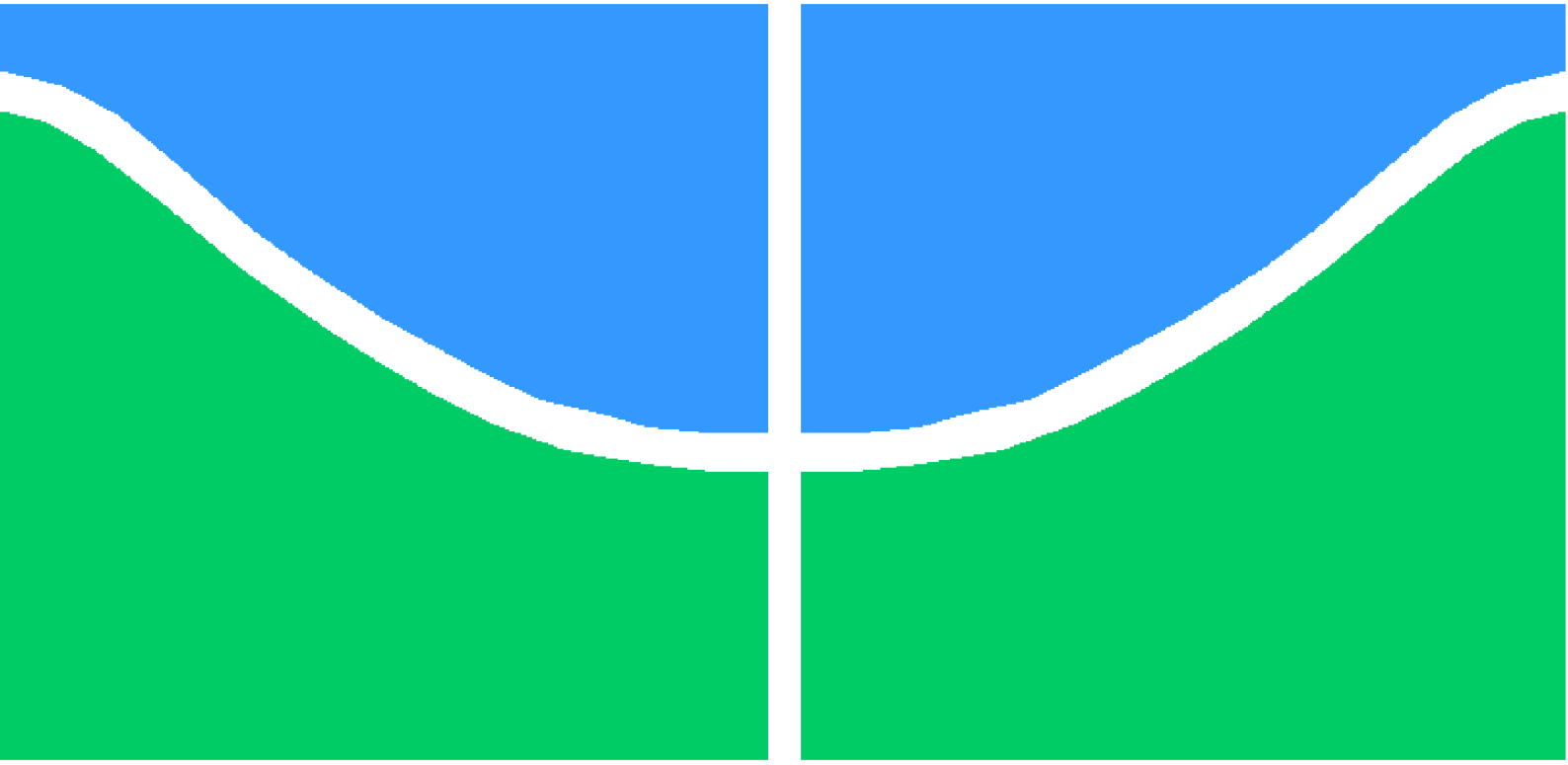

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletrônica

Desenvolvimento de técnicas não-lineares para processamento de áudio profissional

Autor: Vinicius Zschitschick de Oliveira
Orientador: Dr. Diogo Caetano Garcia

Brasília, DF
2018

Vinicius Zschitschick de Oliveira

Desenvolvimento de técnicas não-lineares para processamento de áudio profissional

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Diogo Caetano Garcia

Brasília, DF

2018

Vinicius Zschitschick de Oliveira
Desenvolvimento de técnicas não-lineares para processamento de áudio profissional/ Vinicius Zschitschick de Oliveira. – Brasília, DF, 2018-
79 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Diogo Caetano Garcia

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2018.

1. efeitos sonoros. 2. processamento de áudio. I. Dr. Diogo Caetano Garcia. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de técnicas não-lineares para processamento de áudio profissional

CDU 02:141:005.6

Resumo

Efeitos musicais são formas de manipular um som para adicionar uma determinada característica. Existem diversas técnicas para modificar o som e gerar efeitos que são amplamente utilizados para expressão artística de músicos, tais como efeitos não-lineares. Busca-se neste trabalho descobrir se funções com comportamento diferente de efeitos de distorção já conhecidos também podem gerar mudanças interessantes no timbre. Foram realizadas simulações em que se utilizavam funções trigonométricas como efeito não-lineares, sendo esse um estudo anterior ao projeto. Com base no resultado das simulações propõem-se um projeto de um equipamento capaz de aplicar curvas características de distorção editáveis. Foi observado que as curvas podem ter diferentes formas gerando diferentes timbres e sonoridades. Efeitos deste tipo podem ser executados digitalmente mas necessitam de filtros adequados para evitar a degradação da qualidade sonora.

Palavras-chaves: efeitos sonoros. processamento de áudio.

Abstract

Musical effects are ways to manipulate a sound to add a certain characteristic. There are several techniques for modifying sound and generating effects that are widely used for artistic expression of musicians, such as non-linear effects. The aim of this work is to find out if functions with different behavior of already known distortion effects can also generate interesting changes in the timbre. Simulations were performed using trigonometric functions as non-linear effects. Based on the results of the simulations, a design of an equipment capable of applying editable distortion characteristic curves is proposed. It was observed that the curves can have different forms generating different timbres and sonorities. Effects of this type can be performed digitally but need adequate filters to avoid degradation of sound quality.

Key-words: sound effects. audio processing.

