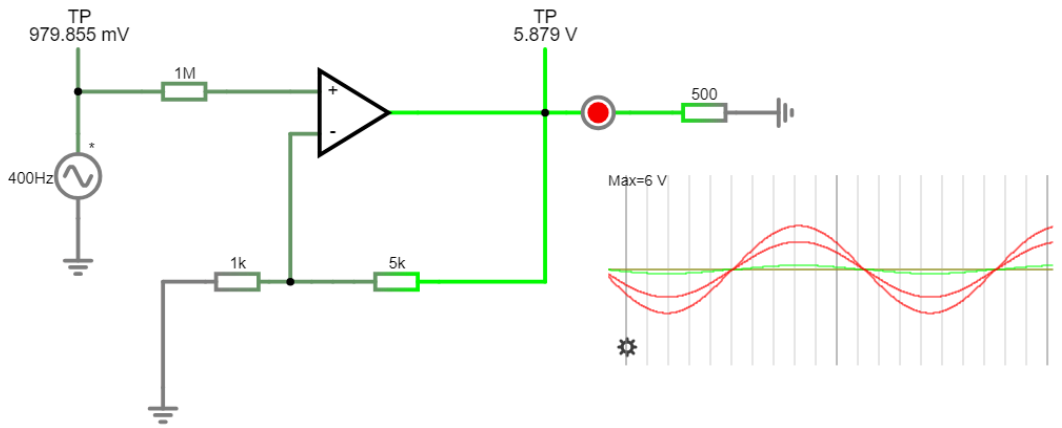
Para o desenvolvimento deste projeto, é imprescindível uma compreensão aprofundada dos conceitos e tecnologias que sustentam o funcionamento da pedaleira eletrônica. O referencial

teórico a seguir apresentará de forma organizada os principais componentes e processos que integram o sistema, começando pela arquitetura do hardware e os mecanismos de conversão analógico-digital, explorando diferentes arquiteturas de ADC, como Flash, SAR e Delta-Sigma. Em seguida, será abordado o processamento de sinais digitais, com foco em processadores e controladores, como DSPs, microcontroladores, FPGAs e microprocessadores, além dos barramentos e interfaces de comunicação, como SPI, I2C, MIDI e USB/UART. Por fim, serão discutidos os softwares e drivers necessários para a implementação, incluindo linguagens de programação como Assembly, C e Python, que permitem a integração eficiente do hardware ao sistema final.

4.1 ARQUITETURA DA PEDALEIRA

O fluxo de informação no sistema da pedaleira eletrônica segue um percurso bem definido. Inicialmente, o sinal de áudio gerado pela guitarra é captado e amplificado por um circuito pré-amplificador (Figura 1). Esse processo é essencial para garantir que o sinal tenha uma amplitude adequada para a posterior conversão. Após amplificado, o sinal é digitalizado por um conversor Analógico-Digital (ADC), que transforma o sinal analógico em dados digitais. Esses dados são então transmitidos ao microcontrolador por meio das portas GPIOs utilizando o protocolo SPI (Serial Peripheral Interface). O microcontrolador, por sua vez, se comunica diretamente com o computador via USB, permitindo a monitoração e manipulação dos dados em tempo real.

Figura 1 - Pré-Amplificador do Sinal



Fonte: Elaborado pelos Autores (2024)

A Figura 1 ilustra uma fonte de energia de ondas senoidais, gerado por simulação no software CircuitJS, passando inicialmente por um resistor de 1MΩ, o qual tem a função de reduzir interferências e minimizar ruídos indesejados. Em seguida, o sinal atravessa o amplificador operacional, configurado para operar em modo não-inversor, e a amplitude do sinal é ajustada, cuja escala de amplificação é definida pela configuração de resistores dispostos no circuito, assegurando uma amplificação controlada e precisa. A saída, finalmente, poderia entrar em um circuito de conversão ADC para dar continuidade à informação coletada.

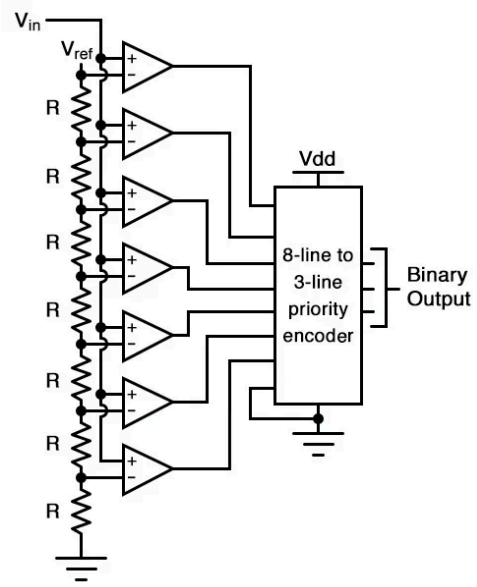
4.2 CONVERSÃO ANALÓGICO-DIGITAL

De acordo com Tocci (2011), a conversão de sinais analógicos para digitais (ADC - Analog-to-Digital Conversion) e de digitais para analógicos (DAC - Digital-to-Analog Conversion) são processos fundamentais em sistemas que lidam com sinais em ambas as formas. O ADC converte um sinal analógico contínuo em uma sequência de valores digitais discretos, representando a amplitude do sinal em instantes específicos de tempo. Esse processo é essencial para o processamento digital de sinais, onde os dados analógicos, como o som de uma guitarra, são convertidos em formato digital para que possam ser manipulados por microprocessadores, microcontroladores, FPGAs (Field-Programmable Gate Array) ou DSPs (Digital Signal Processors). Por outro lado, o DAC reverte esse processo, convertendo os dados digitais de volta para um formato analógico contínuo, permitindo que o sinal processado seja reproduzido em dispositivos de áudio, como amplificadores e alto-falantes.

A conversão de sinais analógicos para digitais em aplicações de áudio de alta qualidade é um processo sofisticado que envolve o uso de circuitos integrados (CIs) específicos, como ADCs dedicados. Os ADCs para áudio utilizam diferentes arquiteturas de conversão, como Flash ADCs, SAR (Successive Approximation Register) e Delta-Sigma.

Também chamado de conversor A/D paralelo, a arquitetura Flash é a mais simples de entender (Kuphaldt, 2007). Ela é formada por uma série de comparadores, cada um comparando o sinal de entrada a uma voltagem de referência única. As saídas do comparador conectam-se às entradas de um circuito codificador de prioridade, que então produz uma saída binária. A Figura 2 mostra um circuito ADC Flash de 3 bits. Este é o tipo de conversão AD mais rápido, constituído essencialmente por divisores resistivos e comparadores. A rapidez de conversão do Flash permite sua utilização em frequências da ordem de 100 GHz.

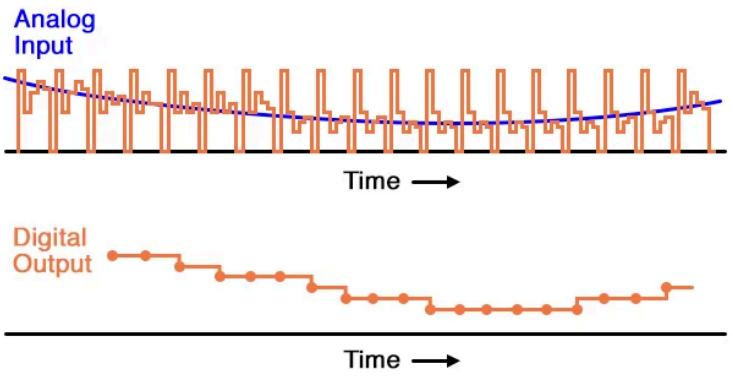
Figura 2 - Arquitetura de Conversão Flash



Fonte: Kuphaldt, 2007

A arquitetura SAR, por sua vez, possui um circuito demasiadamente complexo e para a presente pesquisa não se faz necessária a exposição. Entretanto, nota-se que ao analisar a comparação entre a entrada analógica e a saída digital (Figura 3) é nítida sua forma de funcionamento. É possível observar como as atualizações deste ADC ocorrem em intervalos regulares, se aproximando cada vez mais de um resultado binário equivalente ao decimal obtido na entrada.

Figura 3 - Comparação de sinais analógico e digital na arquitetura SAR

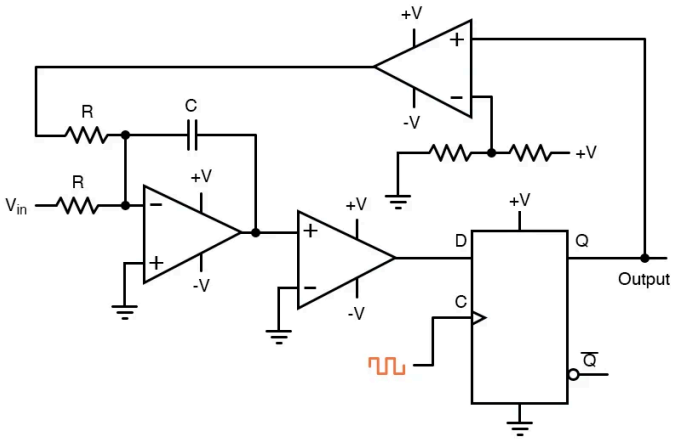


Fonte: Kuphaldt, 2007

Uma das tecnologias ADC mais avançadas é a chamada delta-sigma, ou ΔΣ (usando a notação de letras gregas apropriada). Em matemática e física, a letra grega maiúscula delta (Δ) representa diferença ou mudança, enquanto a letra grega maiúscula sigma (Σ) representa soma. Em um conversor ΔΣ, o sinal de tensão de entrada analógico é conectado à entrada de um integrador, produzindo uma taxa de mudança de tensão, ou inclinação, na saída correspondente à magnitude de entrada. Essa tensão de rampa é então comparada com o potencial de terra (0 volts) por um comparador.

O comparador atua como uma espécie de ADC de 1 bit, produzindo 1 bit de saída (“alto” ou “baixo”) dependendo se a saída do integrador é positiva ou negativa. A saída do comparador é então travada por meio de um flip-flop tipo D com clock de alta frequência e realimentada para outro canal de entrada no integrador, para conduzir o integrador na direção de uma saída de 0 volts.