Lista de ilustrações

Figura 1 – Produção de ondas sonoras. 13
Figura 2 – Representação da altura no campo musical. 14
Figura 3 – Composição de harmônicos da onda quadrada e dente de serra. 15
Figura 4 – Sinal analógico (a) e sinal digital (b) 16
Figura 5 – Diagrama de fluxo do filtro phaser 19
Figura 6 – Curva característica de um overdrive. 21
Figura 7 – Comparação entre efeitos não-lineares dos tipos corte suave e abrupto . 21
Figura 8 – Comparação de amplitude entre onda e sua versão distorcida. 22
Figura 9 – Espectro de saída devido ao efeito não-linear(a),espectro de saída com efeito aplicado após sobreamostragem(b) ,filtragem passa-baixas(c) e subamostragem (d) 23
Figura 10 – Curva característica com diversos valores para o ganho de entrada a . . 27
Figura 11 – Espectro após a aplicação do efeito 2 28
Figura 12 – Curva característica com diversos valores para o ganho de entrada a . . 29
Figura 13 – Espectro após aplicação do efeito 2 29
Figura 14 – Curva característica com diversos valores para o ganho de entrada a . . 30
Figura 15 – Espectro após a aplicação do efeito 3 com b=0 31
Figura 16 – Curva característica do efeito 4. 31
Figura 17 – Espectro após a aplicação do efeito 4 32
Figura 18 – Curva característica arbitrária. 33
Figura 19 – Diagrama geral de funcionamento 34
Figura 20 – Três pontos interpolados por uma spline cúbica. 34
Figura 21 – Comparação entre o filtro ideal e o real. 36
Figura 22 – Circuito condicionador 36
Figura 23 – Entrada e saída do circuito condicionador. 37
Figura 24 – Resposta em frequência do circuito condicionador. 37
Figura 25 – Filtro de ganho AC. 38
Figura 26 – Filtro de ganho DC. 39
Figura 27 – Filtro passa-baixas de terceira ordem. 40
Figura 28 – Amplificador inversor. 40
Figura 29 – Diagrama de rotina executada pelo microcontrolador. 42
Figura 30 – Modo de sinalização diferencial. 42
Figura 31 – Diagrama de envio para DAC 44
Figura 32 – Comunicação I^2S. 44
Figura 33 – Interface de usuário. 45
Figura 34 – Conexões da placa de circuito impresso. 46

Figura 35 – Protótipo construído. 47
Figura 36 – Curva característica do efeito 1 49
Figura 37 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 1 50
Figura 38 – Curva característica do efeito 2 51
Figura 39 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 2 51
Figura 40 – Curva característica do efeito 3 52
Figura 41 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 3 53
Figura 42 – Curva característica do efeito 4 54
Figura 43 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 4 54

Lista de tabelas

Tabela 1 – Duas oitavas musicais e suas frequências em Hertz 24

Lista de abreviaturas e siglas

SPL	Sound pressure level

Sumário

1 INTRODUÇÃO 11
1.1 Objetivos 12
1.1.1 Objetivos Gerais 12
1.1.2 Objetivos Específicos 12
1.2 Justificativa 12
1.3 Estrutura do Trabalho 12

2 REVISÃO BIBLIOGRÁFICA 13
2.1 Som 13
2.1.1 Altura 14
2.1.2 Timbre 14
2.1.3 Intensidade 16
2.2 Áudio Digital 16
2.2.1 Representação e Processamento de Áudio Digital 17
2.3 Efeitos Sonoros 18
2.3.1 Flanger 18
2.3.2 Chorus 18
2.3.3 Phaser 19
2.3.4 Tremolo 19
2.3.5 Efeitos não-Lineares 20
2.4 Aliasing 22
2.5 Distorção de intermodulação 24

3 SIMULAÇÃO EM SOFTWARE 26
3.1 Efeito 1 27
3.2 Efeito 2 28
3.3 Efeito 3 30
3.4 Efeito 4 31

4 METODOLOGIA 33
4.1 Manipulação da Curva Característica 34
4.2 Filtro de Aliasing 35
4.3 Circuito Condicionador 36
4.3.1 Ganho AC 37
4.3.2 Ganho DC 38
4.3.3 Filtro Passa-Baixa 39

4.3.4 Inversor de amplitude 40
4.4 Microcontrolador 41
4.4.1 Conversor AD 42
4.4.2 Timer 43
4.4.3 Comunicação Serial 43
4.5 Conversor Digital Analógico 43
4.6 Interface 44
4.7 Placa de Circuito Impresso 46

5 RESULTADOS 48
5.1 Efeito 1 48
5.2 Efeito 2 50
5.3 Efeito 3 52
5.4 Efeito 4 53

6 CONCLUSÕES 56

REFERÊNCIAS 57

APÊNDICES 59

APÊNDICE A – CÓDIGO DA INTERFACE 60

APÊNDICE B – CÓDIGO MICROCONTROLADOR 66

APÊNDICE C – CÓDIGO ESP8266 78

1 Introdução

Efeitos musicais são utilizados para manipular um som. Basicamente o som de um instrumento sofre alguma manipulação que adiciona uma característica ao som, como a adição de eco por exemplo. Estes efeitos são utilizados para expressão artística de músicos, permitindo que comuniquem suas ideias musicais.

Desde de início da guitarra elétrica, guitarristas vem adicionando distorção ao instrumento. A distorção surgiu inicialmente com o uso da amplificação maior do que suportado pelos amplificadores. (REISS; MCPHERSON, 2014)

O amplificador da Fender lançado em 1947 possuia a característica de introduzir distorção quando utilizada no volume máximo. O que poderia ser considerado um erro de engenharia introduziu um novo tipo de efeito utilizado até os dias atuais. Em 1962 a empresa Gibson lançou o pedal Fuzz-tone, que permitia obter o efeito fuzz sem a necessidade de usar o amplificador no volume máximo. Nos anos de 1960 e 1970 houve uma corrida entre fabricantes de amplificadores e pedais, abrindo uma enorme quantidade de possibilidades para músicos. (ANSELMI, 2017)

Atualmente existem diversas possibilidades de amplificadores e pedais. Há também a possibilidade de aplicar estes efeitos digitalmente. Para aplicar efeitos digitais são tomadas amostras do som, que são armazenadas em forma numérica, e com uso de diferentes técnicas é possível adicionar os mais diversos tipos desejados de efeitos. Ao longo do tempo diversas técnicas foram criadas e são amplamente utilizadas.

Um dos tipos de efeitos são os lineares, que são baseados em processos como atrasos, filtros e ganhos, sendo utilizados para modificar a amplitude e a fase do som introduzido no sistema. Exemplos desse tipo de efeito são chorus e eco.

O outro tipo de efeitos são os de distorção que são baseados em processamento não-linear. Este tipo de efeito adiciona componentes de frequência que não estão presentes no som original e são capazes de gerar uma grande variedade de sons. Dois dos efeitos mais conhecidos deste tipo são overdrive e fuzz.

Este trabalho entra no contexto de efeito não-lineares. Busca estudar como os efeitos clássicos funcionam e busca propor uma forma de abordagem digital para explorar de uma forma diferente os efeitos de distorção.

1.1 Objetivos

1.1.1 Objetivos Gerais

Este trabalho busca estudar se é possível utilizar processamento digital de sinais para obter novos efeitos sonoros baseados em curvas não-lineares para aplicação em guitarra. O objetivo é gerar um protótipo eletrônico que permite aplicar uma curva de distorção personalizada gerada digitalmente, em uma guitarra. Deseja-se assim obter uma forma diferente de ajuste das características de efeitos não-lineares.

1.1.2 Objetivos Específicos

- Criar uma interface de usuário em que uma distorção possa ser customizada.
- Em um microcontrolador aplicar a distorção e o processamento necessário para manter a qualidade sonora.
- Criar um circuito para adaptar a tensão vinda da guitarra para conversão analógico digital.

1.2 Justificativa

Efeitos de distorção foram inicialmente descobertos com amplificadores sendo forçados a amplificarem além de seu limite, ou seja possuem origem analógica. Efeitos digitais podem possuir curvas características mais flexíveis e estudar outras curvas é uma possibilidade de obter efeitos de distorção com sonoridades diferentes.

1.3 Estrutura do Trabalho

O texto foi dividido em cinco capítulos. O primeiro capítulo faz um estudo de conceitos importantes sobre som, áudio digital e efeitos famosos e faz um estudo sobre características que acompanham os efeitos não-lineares. O segundo fala das simulações realizadas. O terceiro capítulo fala da solução encontrada para realizar o protótipo. O quarto mostra os resultados obtidos e discute alguns pontos. O quinto capítulo é a conclusão.

2 Revisão Bibliográfica

Nesta seção são apresentados conceitos básicos de som analógico e digital e de efeitos sonoros amplamente utilizados no meio musical.

2.1 Som

Som é basicamente uma vibração que acontece no ar(SELF et al., 2009). É criado quando algo insere energia no meio e é então propagada da fonte por meio de diferenças de pressão que tem como equilíbrio a pressão atmosférica. (POHLMANN, 2011)

As diferenças de pressão podem ser produzidas periodicamente ou aperiodicamente. Um violino move o ar em uma taxa fixa, ou seja é periódico. Já o som de algo quebrando não possui taxa fixa, ou seja é aperiódico. Uma sequência de uma vibração periódica, de rarefação-compressão-rarefação determina um ciclo. O número de ciclos por segundo é a frequência da onda sonora e é medida em Hertz(Hz). A figura 1 representa a geração e propagação de uma onda sonora. (POHLMANN, 2011)

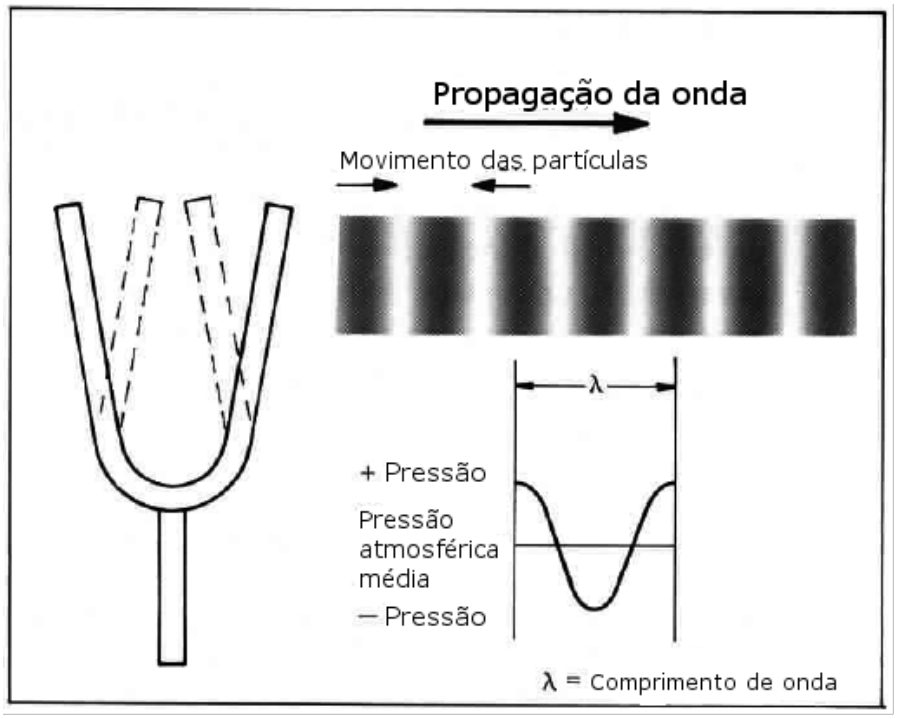

Figura 1 – Produção de ondas sonoras. Fonte: Adaptado de GRAHAM (1978)

As três qualidades do som são timbre, altura e intensidade. (GRAHAM, 1978)

2.1.1 Altura

Altura é a qualidade do som que faz com que sons possam ser considerados mais altos ou baixos. É depende da frequência sonora, quanto maior a frequência maior a altura. (GRAHAM, 1978)

No campo musical não se utiliza frequências para representar a altura, mas possuem representações númericas, ou de nomes. Na figura 2 representações comuns de diferentes alturas.

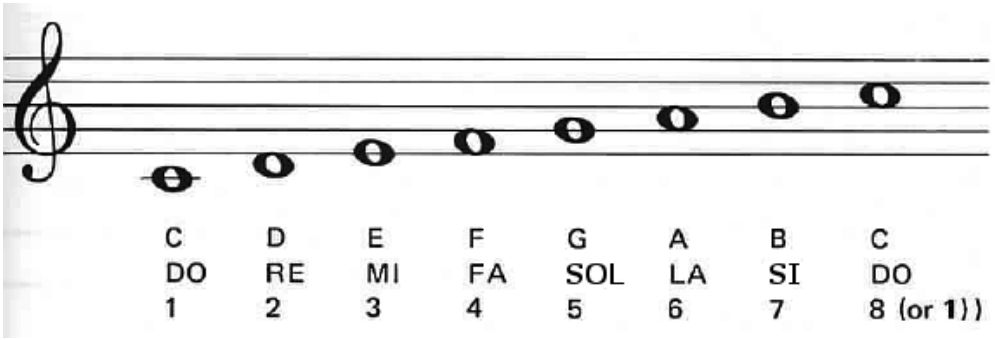

Figura 2 – Representação da altura no campo musical. Fonte: Adaptado de GRAHAM (1978)

O intervalo de frequências que podem ser ouvidas varia de pessoa para pessoa. Em média o intervalo vai de 20 Hz até 20 kHz. O limite máximo cai com a idade, uma pessoa entre 40 e 50 anos pode escutar algo em torno de 15 kHz. (GRAHAM, 1978)

2.1.2 Timbre

Timbre é a qualidade do som que permite diferenciar dois instrumentos diferentes que tocam uma nota com a mesma frequência(GRAHAM, 1978).

A figura 3 mostra um exemplo de como a diferença entre amplitude e deslocamento entre harmônicos muda a forma de onda. Em (a) uma onda dente de serra e em (b) uma onda quadrada. Se a frequência fundamental das duas onda for igual, elas terão a mesma frequência, mas timbres diferentes.

Formas de onda periódicas podem ser representados pela sua frequência fundamental e frequências múltiplas da fundamental. Por exemplo uma formada de um onda com frequência fundamental de 150 Hz terá sons harmônicos em 300,450,600 Hz e assim por diante. As amplitudes e fases relativas entre esses harmônicos são responsáveis pelo timbre da onda. (POHLMANN, 2011)

A estrutura harmônica de ondas periódicas pode ser resumida pelo teorema de Fourier. Ele diz que ondas complexas podem ser decompostas em uma soma de on-

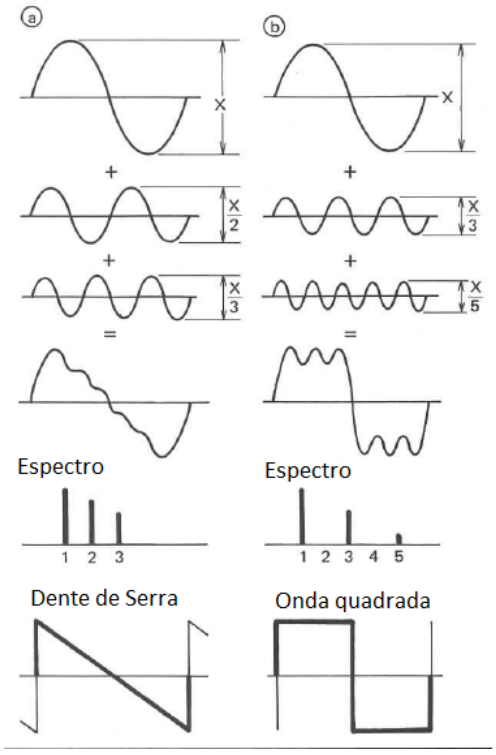

Figura 3 – Composição de harmônicos da onda quadrada e dente de serra. Fonte: Adaptado de GRAHAM (1978)

das senoidais. Essa decomposição pode ser utilizada para estudar a natureza da onda. (POHLMANN, 2011)