Figura 4 - Arquitetura de Conversão Delta-Sigma



Fonte: Kuphaldt, 2007

Para Smith (2024), quando falamos de áudio de alta qualidade, a arquitetura Delta-Sigma é amplamente preferida devido à sua capacidade de proporcionar alta precisão e baixa distorção com taxas de amostragem elevadas e resolução de 24 bits. Um CI ADC típico para áudio, utilizando arquitetura Delta-Sigma, opera capturando o sinal analógico a uma taxa de amostragem alta (como 44,1 kHz, 96 kHz ou até 192 kHz), o que garante que o sinal original seja representado com alta fidelidade.

Dentro do CI, o sinal analógico é comparado continuamente com uma referência interna, e um filtro passa-baixa digital filtra o ruído, resultando em uma conversão precisa do sinal em uma sequência de bits. Para áudio de 24 bits, cada amostra do sinal é representada por 24 bits, o que oferece um alcance dinâmico superior e uma representação mais precisa das nuances do som. A qualidade do som final depende diretamente da precisão do ADC e do tratamento subsequente dos dados digitais, fazendo com que a escolha do CI e a arquitetura de conversão sejam cruciais para a excelência do áudio.

4.4 PROCESSADORES E CONTROLADORES

Os processadores e controladores desempenham um papel fundamental em dispositivos eletrônicos que exigem processamento de sinal em tempo real. No caso da pedaleira eletrônica, esses componentes são responsáveis por gerenciar e executar as instruções necessárias para processar os sinais de áudio capturados, garantindo que a manipulação dos efeitos ocorra sem atrasos perceptíveis. A escolha correta de um processador ou controlador adequado para a aplicação pode impactar diretamente no desempenho e na qualidade sonora do dispositivo, sendo, portanto, crucial o entendimento de suas características e arquiteturas.

4.4.1 DSP

O uso de Processadores de Sinal Digital (DSPs) na manipulação de sinais de áudio é uma abordagem que combina a flexibilidade dos sistemas digitais com a potência de processamento especializada (Torres, 2004). Os DSPs são projetados para realizar operações matemáticas

complexas em sinais digitais, como filtragem, equalização, compressão, reverberação e outros efeitos em tempo real. A principal vantagem de usar DSPs em pedaleiras e outros dispositivos de processamento de áudio é a capacidade de aplicar uma ampla gama de efeitos sem a necessidade de modificar fisicamente o hardware, permitindo que os músicos personalizem seus sons de maneira dinâmica e eficiente. Além disso, DSPs permitem a integração de algoritmos avançados, como convolução e modelagem física, que seriam impraticáveis de implementar com circuitos analógicos convencionais.

Um aspecto crucial na arquitetura dos DSPs é a adoção da arquitetura Harvard, com divergências em relação à de Von Neumann, utilizada em processadores de uso geral. A arquitetura Harvard se distingue pela separação física entre a memória de instruções e a memória de dados, permitindo que ambos sejam acessados simultaneamente. Isso resulta em um aumento significativo na eficiência e na velocidade de processamento, o que é particularmente importante em aplicações de áudio, onde o processamento em tempo real é essencial.

A separação das memórias na arquitetura Harvard também permite que o DSP execute múltiplas operações em paralelo, aproveitando ao máximo o pipeline de processamento. Isso é possível porque as instruções podem ser buscadas da memória de programa ao mesmo tempo em que os dados são lidos ou escritos na memória de dados, eliminando os gargalos comuns à arquitetura Von Neumann, onde um único barramento é usado para ambas as operações. Essa capacidade de realizar cálculos complexos de forma extremamente rápida é uma das razões pelas quais os DSPs são a escolha preferida em sistemas de áudio profissional e em pedaleiras de guitarra de alta performance.

Muitos DSPs modernos incluem unidades de processamento vetorial e suporte para operações em ponto flutuante, permitindo uma precisão ainda maior no processamento de sinais de áudio. Essas características tornam os DSPs ferramentas poderosas para a manipulação de áudio, possibilitando a criação de efeitos digitais de alta qualidade que podem ser aplicados em tempo real, sem latência perceptível.

4.4.2 MICROCONTROLADORES

De acordo com Tocci (2011), os microcontroladores são computadores compactos, integrados em um único circuito, que possuem CPU, memória e portas de entrada/saída, projetados para executar tarefas específicas dentro de um sistema maior. Diferentes dos microcomputadores de uso geral, os microcontroladores não podem ser programados pelo usuário final, sendo destinados a funções de controle automatizado em dispositivos como eletrodomésticos, sistemas automotivos, equipamentos médicos e industriais, entre outros. Eles desempenham um papel crucial em sistemas embarcados, controlando e operando funções previamente determinadas.

4.4.2.1 ESP32

A placa ESP32 é um microcontrolador de baixo custo e eficiente em termos de consumo de energia, desenvolvido pela Espressif Systems. Reconhecida por sua versatilidade, a ESP32 oferece suporte integrado para Wi-Fi, Bluetooth e outras tecnologias de conectividade. Essa combinação de funcionalidades torna a placa uma escolha popular para uma ampla variedade de aplicações, especialmente em projetos de Internet das Coisas (IoT), robótica, automação residencial e até entretenimento (MAKIYAMA, 2023).

Com um processador Xtensa LX6 dual-core de 32 bits, que opera a até 240 MHz, a ESP32 é capaz de realizar tarefas complexas com rapidez e eficiência. A placa conta com 520 KB de RAM, 4 MB de memória flash interna, além de uma variedade de periféricos, incluindo interfaces como UART, SPI, I2C e suporte para câmeras, o que a torna ideal para sistemas embarcados e de prototipagem. O foco principal de seu design é o desenvolvimento de dispositivos conectados, porém a flexibilidade do microcontrolador permite aplicações em diversas áreas.

A programação do ESP32 é feita, principalmente, nas linguagens C e C++, e ele é compatível com ambientes de desenvolvimento como Arduino IDE e ESP-IDF, da própria Espressif. Essa compatibilidade simplifica o processo de desenvolvimento, especialmente para quem já está familiarizado com o Arduino. Além disso, a placa suporta linguagens de alto nível, como MicroPython e Lua, o que facilita o uso por iniciantes e possibilita uma programação mais acessível.

A placa ESP32 foi criada com o intuito de permitir a criação de projetos inovadores, especialmente aqueles relacionados a IoT e automação residencial. Entre as suas aplicações mais comuns estão sistemas de monitoramento e controle, automação industrial, dispositivos de segurança, entretenimento e até mesmo projetos experimentais de ciência e educação. A placa pode ser usada, por exemplo, para construir um despertador inteligente que aciona outros dispositivos eletrônicos ao tocar, ou para desenvolver um sensor de fumaça que envia alertas para o smartphone em caso de emergência.

O ESP32 possui uma série de componentes que o tornam uma ferramenta versátil e adaptável para diferentes aplicações. As especificações do microcontrolador incluem um processador Xtensa LX6 dual-core de 32 bits, com clock de até 240 MHz, e duas memórias de cache: 32 KB de nível 1 e 16 KB de nível 2. No que diz respeito à memória disponível para o usuário, a placa conta com 520 KB de memória flash e 80 KB de RAM.

As capacidades de conectividade sem fio incluem suporte para Wi-Fi IEEE 802.11 b/g/n e Bluetooth 4.2, o que permite a comunicação com uma variedade de dispositivos e redes. A placa possui também uma ampla gama de periféricos, como 30 portas GPIO, duas interfaces UART, duas I2C, duas SPI, 12 canais de conversor analógico-digital (ADC), dois conversores digital-analógico (DAC), um RTC (Real-Time Clock) e um sensor de temperatura.

A ESP32 possui vantagens significativas em relação a outros microcontroladores populares, como o Arduino Uno. A placa ESP32 é equipada com um processador dual-core de 32 bits, que pode atingir até 240 MHz de frequência, enquanto o Arduino Uno possui um processador de 8 bits com frequência de até 16 MHz. Além disso, a ESP32 oferece suporte a Wi-Fi e Bluetooth integrados, eliminando a necessidade de módulos adicionais para comunicação sem fio, ao passo que o Arduino Uno não possui essas capacidades.

Outra vantagem do ESP32 é a sua maior quantidade de memória: são 520 KB de memória flash e 80 KB de RAM, em comparação com os 32 KB de memória flash e 2 KB de RAM do Arduino Uno. A placa ESP32 também suporta uma variedade de linguagens de programação, incluindo C, C++, MicroPython, Lua e JavaScript, tornando-a uma opção acessível para diferentes perfis de programadores e projetos.

Comparando com o ESP8266, seu antecessor, o ESP32 apresenta melhorias significativas em desempenho e funcionalidade. Enquanto o ESP8266 possui um único núcleo de processamento, o ESP32 conta com dois núcleos, o que proporciona maior capacidade de processamento e permite

a execução de múltiplas tarefas em paralelo. Além disso, o ESP32 suporta tanto o Bluetooth clássico quanto o Bluetooth Low Energy (BLE), oferecendo uma gama mais ampla de opções de conectividade. O ESP32 também possui maior estabilidade, mais portas programáveis e suporte avançado para criptografia, incluindo chaves RSA de 4096 bits, tornando-o mais seguro e adequado para aplicações que exigem proteção dos dados.

4.4.3 FPGA

De acordo com a Intel, o FPGA (Field Programmable Gate Array) é um circuito integrado semicondutor cuja funcionalidade pode ser modificada após a fabricação. Conforme explicado por Stephen Brown e Zvonko Vranesic (2008), os FPGAs possuem uma estrutura geral composta por uma rede de interruptores programáveis, permitindo que o circuito interno seja adaptado para implementar diferentes lógicas de acordo com a necessidade do projeto. Essa flexibilidade é valiosa durante o desenvolvimento de protótipos, pois o engenheiro pode re-programar o FPGA diversas vezes, ajustando ou ampliando suas funcionalidades conforme o hardware é testado. Devido à sua capacidade de realizar circuitos lógicos muito maiores que os chips comuns, e de serem ajustados para aplicações específicas, os FPGAs têm ampla aplicação na indústria.