A análise harmônica com a série de Fourier permite separar uma onda periódica arbitrária em termos simples senoidais. Pode-se obter a série de Fourier tomando $f_1(x) = \cos(x)$ e $f_2(x) = \sin(x)$. A série de Fourier de uma função $f(x)$ é dada por (WEISSTEN)

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx) \tag{2.1}$$

onde

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)dx \tag{2.2}$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx \tag{2.3}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx \tag{2.4}$$

e n =1, 2, 3 ... .

Cada onda possui diferentes coeficientes a_0, a_n, b_n, gerando diferentes timbres. A figura 3 como a soma de coeficientes da série de Fourier aproxima diferentes formas de onda, em função da amplitude e fase dos harmônicos.

2.1.3 Intensidade

Quando um corda vibra, o tamanho da corda e a quantidade de movimento diz quanta energia é transferida para o meio. Quanto maior a energia da corda mais é alto será o som produzido. A energia no som é chamada intensidade sonora e é medida em watts. Uma escala relativa logarítmica é usada é medida em dB ou decibéis. (RUSS, 2012) O nível de pressão sonora (SPL, do inglês sound pressure level) é medido numa escala logarítmica com a seguinte equação. (POHLMANN, 2011)

$$\text{Nível de Intensidade} = 10 \log \left(\frac{P_1}{P_2} \right) dB \tag{2.5}$$

P_2 é substituído pela intensidade do limite da audição de $10^{-12} W/m^2$. P_1 é a intensidade medida. Por exemplo, a intensidade de um som medido é de $10 W/(m^2)$, dando uma intensidade de 130dB SPL. (POHLMANN, 2011)

2.2 Áudio Digital

O som pode ser captado por um transdutor, como um microfone, e ser transformado em um sinal de áudio analógico. Para transformar o sinal de áudio analógico em áudio digital é utilizado um conversor analógico-digital que captura amostras do sinal analógico periodicamente, formando um aproximação digital do sinal. (POHLMANN, 2011) Existem três características principais em todos os áudios digitais: taxa de amostragem,

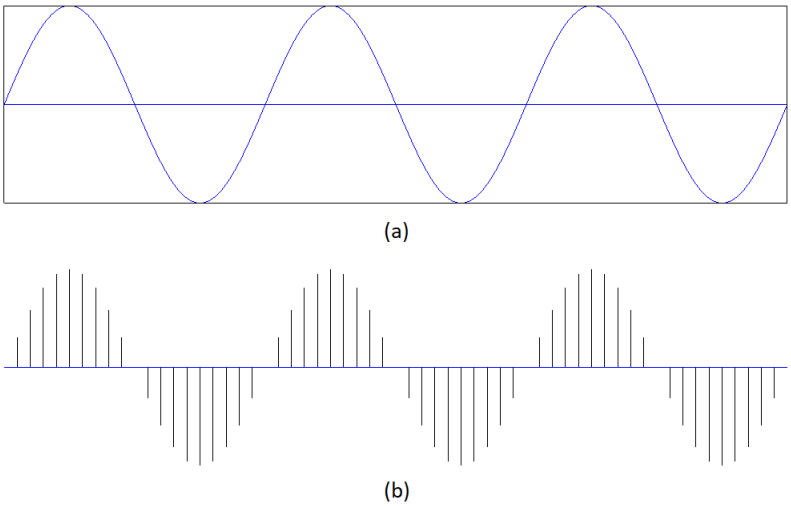

Figura 4 – Sinal analógico (a) e sinal digital (b)

resolução e número de canais. (REISS; MCPHERSON, 2014)

Taxa de amostragem é taxa com qual amostras são obtidas. Medida em Hertz, em que cada ciclo representa uma amostra. Uma taxa de 44,1 kHz é utilizada em CDs de

áudio, e 96 ou 192 kHz em áudio de alta resolução. (REISS; MCPHERSON, 2014) Uma amostragem de 40kHz teoricamente funciona toda a faixa audível, mas taxas maiores são utilizadas para uma margem em caso de ruído adicional ou artefados indroduzidos pela amostragem e processamento. (SELF et al., 2009)

O teorema de amostragem determina a taxa de amostragem que se deve utilizar em função da banda de frequências do sinal. Se um sinal ser amostrado com frequência SHz, então sua frequência não deve exceder a frequência de Nyquist $S/2$ Hz. A taxa de amostragem deve ser, no mínimo, o dobro da frequência máxima do sinal para uma amostragem sem perdas. (POHLMANN, 2011)

A resolução trata da quantidade de bits utilizada para representar cada amostra. Valores comuns são 16 e 24 bits. A quantidade de números utilizados para representar a amostra é $2^{resolução} - 1$. (REISS; MCPHERSON, 2014)

O número de canais trata da quantidade de canais que formam o áudio, cada um representando um sinal. No som estéreo há mais de um canal, onde cada um é utilizado em diferentes caixas de som. No som monaural é utilizado apenas um canal. (REISS; MCPHERSON, 2014)

2.2.1 Representação e Processamento de Áudio Digital

Áudio digital é basicamente uma sequência de números amostrados em uma determinada frequência de amostragem, com valor inteiros de 0 a $2^{\text{Resolução}} - 1$ e posteriormente normalizados. Cada valor representa uma posição possível no alto-falante. (FARNELL, 2010)

O tempo entre as amostras pode ser dado como T_s e a frequência como $f_s = 1/T_s$.O sinal digital pode ser então representado como x(0),x(T_s),x($2T_s$),... . Se considerarmos um sinal finito pelo número N de amostras então x[0], x[1],...,x[N-1]. (REISS; MCPHERSON, 2014)

Em processamento digital de sinais as operações não são feitas com números inteiros, mas sim em ponto flutuante com o intervalo de -1 a 1.Esta é a forma como os sons são armazenados digitalmente. A resolução dos valores neste intervalo depende da resolução. O valor 1 representa o cone do alto-falante empurrado para fora o máximo possível, e -1 representa o cone se movendo para dentro tanto quanto possível. O valor zero representa a posição neutra. (FARNELL, 2010)

O processamento do áudio pode ser feito com uma equação de diferenças, uma formula que calcula a amostra de saída n, baseado na amostra n e anteriores. Um filtro

linear pode ser dado com uma equação de diferenças:(REISS; MCPHERSON, 2014)

$$y[n] = b_0x[n] + b_1x[n-1] + ...b_Nx[n-N] - a_1y[n-1] - a_My[n-M] \tag{2.6}$$

onde x é um sinal de entrada e y o sinal de saída. As constantes $b_0, ..., b_N$ e $a_0, ..., a_N$ são os coeficientes. (REISS; MCPHERSON, 2014)

2.3 Efeitos Sonoros

Efeitos sonoros são caixas ou ferramentas de software com sinais de áudio de entrada que são modificados de acordo com alguns parâmetros de controle e entrega sinais de saída. Modificar a característica sonora do sinal de entrada é o principal objetivo dos efeitos sonoros. As configurações dos parâmetros de controle são comumente feitas por engenheiros de som, músicos ou pelo próprio ouvinte. (ZÖLZER, 2011)

2.3.1 Flanger

Desenvolvido inicialmente utilizando fitas analógicas. Para criar este efeito eram utilizadas duas fitas tocando a mesma fita ao mesmo tempo. A saída das duas fitas é então combinada igualmente. (REISS; MCPHERSON, 2014)

O efeito acontece quando uma das fitas é atrasada. Posteriormente a outra fita é também atrasada para diminuir o atraso entre as duas faixas .O processo é repetido periodicamente. O atraso não pode exceder uma certo limite para que não não ocorra a percepção de eco. O efeito básico pode ser obtido digitalmente com a seguinte equação:(REISS; MCPHERSON, 2014)

$$y[n] = x[n] + gx[n - M[n]] \tag{2.7}$$

O atraso M[n] varia com o tempo sob o controle de um oscilador de baixa frequência.

Este efeito é geralmente utilizado para obter um som de avião a jato. Exemplos de músicas que utilizam este efeito são a Crazy Horse de Zakk Wylde e Barracuda da banda Heart. O efeito é utilizado na guitarra principal das duas músicas.

2.3.2 Chorus

Efeito obtido dobrando um e somando a versão copiada fora de tempo e de tom com o original Isso faz com que se obtenha o som de que dois instrumentos estão sendo tocados ao mesmo tempo. O tempo de atraso e variação de tom pode ser controlada pelo usuário de forma obter diferente sonoridades. (REISS; MCPHERSON, 2014)

A equação que básica tem a seguinte forma:(REISS; MCPHERSON, 2014)

$$y[n] = x[n] + gx[n - M[n]] \tag{2.8}$$

A principal diferença entre o chorus e flanger é tempo de atraso, o atraso do chorus está comumente entre 20 e 30 ms, o atraso do flanger entre 1 e 10 ms. (REISS; MCPHERSON, 2014)

Músicas que utilizam este são Every breath you take da banda The Police e Come as you are da banda Nirvana.

2.3.3 Phaser

É um efeito que cria uma série de cortes no espectro de áudio, onde determinadas frequências do som são atenuadas. Utiliza filtros passa tudo, que mantêm a magnitude do sinal de entrada mas introduz uma fase dependente da frequência. O nível do sinal filtrado pode ser ajustado com um ganho. O sinal filtrado é então somado com o original e baseado no princípio de interferência gera os corte. (REISS; MCPHERSON, 2014)

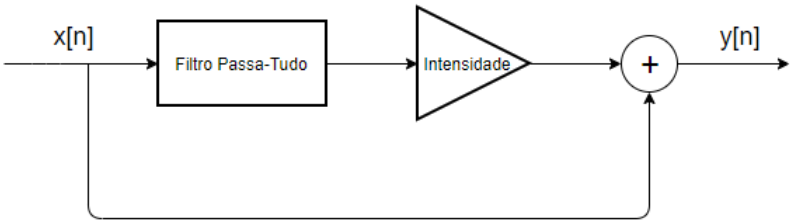

Figura 5 – Diagrama de fluxo do filtro phaser

A intensidade da atenuação pode ser ajustada .

2.3.4 Tremolo

Se refere a um estilo de tocar que envolve notas rápidas repetidas. E efeito de áudio simula este estilo de tocar modulando o sinal de entrada de forma que a nota na saída soe como uma série de notas curtas e rápidas. (REISS; MCPHERSON, 2014)

$$y[n] = m[n]x[n] \tag{2.9}$$

onde m é gerado por um oscilador de baixa frequência , com frequências tipicas entre 0.5 e 20Hz. (REISS; MCPHERSON, 2014)

Exemplos de músicas que utilizam este efeito são How soon is now da banda The Smiths e Gotta Be Somebody da banda Nickelbeck.

2.3.5 Efeitos não-Lineares

Os efeitos não-lineares , conhecidos como efeitos de distorção ,são utilizados desde os primórdios da guitarra elétrica. Estes efeitos são capazes de criar uma variedade de sons desde sons com características mais suaves até sons "sujos". Os efeitos mais conhecidos são Overdrive, Distort e Fuzz, que se baseiam no mesmo princípio mas com diferentes níveis de distorção. (REISS; MCPHERSON, 2014)

A distorção pode ser adicionada adicionada em um situação em que um amplificador é forçado a funcionar fora de seu limite, causando um corte a partir de certa amplitude de som.

É um efeito baseado no princípio de não-linearidade, onde sinais de níveis baixos são quase lineares e são passam a ser cada vez menos lineares conforme o nível do sinal aumenta. (REISS; MCPHERSON, 2014)

Efeitos de distorção como o overdrive podem ser descritos por uma curva característica que relaciona uma amostra de saída $y[n]$ a uma amostra de entrada $x[n]$.A seguinte equação é classificada como um overdrive pois possui uma região linear com uma graduação para uma região não-linear conforme cresce a amplitude :(REISS; MCPHERSON, 2014)

$$f(x) = \begin{cases} 2x & \text{if } 0 \leq |x| < 1/3 \\ 1 - (2 - 3x)^2/3 & \text{if } 1/3 \leq |x| < 2/3 \\ 1 & \text{if } 2/3 \leq |x| \leq 1 \end{cases}$$

A região não-linear desta equação define o som especifico deste efeito. A curva característica deste efeito está representado na figura 6. A curva característica de um efeito não-linear define um efeito sem memória, onde a amostra de saída $y[n]$ só depende da amostra atual $x[n]$ e de nenhuma amostra passada, de saída ou entrada. (REISS; MCPHERSON, 2014)

Cada um destes efeitos possuem uma região de corte, onde a partir de certo limite há o corte do sinal. A transição para essa região de distorção pode ser suave ou abrupta, dando diferente características sonoras. O corte suave é geralmente associado à amplificadores valvulados e o corte abrupto à amplificadores baseados em transistores.

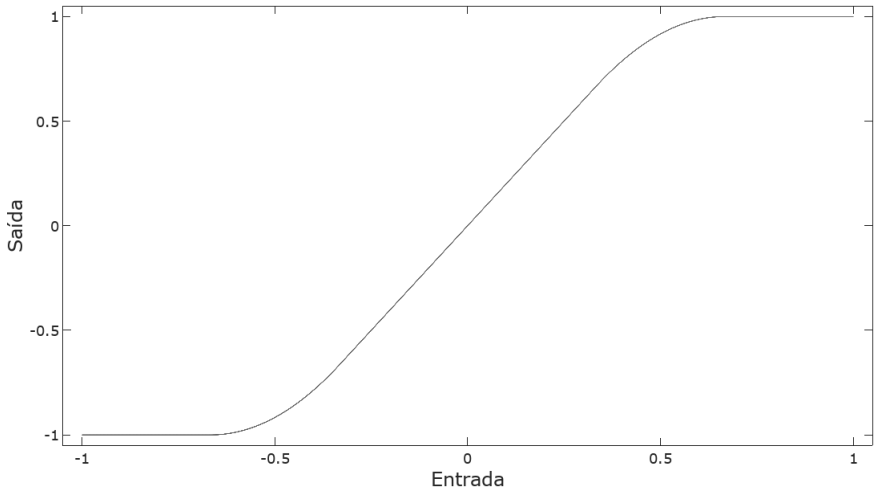

Figura 6 – Curva característica de um overdrive.