4.4.4 MICROPROCESSADORES

De acordo com a IBM, o microprocessador é o tipo predominante de processador em computadores modernos, funcionando como uma espécie de "cérebro" do sistema ao combinar os componentes de uma unidade central de processamento (CPU) em um único circuito integrado. Ao contrário dos CPUs tradicionais, os microprocessadores integram funções aritméticas, lógicas e de controle em um único chip, tornando-os mais confiáveis ao reduzir pontos potenciais de falha. Esse avanço começou em 1971, com a introdução do Intel 4004, que revolucionou a computação ao consolidar múltiplos chips em um único dispositivo de uso geral. Atualmente, os microprocessadores são, de certa forma, indispensáveis, sendo possível encontrá-los em diversos dispositivos.

O funcionamento de um microprocessador envolve quatro etapas principais: buscar, decodificar, executar e armazenar instruções, com componentes essenciais como a unidade lógica aritmética (ULA), a unidade de controle, registros, cache de memória e transistores. Diversos tipos de microprocessadores são usados atualmente, como microcontroladores, processadores digitais de sinal (DSPs) e unidades de processamento gráfico (GPUs), cada um projetado para diferentes aplicações.

4.5 BARRAMENTOS E INTERFACES

Para o funcionamento integrado de dispositivos eletrônicos complexos, como a pedaleira eletrônica, é necessário garantir que todos os componentes se comuniquem de forma eficiente e sincronizada. Para isso, o uso de barramentos e interfaces desempenha um papel essencial, facilitando a transferência de dados entre processadores, controladores e outros módulos de hardware. Estes mecanismos de comunicação determinam não apenas a velocidade e eficiência do sistema, mas também a sua escalabilidade e flexibilidade para futuras expansões e melhorias (Sacco, 2014).

Diversas tecnologias de interligação serial entre dispositivos foram desenvolvidas, podendo ser separadas em duas grandes categorias, a comunicação síncrona e a comunicação assíncrona. Dentre os métodos de comunicação mais conhecidos, destacam-se três: SPI, I2C e UART. Há grandes diferenças mesmo entre os protocolos síncronos. Apesar do padrão de cada protocolo definir limites máximos de taxas, cada fabricante possui a liberdade de desenvolver dispositivos com suas velocidades. Abaixo temos um comparativo entre padrões de dispositivos seriais:

Tecnologia	**N° de Barramentos para Comunicação**	**Taxa Máxima**	**Fluxo de Dados**
UART (RS232)	2 (sem controle de fluxo)	115.200 bps	Half ou Full Duplex
SPI	3 + n° de Slaves	2 Mbps	Full Duplex
I2C	2 (até 127 dispositivos)	400 Kbps	Half Duplex

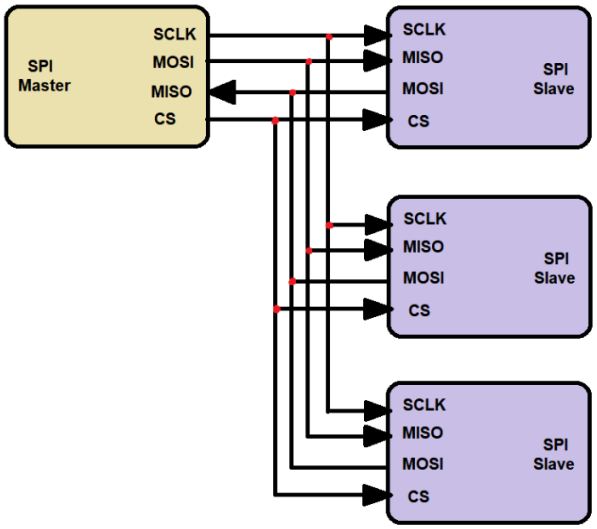
4.5.1 SPI

O protocolo de Interface Serial Periférica (SPI) é uma solução eficaz para comunicação serial entre dispositivos digitais, desenvolvido pela Motorola em 1979 e amplamente adotado desde então por fornecedores de circuitos integrados (KUNDU, 2014). Por sua simplicidade e flexibilidade, o SPI é popular em aplicações que envolvem comunicação entre microprocessadores, sensores, drivers de LCD e outros dispositivos periféricos. A seguir, detalha-se a arquitetura e funcionamento deste protocolo, além de suas especificações técnicas e aplicações.

4.5.1.1 Introdução ao SPI

O SPI (Serial Peripheral Interface) é uma interface de comunicação serial síncrona que opera em uma arquitetura mestre-escravo. Nessa configuração, um dispositivo controlador (mestre) gerencia a comunicação com um ou mais dispositivos periféricos (escravos) por meio de quatro linhas de sinal: MOSI, MISO, SCK e SS (Figura 5). Esta estrutura facilita a troca de dados com uma taxa de transferência elevada, ideal para sistemas que exigem comunicação rápida e eficiente (STAPLES E NIAZI, 2008).

Figura 5 - Comunicação do Protocolo SPI



Fonte: Kim, 2022

4.5.1.2 Arquitetura e Funcionamento

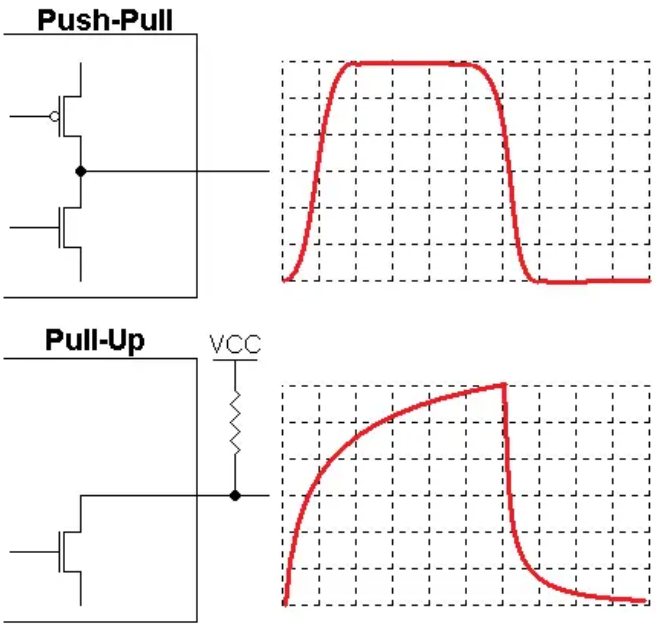
O SPI é conhecido por sua capacidade de transmissão full-duplex, permitindo comunicação simultânea entre mestre e escravo. As linhas de comunicação desempenham papéis específicos:

- **MOSI (Master Out Slave In)**: utilizada para o mestre enviar dados ao escravo.
- **MISO (Master In Slave Out)**: usada para o escravo enviar dados ao mestre.
- **SCK (Serial Clock)**: fornece o sinal de clock, gerado pelo mestre, que sincroniza a troca de dados entre os dispositivos.
- **SS (Slave Select)**: ativa individualmente cada dispositivo escravo, permitindo que o mestre selecione o periférico específico para a comunicação.

Essa configuração, com um clock dedicado, permite taxas de transferência na faixa de megahertz, tornando o SPI adequado para aplicações que exigem velocidade. Em sistemas com múltiplos dispositivos, o mestre controla o pino SS de cada escravo para simplificar a comunicação (KUNDU; KUMAR, 2014).

Os sinais de comunicação possuem uma direção fixa e definida. Isso significa que sempre existem dois transistores definindo o estado de um pino (Push-Pull). Essa característica é uma das grandes diferenças entre outras comunicações seriais como I2C e OneWire, que possuem um mesmo barramento de dados para os sinais de entrada e saída através do esquema de dreno-aberto (Pull-Up) (Figura 6).

Figura 6 - Comparação entre esquema Push-Pull e Pull-Up



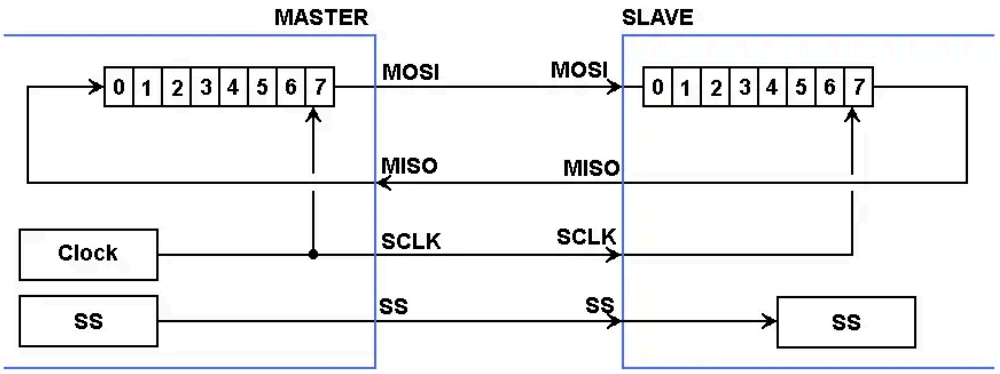
Fonte: Sacco, 2014

O esquema push-pull utiliza dois transistores para definir os estados de um pino, onde um transistor puxa o sinal para o nível alto e outro puxa para o nível baixo. Esse arranjo permite controle mais preciso do sinal e reduz distorções, tornando o sinal mais robusto e rápido, ideal para comunicações de alta velocidade, como o protocolo SPI. Em contraste, o esquema pull-up (usado em protocolos como I2C e OneWire) usa um único transistor de dreno-aberto que apenas conduz para o nível baixo; quando não conduz, uma resistência de pull-up mantém o nível alto. No SPI, a configuração push-pull para cada sinal de comunicação permite maior velocidade e menor interferência, pois elimina a necessidade de resistores externos para definir o estado do pino, além de oferecer uma transição de sinal mais rápida e estável.

De acordo com Sacco (2014), o princípio básico de um dispositivo SPI é o Shift-Register. Esse tipo de dispositivo faz a conversão de um registrador paralelo para sinais seriais de acordo com clock. Cada borda recebida no terminal de clock do dispositivo significa um bit transferido. Da mesma forma, esse tipo de dispositivo é capaz de receber dados vindos de maneira serial e convertê-los para um valor em um registrador paralelo.