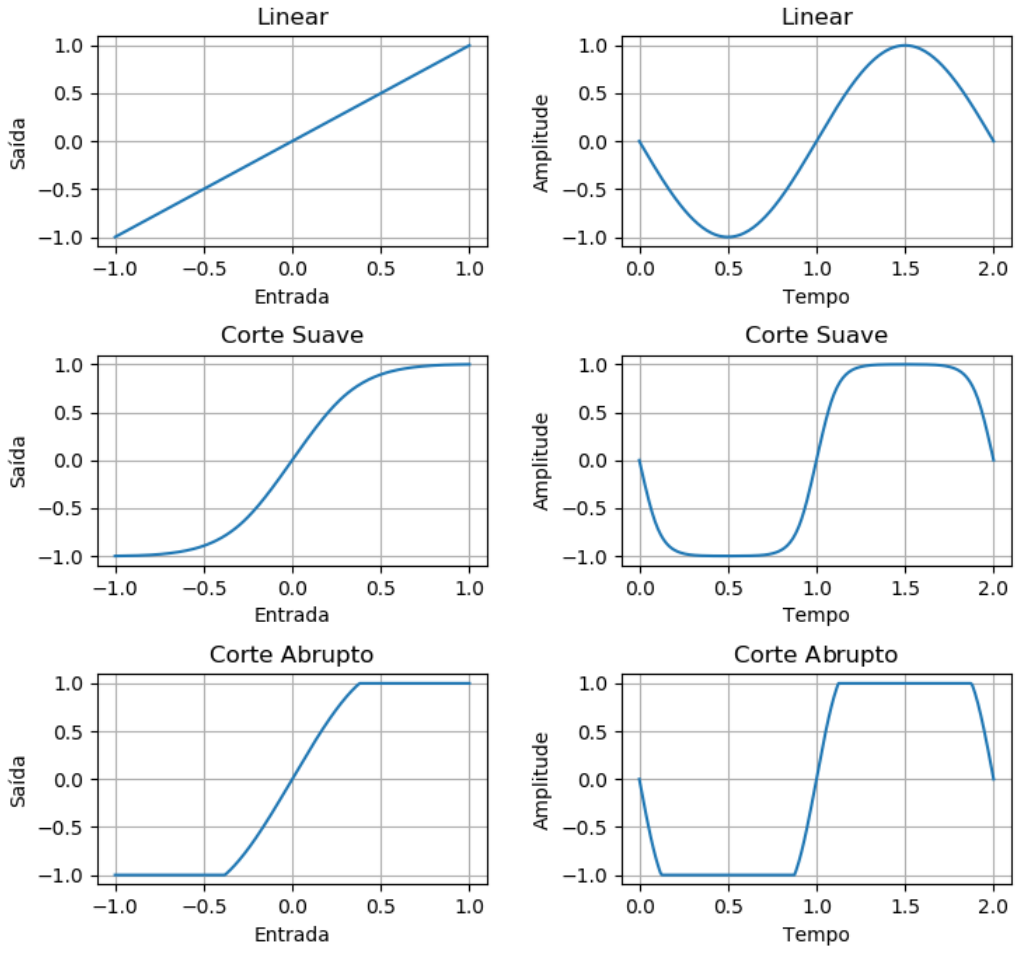

Figura 7 – Comparação entre efeitos não-lineares dos tipos corte suave e abrupto

Uma importante característica deste tipo de efeito é o de dependência da amplitude. No caso da guitarra a força que é aplicada na corda e o volume influenciam na sonoridade obtida. Uma comparação entre uma onda e sua versão distorcida com tipo de corte suave pode ser visto na seguinte figura:

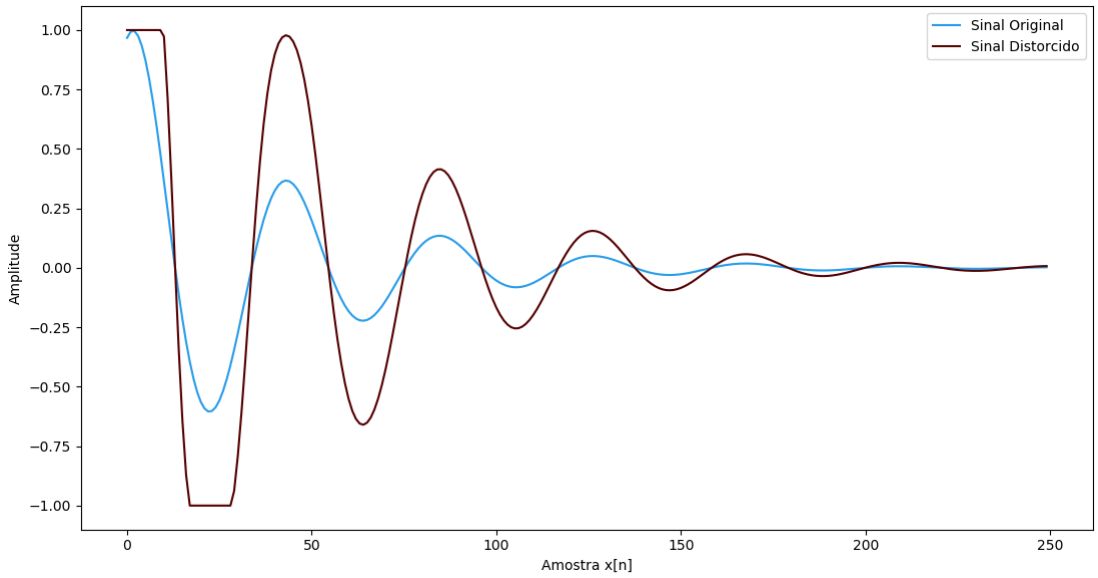

Figura 8 – Comparação de amplitude entre onda e sua versão distorcida.

Um exemplo de música que utiliza o overdrive é a Anesthesia(Pulling Teeth) da banda Metallica. Um exemplo de Fuzz é a música Seer da banda Witch.

2.4 Aliasing

Funções não-lineares criam um número infinito de harmônicos. No domínio digital isto gera problemas com relação ao aliasing. Harmônicos com frequência maior de que a de Nyquist aparecem na saída em componentes de menor frequência. Estas componentes de frequência não são mais relacionados harmonicamente com o som original, gerando perdas na qualidade de áudio. (REISS; MCPHERSON, 2014)

O aliasing pode ser removido utilizando a técnica de sobre amostragem. Pode-se aumentar a taxa de amostragem por um fator L inserindo $L-1$ zeros entre cada amostra. (REISS; MCPHERSON, 2014)

Representação matemática do sinal $w(m)$ sobre amostrado: (ZÖLZER, 2008)

$$w(m) = \begin{cases} x(\frac{m}{L}), & m = 0, \pm L, \pm 2L, ..., \\ 0, & \text{caso contrário} \end{cases} \tag{2.10}$$

Como a nova frequência e amostragem $f'_s = Lf_s$. (ZÖLZER, 2008)

Há o surgimento de uma imagem espectral, que é eliminada com um filtro anti-imagem h(m), de forma que a transformada de Fourier do sinal de saída seja: (ZÖLZER, 2008)

$$Y(e^{j\Omega'}) = H(e^{j\Omega'})X(e^{j\Omega' L}) \tag{2.11}$$

Onde $\Omega = \omega T$ e $\Omega' = \Omega / L$.