A SPI não apenas é capaz de fazer essa conversão serial/paralelo, como também possui o gerador de clock, o controle para a troca do frame e o slave-select. Dessa forma, tornando ele um dispositivo de comunicação completo. Na figura abaixo podemos observar como ele se dispõe.

Figura 7 - Diagrama de Transferência de Dados SPI Interna



Fonte: Sacco, 2014

O Shift Register em ambos os dispositivos (Master e Slave) é a base para a transferência de dados. Ele permite que o Master e o Slave troquem informações simultaneamente, bit a bit, a cada pulso de clock. No exemplo do diagrama, ambos possuem um registrador de 8 bits (com posições de 0 a 7). Quando o clock sinaliza, um bit do MOSI é transferido do Master para o Slave, enquanto, ao mesmo tempo, um bit do MISO é enviado do Slave para o Master. Esse processo de "troca de bits" permite uma transferência bidirecional de dados em tempo real, em que cada dispositivo envia e recebe dados ao mesmo tempo.

O SPI permite ajustes na polaridade e fase do clock, definidos pelos parâmetros CPOL e CPHA, para garantir interoperabilidade entre dispositivos de diferentes fabricantes. O parâmetro CPOL determina o nível inicial do clock (alto ou baixo), enquanto CPHA define se a amostragem ocorre na borda de subida ou descida do clock. Essas combinações permitem até quatro modos de transmissão, proporcionando flexibilidade na configuração da interface para atender às exigências dos dispositivos conectados (KUNDU; KUMAR, 2014).

Além disso, a frequência do clock no SPI pode ser ajustada de acordo com a capacidade do sistema, sendo um fator chave para balancear a velocidade de comunicação e o consumo de energia. Essa flexibilidade é particularmente útil em projetos com restrições energéticas, como sistemas baseados em FPGA, onde é possível ajustar a frequência do clock para reduzir a dissipação de energia sem comprometer o desempenho (STAPLES E NIAZI, 2008). Em dispositivos móveis e sensores IoT, essa abordagem é crítica para aumentar a vida útil da bateria e minimizar o aquecimento.

4.5.1.5 Aplicações

Em microcontroladores que possuem suporte nativo ao SPI, como o Arduino UNO, bibliotecas como a SPI.h facilitam o desenvolvimento ao oferecer funções para configurar e gerenciar a comunicação de maneira eficiente. Em plataformas ARM, como o Mbed, a biblioteca SPI permite configurar os pinos e os modos de comunicação, simplificando ainda mais o desenvolvimento de firmware e aumentando a confiabilidade da comunicação SPI (Sacco, 2014).

Quando o microcontrolador não possui um módulo SPI integrado, é possível emular essa comunicação via software utilizando pinos GPIO para gerar os sinais MOSI, MISO e SCK, através de uma técnica conhecida por *bit banging*. Essa abordagem, embora funcione bem em situações

mais simples, demanda maior processamento e é mais vulnerável a interrupções, o que pode comprometer a qualidade e a estabilidade da transmissão, além de limitar a velocidade de comunicação.

Outra alternativa é utilizar bibliotecas específicas para controlar dispositivos específicos, como o conversor analógico-digital MCP3008. Nesse caso, a biblioteca oferece uma interface dedicada para acessar as funcionalidades do chip de maneira otimizada, tornando o controle mais direto e adequado às características do dispositivo.

Cada método apresenta diferenças em termos de complexidade, eficiência e desempenho. O uso de bibliotecas SPI nativas em microcontroladores com suporte ao protocolo SPI garante maior eficiência e velocidade, pois utiliza hardware dedicado. O bit banging, por outro lado, é uma solução mais simples e versátil, mas que consome mais recursos de processamento e apresenta limitações de velocidade. Já as bibliotecas específicas para um chip, como a do MCP3008, proporcionam uma comunicação mais eficiente com o dispositivo, mas a velocidade final ainda depende das capacidades do microcontrolador. Assim, a escolha do método adequado deve levar em conta o nível de exigência de desempenho, a capacidade de processamento e as particularidades do projeto.

4.5.1 I2C

No livro "I²C Bus: From Theory to Practice" de Dominique Paret, o I²C (Inter-Integrated Circuit) é descrito como um protocolo de comunicação serial síncrono de dois fios, amplamente utilizado para interligar microcontroladores e dispositivos periféricos. Ele foi desenvolvido pela Philips (agora NXP Semiconductors) e é projetado para simplificar a comunicação entre componentes em sistemas eletrônicos, particularmente em sistemas embarcados.

Paret destaca que o I²C permite a comunicação entre um dispositivo mestre e múltiplos escravos usando apenas duas linhas: uma linha de dados (SDA) e uma linha de clock (SCL). Isso o torna eficiente em termos de economia de pinos e cabos, o que é ideal para aplicações com limitação de espaço e complexidade. O protocolo é baseado em um esquema de endereçamento que permite ao mestre identificar e se comunicar com diferentes dispositivos conectados no barramento, sem a necessidade de um barramento dedicado para cada componente.

O autor também explica que o I²C opera em diferentes velocidades, que podem variar de 100 kHz no modo padrão até 3,4 MHz no modo de alta velocidade, com tempos de resposta que variam de acordo com a capacidade dos dispositivos e a velocidade de operação. Ele explora as limitações do protocolo, como o comprimento máximo do cabo e a quantidade de dispositivos que podem ser conectados no mesmo barramento, devido a restrições de capacitância e velocidade.

Dominique Paret enfatiza a versatilidade do I²C, especialmente em sistemas com sensores, displays, memória EEPROM e ADCs (conversores analógico-digitais), tornando-o uma escolha popular em uma vasta gama de aplicações industriais e de consumo.

4.5.2 MIDI

MIDI (Musical Instrument Digital Interface) é um protocolo de comunicação que permite a interação de instrumentos num ambiente digital (MIDI, 2024). O que o MIDI faz é criar uma série de sinais e controles que serão traduzidos - por outro instrumento ou num editor de áudio digital -

em som. De forma simplista, o MIDI gera uma espécie de pauta digital: indica as notas, a duração, a velocidade, a modulação, entre outros parâmetros. Mas, em vez de usar sinais numa folha de papel, usa comandos que instruem o receptor sobre como reproduzir o som. A vantagem é que tudo isto acontece sem latência, usando ficheiros muito pequenos, altamente editáveis, aos quais podem ser associados os sons que quisermos, desde baterias a sintetizadores. Um controlador MIDI não produz som por si mesmo. Apenas controla os sinais enviados para o programa ou sintetizador que usa essa informação para gerar o som.

Originalmente desenvolvido para permitir que músicos interconectem sintetizadores, o protocolo MIDI é agora difundido como um meio de comunicação para substituir ou suplementar o áudio digitalizado em jogos e aplicações multimídia. Há várias vantagens em se gerar som com um sintetizador MIDI ao invés de uma amostragem por disco ou CD-ROM. A primeira vantagem é o espaço de armazenamento. Arquivos de dados usados para armazenar digitalmente amostras de áudio em formato PCM (assim como .WAV), tendem a ser grandes. Isso se aplica especialmente a peças musicais longas gravadas com altas taxas de amostragem.

Arquivos de dados MIDI, por outro lado, são extremamente pequenos quando comparados com arquivos de amostragem de áudio. Por exemplo, arquivos contendo alta qualidade estéreo de amostragem de áudio requerem cerca de 10Mbytes de dados por minuto de som, enquanto uma sequência típica MIDI pode consumir menos que 10Kbytes de dados por minuto. Isso ocorre pois o arquivo MIDI não contém dados de amostragem de áudio, apenas contém as instruções necessárias para um sintetizador para reproduzir os sons. Essas instruções estão em forma de mensagens MIDI, as quais instruem o sintetizador sobre quais sons ele deve usar, quais notas tocar e quão alto tocar cada nota. O verdadeiro som é então gerado pelo sintetizador.

De acordo com Walker (1997), para computadores, o tamanho reduzido de arquivos também significa uma menor largura de banda no transporte de dados entre o PC e o periférico de saída que irá gerar o som. Outras vantagens de utilizar MIDI para gerar sons incluem a habilidade de editar facilmente a música, e a habilidade de mudar a velocidade de reprodução e o tom dos sons independentemente. Este último ponto é particularmente importante em aplicações de karaoke, onde a afinação e o tempo podem ser escolhidas pelo usuário.

O protocolo MIDI provê meios eficientes e padronizados de transmitir informação de performance musical como dados eletrônicos. A informação MIDI é transmitida em “mensagens MIDI”, as quais podem ser entendidas como instruções que orientam o sintetizador sobre como tocar determinado trecho de música. O sintetizador recebendo os dados MIDI deve gerar o verdadeiro som.

4.5.3 USB/UART

O USB (Universal Serial Bus) permite a conexão de periféricos de largura de banda média, como teclados, mouses, tablets, modems, telefones, unidades de CD-ROM, impressoras e outros periféricos externos de baixa a moderada velocidade em uma topologia estrela hierárquica (BUCHANAN, 2000). O Barramento USB simboliza um aperfeiçoamento em relação aos barramentos seriais e paralelos. Na prática ele trabalha da mesma forma que uma porta serial, ou seja, transmitindo os dados bit a bit, porém, em velocidades maiores que a comunicação serial (CUNHA, 2012).

Outra diferença do barramento USB em relação ao serial deve-se ao fato de o USB poder conectar até 127 equipamentos em cada micro com velocidades de transmissão de 1,5 Mbps a 4,8 Gbps, o que dá de 192 KB/s até 600MB/s. Tudo isso sem a necessidade de desligar o computador para fazer as ligações e com o reconhecimento automático dos aparelhos adicionados. É o Plug and Play em sua melhor forma.

A inclusão de interfaces USB em pedais de efeitos representa uma evolução significativa na integração de dispositivos analógicos com sistemas digitais modernos. A conexão USB permite que os pedais se comuniquem diretamente com computadores, tablets e outros dispositivos digitais, abrindo a possibilidade de controlar os pedais através de software, atualizar firmware, ou até mesmo gravar diretamente o sinal processado. Este tipo de interfaceamento simplifica o processo de integração dos pedais em setups de gravação digital, tornando-os mais versáteis e acessíveis para músicos que trabalham tanto em estúdios quanto ao vivo. A funcionalidade adicional, como a alimentação via USB, também reduz a necessidade de fontes de alimentação dedicadas, simplificando ainda mais o setup.

O UART (Universal Asynchronous Receiver/Transmitter) pode ser entendido como um dispositivo de hardware presente em microcontroladores e computadores que facilita a comunicação serial entre dispositivos (Roisenberg, 2023), ou um protocolo de comunicação que estabelece conjunto de regras para a troca de dados seriais entre dois dispositivos (R&D, 2024). Ele converte os bytes de dados em um fluxo serial de bits, transmitindo-os um a um sequencialmente, o que permite a comunicação ponto a ponto entre dispositivos.

Diferentemente de outros métodos de comunicação, como o SPI ou o I2C, o UART não exige um relógio de sincronização, operando de forma assíncrona. Cada dispositivo define sua própria taxa de transmissão, conhecida como baud rate, para garantir que tanto o transmissor quanto o receptor estejam em sincronia.

O protocolo UART é conhecido por sua simplicidade, pois requer apenas dois fios para a comunicação bidirecional: um para transmissão (TX) e outro para recepção (RX). Isso facilita a integração e reduz o custo do sistema de comunicação. Ao implementar a comunicação UART, os desenvolvedores precisam considerar a configuração dos bits de parada, bits de início e a paridade. Esses elementos garantem a integridade dos dados e ajudam a detectar possíveis erros de transmissão.

Essa interface pode ser usada para transmitir dados de controle, como ajustes de parâmetros ou status do pedal, para um processador central. A simplicidade e a baixa necessidade de pinos do UART tornam-no ideal para aplicações onde o espaço e a complexidade do circuito são limitados. Além disso, sua combinação com outras interfaces, como USB, possibilita a criação de sistemas híbridos que aproveitam os pontos fortes de diferentes tecnologias.

4.6 SOFTWARE DRIVER

No livro "Sistemas Operacionais: Projeto e Implementação" (Operating Systems: Design and Implementation), Andrew Tanenbaum aborda os drivers como componentes essenciais para o funcionamento de um sistema operacional, descrevendo-os como programas responsáveis pela comunicação entre o sistema operacional e os dispositivos de hardware. Ele explica que o papel dos drivers é atuar como uma camada de abstração que permite ao sistema operacional gerenciar

diferentes tipos de hardware sem precisar lidar diretamente com os detalhes específicos de cada dispositivo.

Tanenbaum enfatiza que os drivers são geralmente executados no modo kernel, o que lhes dá acesso direto aos recursos de hardware, como memória, registradores e dispositivos de E/S (entrada/saída). Devido a essa proximidade com o hardware e à necessidade de alto desempenho, os drivers são frequentemente escritos em linguagens de baixo nível, como C ou até mesmo Assembly, para garantir um controle eficiente dos recursos.

A dependência entre o driver e o dispositivo, ressaltando que cada dispositivo de hardware precisa de um driver específico para funcionar corretamente. Esse driver, por sua vez, deve ser compatível com o sistema operacional que o gerencia. O processo de comunicação entre o sistema operacional e o hardware ocorre por meio de instruções específicas do driver, que convertem as solicitações de alto nível (como leitura de dados de um disco) em comandos que o hardware pode executar. (TANENBAUM, 2000)

Os drivers também são responsáveis por gerenciar interrupções, que são sinais enviados pelos dispositivos ao processador para indicar que uma tarefa foi concluída ou que é necessário realizar alguma operação, como a transferência de dados. (TANENBAUM, 2000)

4.6.1 LINGUAGEM ASSEMBLY

A linguagem Assembly é uma linguagem de baixo nível diretamente relacionada ao hardware de um computador. Ela é uma representação simbólica do código de máquina, o conjunto de instruções que a CPU executa diretamente. (TANENBAUM, 2007)

Essa linguagem atua como uma interface entre o programador e o hardware, permitindo o controle dos recursos da CPU, como registradores, memória e unidade aritmética. Por ser uma linguagem de baixo nível, ela oferece um acesso detalhado ao funcionamento interno do processador, mas assim, sendo dependente da arquitetura do processador. (TANENBAUM, 2007)

Outra característica citada por Andrew é que cada uma de suas instruções é diretamente convertida em uma instrução de máquina, aproximando-a do modo como a CPU processa os dados. Esse fator permite que os programadores tenham um nível de controle e precisão sobre o desempenho do programa que não é possível em linguagens de mais alto nível. No entanto, esse controle vem com um custo: a assembly é notoriamente difícil de usar.

A desvantagem da programação em Assembly e a sua complexidade, suscetível a erro e muito mais difícil de aprender e manter do que as linguagens de alto nível. Devido a essas limitações, essa linguagem deve ser apenas usada em situações específicas, como no desenvolvimento de drivers, sistemas operacionais ou em outras áreas que demandam um controle direto e detalhado. (TANENBAUM, 2007)

4.6.2 LINGUAGEM C

No livro "Linguagem C: Completa e Descomplicada", de André Backes (2013), a linguagem C é apresentada como uma das mais fundamentais e importantes linguagens de programação, sendo amplamente utilizada em diversos contextos da computação. Para o autor, C é uma linguagem que combina simplicidade e poder, oferecendo ao programador um nível elevado de controle sobre o

hardware, o que a torna especialmente adequada para o desenvolvimento de sistemas operacionais, softwares embarcados e aplicações que exigem alta performance.

Backes (2013) caracteriza C como uma linguagem de baixo nível em comparação com outras linguagens de programação, pois permite uma interação mais direta com a memória e os recursos do sistema. Ao mesmo tempo, ele destaca que C possui um grau significativo de portabilidade, o que possibilita a criação de programas que podem ser executados em diferentes sistemas operacionais com poucas modificações. Ele considera a linguagem uma escolha versátil, eficiente e apropriada tanto para desenvolvedores iniciantes quanto para profissionais que buscam desempenho em seus projetos.

4.6.3 LINGUAGEM PYTHON

A linguagem de programação Python foi criada no final dos anos 1980 por Guido van Rossum, enquanto trabalhava no Centrum Wiskunde & Informatica (CWI), na Holanda. O projeto começou como uma iniciativa pessoal de Guido durante o período de Natal de 1989, com o objetivo de criar uma linguagem que equilibrasse simplicidade e poder, visando à facilidade de leitura e manutenção do código. A primeira versão pública do Python foi lançada em fevereiro de 1991, e a linguagem rapidamente ganhou aceitação devido à sua sintaxe clara e intuitiva, que facilita o aprendizado e o uso por programadores de todos os níveis de experiência (Lutz, 2013).

O nome "Python" foi escolhido em referência ao grupo de comédia britânico "Monty Python's Flying Circus", e não à serpente homônima. Guido van Rossum, admirador do grupo, buscava um nome que soasse divertido e diferente, refletindo a filosofia de design da linguagem, que prioriza a legibilidade e o prazer em programar (Zelle, 2018). A escolha da linguagem de ser interpretada e multiparadigma (suportando paradigmas procedurais, orientados a objetos e funcionais) foi essencial para ampliar sua flexibilidade e aplicação em diversas áreas da computação, como desenvolvimento web, automação de scripts e análise de dados.

Python se destaca por ser uma linguagem interpretada, o que significa que o código-fonte é executado diretamente por um interpretador sem passar por um processo de compilação prévio. Essa característica facilita o desenvolvimento rápido e a depuração de programas. Além disso, Python adota uma licença de código aberto desde seu lançamento, incentivando uma comunidade global de desenvolvedores a contribuir com melhorias para a linguagem, bem como a criação de bibliotecas e frameworks voltados para aplicações específicas, como aprendizado de máquina, processamento de imagens e redes neurais (Sweigart, 2016).

5. DESENVOLVIMENTO

Nesta seção, são apresentados os detalhes do desenvolvimento do projeto, desde a simulação dos circuitos no software até a implementação dos protótipos físicos. As simulações permitem testar e validar o comportamento dos componentes antes de construir o sistema final em hardware. Este capítulo explora o funcionamento dos protótipos em software e os passos práticos para a implementação real.

5.1 PROTÓTIPO EM SOFTWARE

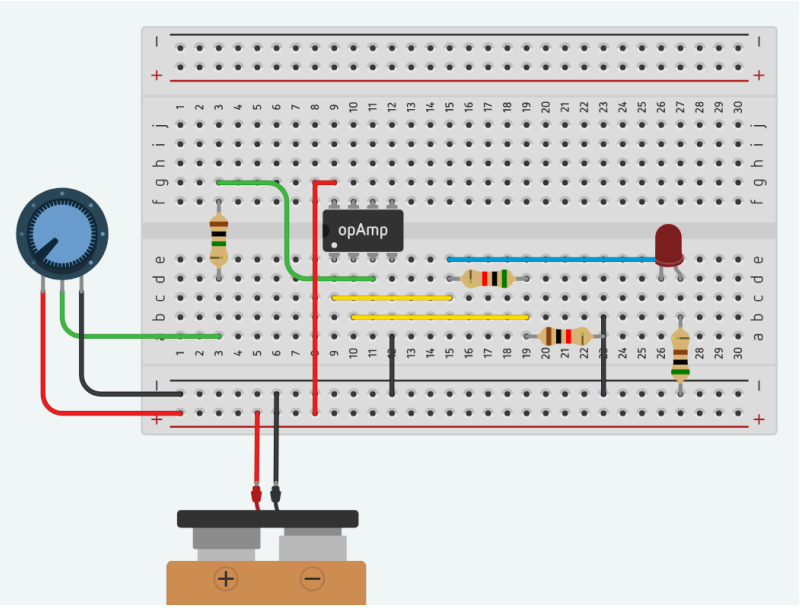
Para garantir o funcionamento conforme esperado, diferentes ferramentas de simulação foram utilizadas no processo de prototipação digital. CircuitJS, LTspice e TinkerCad foram escolhidos para simular diversos circuitos e a integração dos componentes. Esses softwares permitiram a visualização do comportamento elétrico e eletrônico em tempo real, ajudando a evitar possíveis falhas no momento da montagem física.