Com o processo de upsampling completo é possível aplicar efeito não-linear. O efeito ainda irá gerar infinitos harmônicos, mas uma faixa maior de frequência irá se ajustar dentro da nova taxa de Nyquist. A amplitude das componentes diminui conforme cresce a frequência, então as componentes que aparecem como aliasing são de amplitude desprezível. (REISS; MCPHERSON, 2014)

A processo de diminuir a taxa de amostragem é chamado de downsampling. Para isso é primeiro limitar a banda do sinal para π/M para evitar aliasing. Para isso o sinal pode ser filtrado com o $H(e^{j\Omega})$ da seguinte forma: (ZÖLZER, 2008)

$$W(e^{j\Omega}) = X(e^{j\Omega})H(e^{j\Omega}) \tag{2.12}$$

$$H(e^{j\Omega}) = \begin{cases} 1 & |\Omega| \leq \pi/M, \\ 0, & \text{caso contrário.} \end{cases} \tag{2.13}$$

No final do processo são tomadas amostradas a cada M amostras, que leva ao sinal de saída:(ZÖLZER, 2008)

$$y[n] = w(Mn) \tag{2.14}$$

O processo pode ser visualizado na seguinte figura:

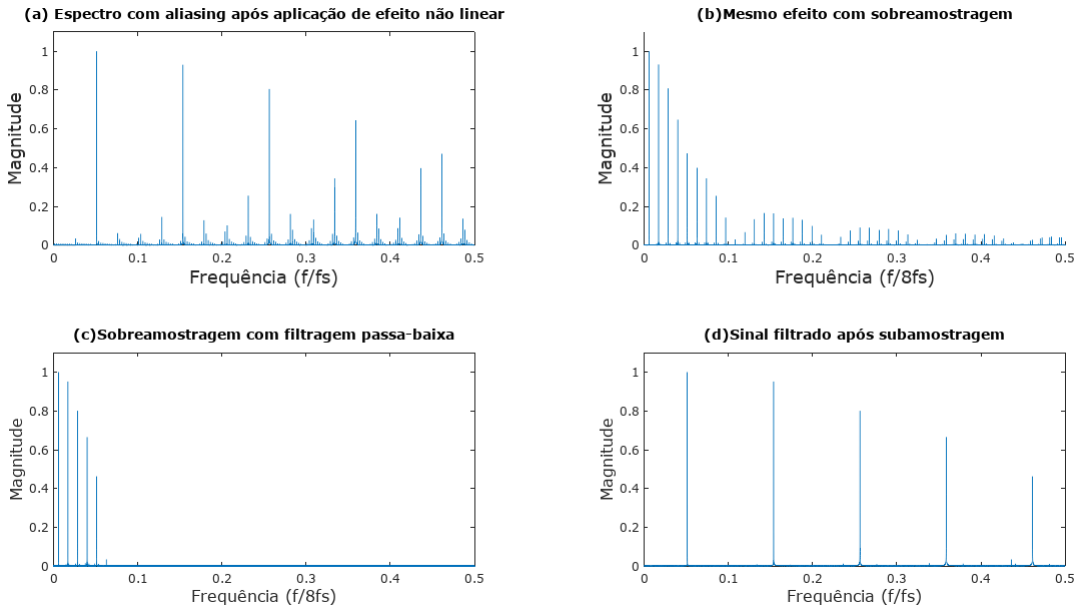

Figura 9 – Espectro de saída devido ao efeito não-linear(a),espectro de saída com efeito aplicado após sobreamostragem(b) ,filtragem passa-baixas(c) e subamostragem (d). Fonte: Adaptado de REISS; MCPHERSON (2014)

2.5 Distorção de intermodulação

Uma característica de funções de transferência não-lineares é a geração de distorção de intermodulação.Geralmente este não é um resultado desejado no ramo musical. (REISS; MCPHERSON, 2014)

Se tomarmos duas componentes de entrada com frequência f_1 e f_2 e aplicarmos um efeito não-linear $f(x(t)) = x^2$ (REISS; MCPHERSON, 2014)

$$x(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t) \tag{2.15}$$

$$f(x(t)) = \sin^2(2\pi f_1 t) + 2\sin(2\pi f_1 t)\sin(2\pi f_2 t) + \sin^2(2\pi f_2 t) \tag{2.16}$$

O termo que gera a distorção é o produto de duas frequências.Por identidade trigonométrica:

$$2\sin(2\pi f_1 t)\sin(2\pi f_2 t) = \cos(2\pi(f_1 - f_2)t) - \cos(2\pi(f_1 + f_2)t) \tag{2.17}$$

Então a saída irá apresentar termos com soma e subtrações entre as duas frequências de entrada. Para que essas frequências geradas sejam harmonicamente relacionadas à entrada f_1 e f_2 devem ser múltiplos. Essas frequências podem ser discordantes com o som original e gerar um efeito desagradável. (REISS; MCPHERSON, 2014)

Devido à essa característica dos efeitos não-lineares este tipo de feito é geralmente utilizado com notas simples ou "power chords"(combinações de fundamental e quintas justas). (REISS; MCPHERSON, 2014) Seres humanos são sensíveis à mudança de frequência nas notas, por exemplo numa nota de $880Hz$ é possível notar diferença a partir de $2,5Hz$. Devido à esta sensibilidade este efeito essa distorção é geralmente indesejada. Como exemplo pode-se observar a seguinte tabela: (RUSS, 2012)

Dó	Dó#	Ré	Ré#	Mi	Fá	Sol	Sol#	Lá	Lá #	Si
130,81	138,59	146,83	155,56	164,81	174,61	195,99	207,65	220	233,08	246,94
261,66	277,18	293,66	311,12	329,62	349,23	391,99	415,3	440	466,16	493,88

Tabela 1 – Duas oitavas musicais e suas frequências em Hertz

Nesta tabela há duas oitavas e a frequência em Hertz de cada nota contida. Se escolhermos f_1 como $391,99Hz$ e f_2 como $261,66Hz$, $f_1 - f_2 = 130,81Hz$ que é a nota Dó contida na tabela que é um múltiplo de $261,66Hz$ e $f_1 + f_2 = 653,32Hz$ que resulta numa frequência próxima à da nota Mi. Dessa forma é possível observar que combinação de fundamentais com quintas justas minimizam a distorção de intermodulação.

Não é possível utilizar efeitos não-lineares sem distorção de intermodulação harmônica, desta forma é necessário que cada tipo de efeito seja utilizado na aplicação musical apropriada. (REISS; MCPHERSON, 2014)

3 Simulação em Software

Foi realizado um estudo de efeitos não-lineares para verificar as possibilidades sonoras de curvas características diferentes das de corte suave/abrupto. Para realizar o estudo foi observado como funções trigonométricas e hiperbólicas modificam a sonoridade do som da guitarra. Foram escolhidas essas funções por possuírem diversos tipos de curvatura que podem ser acessadas rapidamente em softwares de computação numérica.