A simulação prévia em software oferece vários benefícios, como a prevenção de danos aos componentes físicos, a possibilidade de ajustar os valores de resistores, capacitores e fontes de alimentação, e a análise de diferentes cenários operacionais sem a necessidade de um ambiente laboratorial completo.

5.1.1 PROTÓTIPO DE AMPLIFICAÇÃO DE SINAL ANALÓGICO

O primeiro protótipo simulado foi o circuito de amplificação de sinal analógico, utilizando um amplificador operacional LM358 configurado em modo não inversor. A simulação foi realizada no TinkerCad, permitindo a visualização do comportamento do sinal ao passar pelo circuito. A figura 8 ilustra o circuito montado em uma protoboard simulada, composta por uma fonte de 9V, um potenciômetro para controle da tensão de entrada e um LED para indicação visual da saída amplificada.

Figura 8 - Sistema de Amplificação



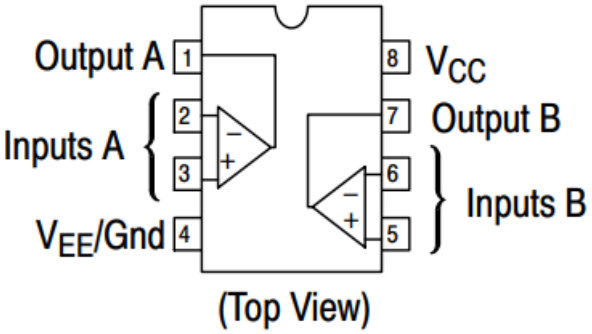
Fonte: Elaborado pelos Autores (2024)

O amplificador operacional LM358, mostrado na Figura 9, é um componente chave neste processo. Ele possui duas seções amplificadoras, A e B, permitindo a amplificação de dois sinais independentes ou em cascata. A pinagem do chip é descrita da seguinte forma:

Pino 1 (Output A): Saída do amplificador A.

Pino 2 (Input A-): Entrada inversora do amplificador A.
Pino 3 (Input A+): Entrada não inversora do amplificador A.
Pino 4 (VEE/GND): Conexão ao terra.
Pino 5 (Input B+): Entrada não inversora do amplificador B.
Pino 6 (Input B-): Entrada inversora do amplificador B.
Pino 7 (Output B): Saída do amplificador B.
Pino 8 (VCC): Conexão à tensão de alimentação positiva (geralmente 9V ou 12V).

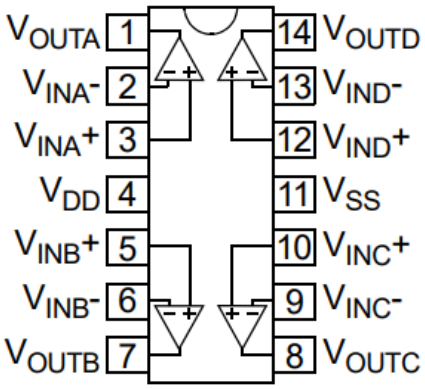
Figura 9 - Amplificador Operacional (LM358)



Fonte: Onsemi, 2024

Essa simulação foi essencial para validar o comportamento do circuito de amplificação, garantindo que o sinal de entrada fosse amplificado conforme os parâmetros do projeto. Ela permitiu avaliar a estabilidade do circuito e realizar ajustes antes da implementação física. Durante a montagem prática, o amplificador LM358 apresentou defeitos e precisou ser substituído pelo MCP6024, que possui um funcionamento semelhante. A principal diferença entre eles é que o MCP6024 contém mais amplificadores operacionais internos, embora, neste caso, apenas um tenha sido utilizado no circuito.

Figura 10 - Amplificador Operacional (MCP6024)



Fonte: Microchip, 2023

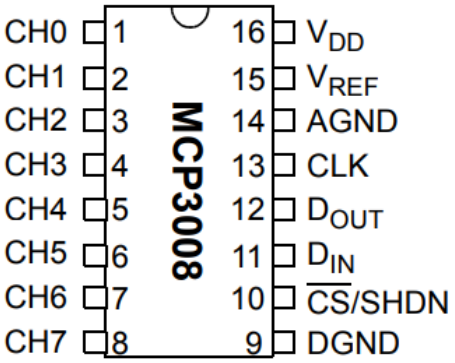
5.1.2 PROTÓTIPO DE CONVERSÃO ANALÓGICO DIGITAL

Para realizar a conversão do sinal analógico amplificado em digital, foi escolhido o CI MCP3008, um conversor Analógico-Digital (ADC) de 10 bits com 8 canais de entrada, ideal para leitura simultânea de múltiplos sinais analógicos. Este componente utiliza a interface SPI, permitindo uma comunicação rápida e eficiente com microcontroladores como a ESP32. Antes da implementação prática, a simulação do circuito foi realizada no software LTspice, permitindo avaliar o comportamento do sinal durante a conversão e sua transmissão ao microcontrolador.

A pinagem do MCP3008, conforme ilustrada na Figura 11, contém os pinos de entrada para sinais analógicos e as conexões para a interface SPI, que é responsável pela comunicação com o microcontrolador. A seguir estão as descrições detalhadas dos principais pinos:

Pinos CH0 a CH7: Entradas analógicas, onde os sinais a serem digitalizados são aplicados.
Pino 16 (VDD): Alimentação do chip.
Pino 15 (VREF): Tensão de referência para o ADC (define o intervalo de valores que o conversor pode ler).
Pino 14 (AGND): Terra analógico.
Pinos relacionados à comunicação SPI:
Pino 13 (CLK): Clock para sincronização dos dados.
Pino 12 (DOUT): Saída dos dados digitalizados.
Pino 11 (DIN): Entrada de dados de controle.
Pino 10 (CS/SHDN): Chip Select (habilita o CI para a comunicação SPI).
Pinos 9 (DGND): Terra digital.

Figura 11 - CI para Conversão Analógico-Digital (MCP3008)



Fonte: Microchip, 2023

Para a comunicação entre o MCP3008 e a ESP32, foi configurado o protocolo SPI usando as seguintes portas padrão da ESP32: pino 18 (CLK), pino 19 (MISO), pino 23 (MOSI) e pino 5 (CS). A biblioteca SPI.h foi utilizada na programação da ESP32 para facilitar a configuração da interface SPI e a leitura dos dados do MCP3008. Durante o processo de leitura, o MCP3008 converte o sinal analógico no canal selecionado (neste caso, CH0) em um valor digital de 10 bits, que é então enviado para a ESP32 pelo pino DOUT.

A ESP32, após receber os dados, os transmite para o computador via comunicação serial, possibilitando a análise em tempo real. Durante os testes, utilizou-se o Serial Plotter da IDE Arduino para visualizar o sinal digitalizado, o que permitiu ajustes e validações detalhadas nas amostragens coletadas.

5.1.3 INTERFACEAMENTO E MANIPULAÇÃO DE DADOS

Após a digitalização dos dados pelo MCP3008, a ESP32 processa e transmite as amostras para um computador por meio de um programa em C++ (Anexo I). Esse código gerencia a comunicação SPI com o ADC e envia as leituras de forma contínua pela interface serial. A configuração é otimizada para garantir uma taxa de amostragem adequada, com um intervalo de aproximadamente 125 microsegundos, para obter uma frequência de amostragem próxima de 8 kHz.

Para manipular e reproduzir esses dados digitalizados em tempo real, foi desenvolvido um script em Python (Anexo II) que utiliza a biblioteca PyAudio. Esse script recebe os dados transmitidos pela ESP32 através da porta serial (COM11, configurada com uma taxa de 115200 bps) e reproduz o sinal digitalizado em uma taxa de amostragem de 8000 Hz, utilizando um canal mono. Cada amostra é lida em dois bytes, convertida para 16 bits e enviada para a saída de áudio, permitindo a reprodução dos sons captados.

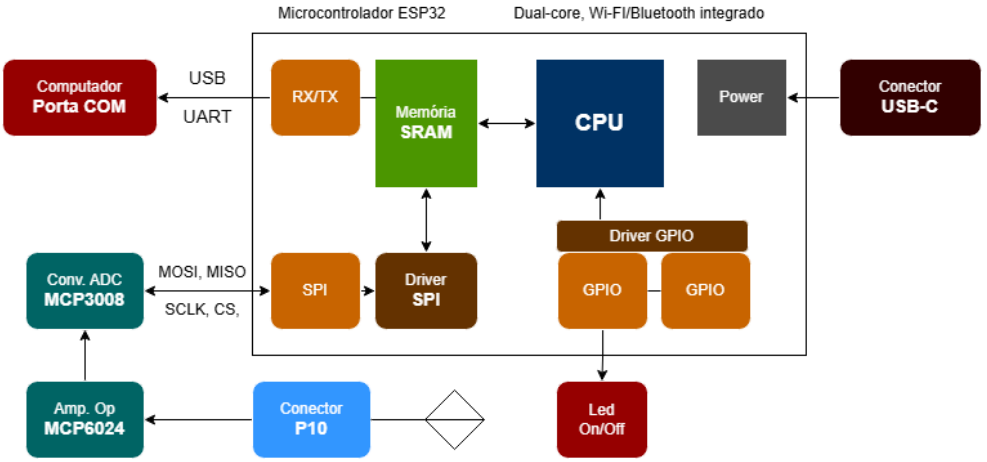
Essa abordagem possibilita a visualização e análise do sinal de áudio em tempo real, além de permitir a experimentação de efeitos sonoros a partir dos dados digitalizados, simulando a funcionalidade de uma pedaleira digital básica.

5.2 DIAGRAMA DO SISTEMA

O diagrama abaixo apresenta uma visão geral dos principais componentes que compõem o sistema de hardware desenvolvido para a pedaleira eletrônica. Ele detalha a interação entre o amplificador operacional MCP6024, responsável por amplificar o sinal analógico da guitarra, o conversor ADC MCP3008, que realiza a conversão analógico-digital do sinal amplificado, e a ESP32, encarregada de transmitir os dados digitalizados para o computador via comunicação serial.

O diagrama de blocos do sistema de hardware é uma ferramenta essencial para a documentação e compreensão do projeto, pois fornece uma visão clara e simplificada da interação entre os principais componentes. Ele facilita a identificação do fluxo de dados e dos sinais de controle, permitindo que engenheiros e desenvolvedores compreendam rapidamente como os elementos do sistema estão conectados e operam em conjunto. Além disso, o diagrama serve como um recurso valioso para a manutenção do projeto, pois torna mais fácil localizar possíveis falhas ou pontos de melhoria, além de agilizar a implementação de futuras modificações ou expansões. Em resumo, o diagrama contribui significativamente para a clareza do sistema, promovendo uma abordagem mais organizada e eficiente ao longo do ciclo de vida do projeto.

Figura 12 - Diagrama de Blocos do Sistema de Hardware



Fonte: Elaborado pelos Autores (2024)

6. RESULTADOS E DISCUSSÕES

Visando garantir o correto funcionamento dos componentes de forma isolada, utilizaram-se como instrumentos auxiliares o osciloscópio e uma pedaleira eletrônica de fabricação profissional. Desta forma, foi possível monitorar a voltagem produzida e amplificada, assim como a fidelidade do som manipulado para a posterior conversão para digital. O software Audacity foi escolhido para realizar a gravação e análise do áudio produzido.

6.1 AMPLIFICAÇÃO DE SINAL ANALÓGICO

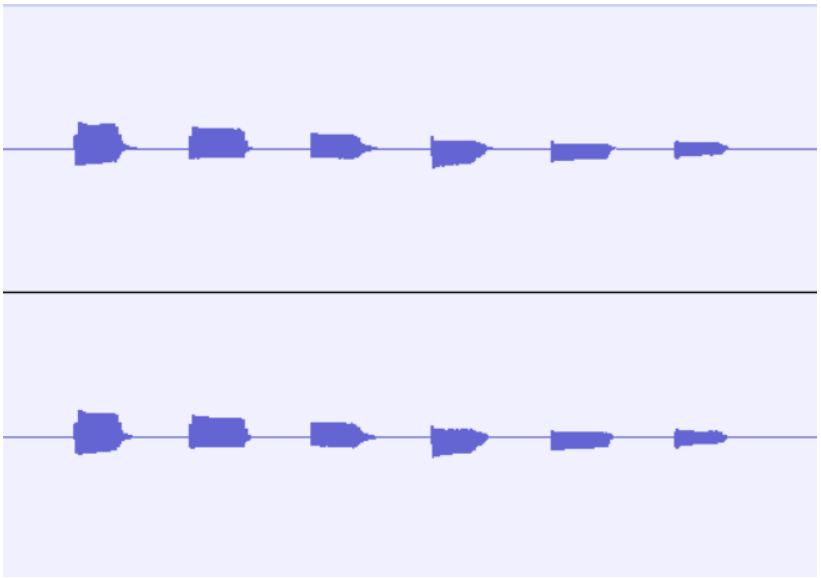
Em relação à amplificação do sinal analógico, dois resultados distintos foram observados. Na primeira tentativa, a amplificação não ocorreu conforme esperado devido a um defeito no amplificador operacional utilizado, comprometendo o desempenho do circuito. Após identificar o problema, o amplificador defeituoso foi substituído pelo MCP6024, um amplificador operacional de baixo ruído e consumo, adequado para aplicações com sinais de baixa tensão. Com a troca do componente, o circuito foi reconstruído, e foi possível capturar a seguinte onda de áudio.

Figura 13 - Captura do sinal analógico amplificado

Fonte: Elaborado pelos autores (2024)

Os testes foram realizados em laboratório na universidade, utilizando uma guitarra com captação ativa (modelo desconhecido). Para a amostragem, foram tocadas individualmente as cordas Mi, Lá, Ré, Sol, Si e Mi. As tensões medidas no circuito variaram entre 0 e 3V, conforme esperado para o sinal amplificado pelo MCP6024. Ao comparar os resultados com a amostragem obtida pela pedaleira profissional (Figura 14) isolada, nota-se uma diferença significativa na amplitude das ondas. Na prática, essa variação representa um grau de distorção não intencional no sinal amplificado pela pedaleira do projeto, possivelmente resultante das características do amplificador operacional utilizado.

Figura 14 - Captura do sinal digital da pedaleira profissional



Fonte: Elaborado pelos autores (2024)

Com a etapa de amplificação do sinal concluída e os resultados validados, as condições se mostram adequadas para iniciar o processo de conversão analógico-digital no projeto. O sinal amplificado está dentro do intervalo de tensão esperado (0 a 3V), o que garante uma leitura precisa pelos conversores AD planejados, como o MCP3008. A partir desta fase, será possível manipular o sinal digitalizado, dando sequência às etapas de processamento e controle necessários para o desenvolvimento completo da pedaleira.

6.2 CONVERSÃO ANALÓGICO DIGITAL

Durante o processo de conversão, foram realizadas leituras de diversas frequências e amplitudes do sinal, com o objetivo de analisar a qualidade e a precisão do ADC na captura do áudio. No entanto, o MCP3008, por especificação, não é capaz de ler tensões negativas, o que impõe uma limitação ao processo de digitalização. Como resultado, qualquer componente negativo do sinal de áudio, normalmente oscilatório, é ignorado pelo ADC, resultando em uma perda parcial das informações do sinal original.

Considerou-se a possibilidade de implementar um offset no sinal de entrada para deslocá-lo para uma faixa completamente positiva, permitindo a digitalização dos componentes negativos. No entanto, a introdução de um circuito de offset mostrou-se complexa para o escopo do projeto, exigindo componentes adicionais e ajustes que comprometeriam a simplicidade e a eficiência da pedaleira. Assim, optou-se por manter o sinal dentro da faixa de 0 a 3V, aceitando a perda do componente negativo.

Esse processo de conversão foi acompanhado de monitoramento contínuo no osciloscópio, onde foi possível observar as limitações e o comportamento do sinal digitalizado. A perda do componente negativo, embora impacte ligeiramente a fidelidade do áudio digitalizado, permitiu que o projeto avançasse sem adicionar complexidade excessiva ao circuito, mantendo o foco na funcionalidade básica da pedaleira.

6.3 MANIPULAÇÃO DE DADOS DIGITAIS

Após a conversão do sinal analógico para digital, a próxima etapa consistiu na manipulação dos dados digitalizados. Utilizando a ESP32 e um script em Python, foram estabelecidas as bases para o processamento em tempo real dos dados obtidos. A ESP32 enviava as amostras digitais via comunicação serial para o computador, onde o script em Python utilizava a biblioteca PyAudio para reproduzir o áudio de forma contínua.

A manipulação dos dados foi projetada para permitir uma análise do áudio em tempo real com o objetivo de validar a precisão do sinal digital. Durante os testes, ajustes foram feitos na taxa de amostragem e na formatação dos dados recebidos, garantindo uma reprodução fluida e sem interrupções. A configuração final estabeleceu uma taxa de amostragem de 8000 Hz, considerada adequada para a representação fiel dos sons captados pela guitarra.

Os testes revelaram uma resposta satisfatória na reprodução do áudio, com uma latência mínima, possibilitando a percepção de variações sonoras de forma quase instantânea. Essa etapa confirmou a viabilidade de utilizar a ESP32 e o MCP3008 para criar uma pedaleira digital capaz de capturar, processar e reproduzir sons de maneira eficiente, preparando o projeto para futuras implementações de efeitos mais complexos.

Cada corda da guitarra foi tocada individualmente para identificar as frequências e variações de amplitude associadas a ela. As amostras digitais resultantes foram processadas e exibidas em um gráfico de ondas (Figura 15), representando as capturas das seis cordas.

Figura 15 - Áudio capturado da pedaleira do projeto

Fonte: Elaborado pelos autores (2024)

Apesar de algum nível de ruído presente nos dados, os resultados mostraram-se satisfatórios para diferenciar os tons das cordas de forma clara. A análise revelou que, mesmo com a perda de componentes negativos do sinal (devido às limitações do ADC), as características sonoras específicas de cada corda foram preservadas. Isso indica que o sinal digitalizado é adequado para o propósito de distinção tonal, permitindo que cada corda seja identificada e processada de maneira distinta.

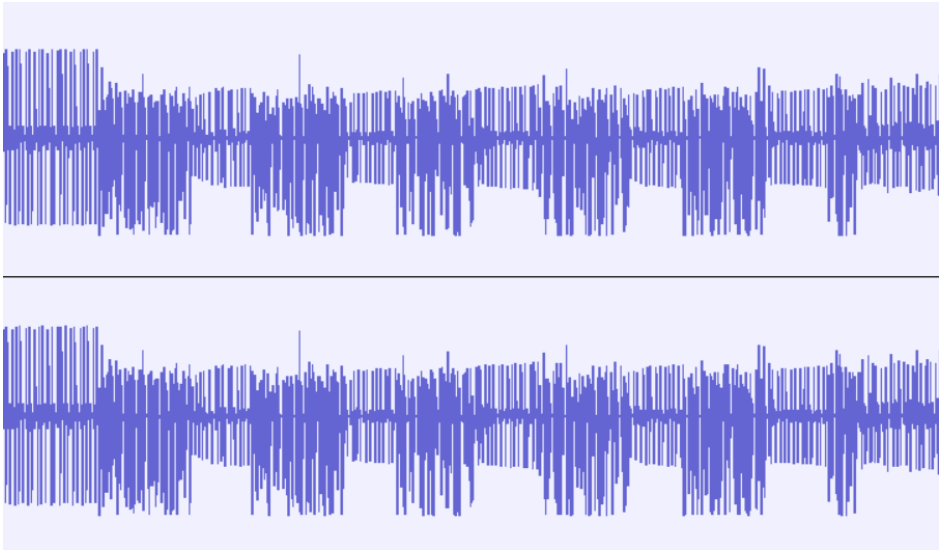
6.4 DIFICULDADES DO PROCESSO

Durante o desenvolvimento do projeto, uma das principais dificuldades encontradas foi a manipulação dos sinais elétricos. Estudantes da área de ciência da computação têm maior familiaridade com processamento de dados e programação, enquanto o ajuste preciso de sinais analógicos exige conhecimento específico em eletrônica, como amplificação, filtragem e estabilização de sinais. A falta de experiência com esses conceitos tornou o ajuste do circuito amplificador e a eliminação de ruídos desafiadores, exigindo várias tentativas até que o sinal apresentasse uma qualidade satisfatória para a conversão digital.

Outro desafio significativo surgiu na tentativa de equilibrar a redução de ruídos com a preservação do tom das cordas. Em alguns testes, a disposição específica dos componentes no circuito permitia a eliminação quase completa dos ruídos, mas comprometia a percepção dos tons das cordas, tornando-os inaudíveis. Ajustando a disposição para recuperar o tom, os ruídos voltavam a se manifestar no sinal digitalizado. Esse dilema demandou uma configuração

intermediária, que embora não eliminasse completamente os ruídos, mantinha a distinção tonal de cada corda de forma adequada.

Figura 16 - Áudio capturado da pedaleira do projeto com ruído



Fonte: Elaborado pelos autores (2024)

Para a reprodução do áudio digitalizado, optou-se por desenvolver um programa em Python, que realizasse a transmissão e reprodução em tempo real das amostras capturadas. Embora existissem diversas abordagens para o processamento de áudio, como softwares especializados ou plataformas de controle mais complexas, a implementação em Python com a biblioteca PyAudio mostrou-se uma solução simples e eficaz, atendendo aos requisitos do projeto sem a necessidade de configurações avançadas. Essa abordagem permitiu a reprodução contínua e clara dos sons, além de facilitar ajustes rápidos no código, contribuindo para o desenvolvimento ágil e funcional do sistema de captura e reprodução de áudio.

7. CONCLUSÃO

A construção da pedaleira eletrônica permitiu a integração prática de conceitos teóricos de sistemas digitais e arquitetura de computadores, aplicados ao tratamento de sinais de áudio. A implementação dos componentes, desde o circuito de amplificação até a conversão analógico-digital, exigiu uma série de escolhas tecnológicas e metodológicas que garantiram a fidelidade e a eficiência do projeto.

O estudo teórico e prático envolveu análise de protocolos de comunicação e tecnologias de conversão, além de ajustes contínuos para a adequação do circuito e dos algoritmos de reprodução sonora. O resultado final demonstrou que os objetivos foram atendidos, tanto em termos de desempenho quanto de aprendizado, possibilitando uma base sólida para futuras melhorias, como a implementação de efeitos de áudio e o aprimoramento na captura de frequências. Esse projeto contribuiu significativamente para o aprofundamento dos conhecimentos em sistemas digitais, circuitos eletrônicos e manipulação de áudio digital em tempo real.

REFERENCIAL BIBLIOGRÁFICO

TOCCI, Ronald; WIDMER, Neal; MOSS, Gregory. **Sistemas Digitais: Princípios e Aplicações**. 11. ed. São Paulo: Pearson Prentice Hall, 2011;

TANENBAUM, Andrew. **Organização Estruturada de Computadores**. 5 ed. São Paulo: Pearson, 2007;

TANENBAUM, Andrew. **Sistemas Operacionais: Projeto e Implementação**, 2 ed. Porto Alegre: Bookman, 2000;

CUNHA, Judson M.; **JUNIOR**, Danton C. F., **Arquitetura de Computadores**. Indaial: Uniasselvi, 2012. Disponível em: <https://www.uniasselvi.com.br/extranet/layout/request/trilha/materiais/livro/livro.php?codigo=10873>. Acesso em 12/09/2024;

BUCHANAN, William. **Computer Busses: Design and Application**. Palgrave Macmillan, 2000;

TORRES, Rodrigo C., **Processadores Digitais de Sinais (DSPs) e Suas Aplicações.** 2004. Disponível em <https://mesonpiold.cbpf.br/cat/pdsi/downloads/DSPs_E_Suas_Aplicacoes_Em_DSP.pdf>. Acesso em 19/08/2024;

LIMA, Manoel E., **Conversão Digital Analógico e Analógico Digital**. 2016. Disponível em: <https://www.cin.ufpe.br/~es238/arquivos/aulas/aula17_conversores_adda.pdf>. Acesso em 19/08/2024;

SMITH, Grant M. **Tipos de Conversores ADC.** 2024. Disponível em: <https://dewesoft.com/pt/blog/tipos-de-converrsores-ad> Acesso em 19/08/2024;

WALKER, Martin. **MIDI Interfaces for the PC**. 1997. Disponível em: <https://www.soundonsound.com/sound-advice/midi-interfaces-pc#top>. Acesso em 12/09/2024;

KUPHALDT, Tony R. **Lessons In Electric Circuits**. Vol 4. 2007. Disponível em: <allaboutcircuits.com/assets/pdf/digital.pdf>. Acesso em 12/09/2024;

BACKES, André. **Linguagem C: completa e descomplicada**. Rio de Janeiro: Elsevier, 2013;

PARET, Dominique. **I²C Bus: From Theory to Practice.** New York: Wiley-IEEE Press, 1997;

LUTZ, Mark. **Programação em Python**. São Paulo: Novatec Editora, 2013;

ZELLE, John. **Python: Como Programar**. São Paulo: Pearson, 2018;

SWEIGART, Al. **Automatize Tarefas Maçantes com Python**. São Paulo: Novatec Editora, 2016;

MAKIYAMA, Marcio. **Placa ESP32: O que é, para que serve e uso.** 2023. Disponível em: <https://victorvision.com.br/blog/placa-esp32/> . Acesso em:16/10/2024;

ROISENBERG, Leandro. **Entenda a UART: Guia Completo sobre Comunicação Serial e Aplicações em Eletrônica**. 2023. Disponível em: <https://blog.lri.com.br/entenda-a-uart-guia-completo-sobre-comunicacao-serial-e-aplicacoes-em-eletronica/>. Acesso em 17/10/2023;

R&S. **Compreender UART**. 2024. Disponível em <https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/essentials-test-equipment/digital-oscilloscopes/compreender-uart_254524.html>. Acesso em 12/09/2024;

Intel. FPGA: Noções básicas e primeiros passos. Disponível em: <https://www.intel.com.br/content/www/br/pt/support/programmable/support-resources/fpga-training/getting-started.html>. Acesso em 13/09/2024;

BROWN, Steven; ZVONKO, Vranezic. **Fundamentals of Digital Logic with VHDL Design with CD-ROM**. McGraw-Hill Education, 2008;

SCHNEIDER, Josh; **SMALLEY**, Ian. **What is a microprocessor?**. 2024. Disponível em: <https://www.ibm.com/think/topics/microprocessor>. Acesso em 13/09/2024;

VIVANO, Amy. **What is a driver?.** 2024. Disponível em: <https://learn.microsoft.com/pt-br/windows-hardware/drivers/gettingstarted/what-is-a-driver>. Acesso em 17/10/2023;

KUNDU, Manish; **KUMAR**, Abhijeet. **A Review on Low Power SPI Protocol**. 2014. Disponível em: <https://www.ijvdcs.org/uploads/134652IJVDCS1458-29.pdf>. Acesso em: 11/11/2024;

SACCO, Francesco. **Comunicação SPI - Parte 1**. 2014. Disponível em: <https://embarcados.com.br/spi-parte-1/>. Acesso em 11/11/2024;

KIM, Ji-seong. **What is SPI (Serial Peripheral Interface)?**. 2022. Disponível em: <https://velog.io/@lutein/SPISerial-Peripheral-Interface%EB%9E%80>. Acesso em: 11/11/2024;

STAPLES, M.; NIAZI, M. Systematic review of organizational motivations for adopting CMM-based SPI. Information and Software Technology, v. 50, n. 7-8, p. 605–620, jun. 2008.

ANEXO I

Código-fonte em C++ da ESP32

```
#include <SPI.h>

const int csPin = 5;
const int adcChannel = 0; // Canal do MCP3008 usado

void setup() {
  Serial.begin(115200); // Configuração da porta serial
  SPI.begin(18, 19, 23, csPin); // Clock: 18, MISO: 19, MOSI: 23, CS: 5
  pinMode(csPin, OUTPUT);
  digitalWrite(csPin, HIGH);
}

int readADC(int channel) {
  // Leitura do MCP3008
  digitalWrite(csPin, LOW);

  byte command = 0b00011000 | (channel & 0x07);
  SPI.transfer(command);
  int adcValue = (SPI.transfer(0x00) & 0x03) << 8;
  adcValue |= SPI.transfer(0x00);

  digitalWrite(csPin, HIGH);
  return adcValue;
}

void loop() {
  // Captura o valor do canal 0 do MCP3008
  int sample = readADC(adcChannel);
  // Escala para 15 bits (0 a 32736) para aproximar a faixa de 16 bits
  sample = sample << 5;

  byte highByte = (sample >> 8) & 0xFF; // Byte mais significativo
  byte lowByte = sample & 0xFF;         // Byte menos significativo

  // Envia ambos os bytes pela Serial
  Serial.println(sample);

  //Serial.write(highByte);
  //Serial.write(lowByte);
  delayMicroseconds(125); // Ajuste do intervalo de amostragem (~8kHz)
}
```

ANEXO II

Código-fonte em Python para reprodução dos sons

```
import serial
import pyaudio
import struct

# Configuração da porta Serial
serial_port = 'COM11'
baud_rate = 115200
ser = serial.Serial(serial_port, baud_rate)

# Configuração do áudio
p = pyaudio.PyAudio()
sample_rate = 8000       # Taxa de amostragem para reprodução
channels = 1             # Mono
format = pyaudio.paInt16

stream = p.open(format=format, channels=channels, rate=sample_rate,
output=True)

try:
    while True:
        # Verifica se há pelo menos 2 bytes disponíveis
        if ser.in_waiting >= 2:

            # Lê dois bytes para formar uma amostra de 16 bits
            data_bytes = ser.read(2)

            # Converte os bytes em um inteiro de 16 bits
            sample = struct.unpack('<h', data_bytes)[0]

            # Reproduzir o áudio captado
            stream.write(data_bytes)
except KeyboardInterrupt:
    pass
finally:
    stream.stop_stream()
    stream.close()
    p.terminate()
    ser.close()
```