A linguagem de programação utilizada para as simulações foi a GNU Octave. Nesse software é possível realizar manipulação de arquivos de áudio e escutar o resultado. É possível também gerar gráficos e analise espectral com auxílio de transformada rápida de Fourier.

Para analisar o comportamento das funções como função de transferência foi utilizado seguinte equação:

$$y[n] = f(ax[n] + b)c \tag{3.1}$$

onde f são as funções trigonométricas e hiperbólicas,$x[n]$ é o sinal sonoro de entrada ,a é o ganho de entrada e b uma constante que é somada ao valor de entrada e c é um fator multiplicativo que é responsável por normalizar o sinal na faixa entre -1 e 1.

Os valores a e b são ajustados inicialmente para observar como a função se comporta conforme estes termos são variados, se for encontrada uma região da função que funcione como um efeito o valor c é então ajustado para garantir que a função permaneça no intervalo desejado.

Algumas características são esperadas de cada efeito obtido:

- Resultados no intervalo entre -1 e 1 pois é a uma forma comum de armazenar áudio digital.
- Apenas números reais de saída,de forma que apenas a parte real de cada função é utilizada.
- Efeito sonoro interessante e que não transforme o sinal de entrada em ruído.

Foram testadas as seguinte funções:

Trigonométricas seno,cosseno,tangente,arco seno,arco cosseno,arco tangente e suas respectivas funções inversas.

Hiperbólicas seno,cosseno,tangente,arco seno,arco cosseno,arco tangente e suas respectivas funções inversas..

O programa principal lê um arquivo de áudio de guitarra previamente gravado e aplica o efeito e reproduz o arquivo de som original e o com efeito. Dessa forma é possível comparar o som com e sem efeito e analisar como a função influencia a sonoridade.

Para filtrar o aliasing gerado pelas funções foram utilizadas duas funções disponíveis no Octave,as funções interp e decimate.A função interp que realiza a sobre amostragem e interpola os zeros inseridos .A função decimate realiza a sub amostragem .

3.1 Efeito 1

Definido pela equação :

$$y[n] = \frac{\mathbb{R}(\arcsin(x[n]\pi a + b))}{\pi/2} \tag{3.2}$$

Onde a pode ter um valor de 0.6 a 3.5, e b de -0.7 a -1.A função é dividida por $\pi/2$ pois este é o valor máximo de saída da função arco seno, de forma a ter a saída limitada entre -1 e 1.O domínio real da função arco seno é de -1 a 1. Após esse intervalo se utiliza apenas a parte real do resultado.

Na imagem 10 é possível observar a curva característica deste efeito , para diferentes valores de a e b constante em 0.8:

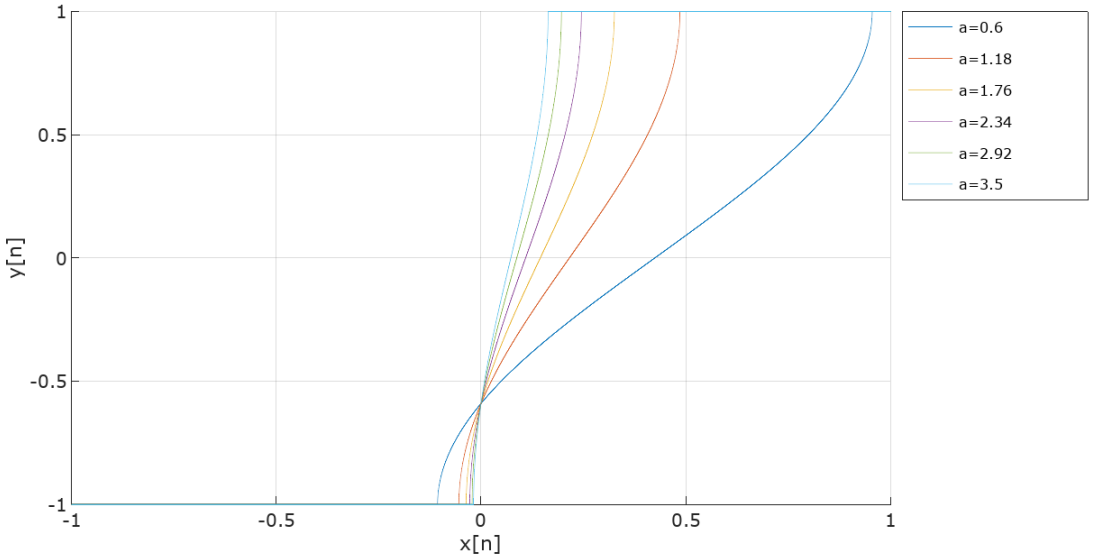

Figura 10 – Curva característica com diversos valores para o ganho de entrada a

O filtro apresenta basicamente um comportamento de corte abrupto, com um intervalo entre limites de corte ajustável. Este tipo de filtro faz que o sinal apresente cantos agudos. (REISS; MCPHERSON, 2014)

Na figura 11 é possível observar o espectro após aplicação do efeito 2.

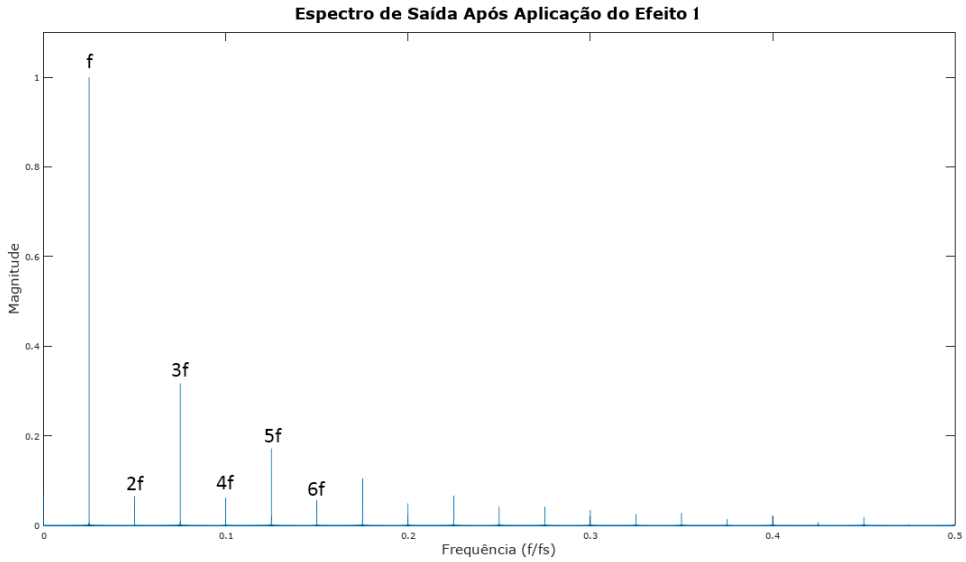

Figura 11 – Espectro após a aplicação do efeito 2

Funções assimetricas geram harmônicos pares e ímpares. Devido à assimetria introduzida pela constante b é possível observar que são gerados harmônicos pares e impares. (REISS; MCPHERSON, 2014)

3.2 Efeito 2

Definido pela equação :

$$y[n] = sgn(x[n])\mathbb{R}((acosh(x[n]\pi a + b))) \tag{3.3}$$

Onde o ganho de entrada a pode ter um valor de 0.5 a 3.5, e b de 1 a 1.2. Na imagem 12 é possível observar a curva característica deste efeito com $b = 1$.

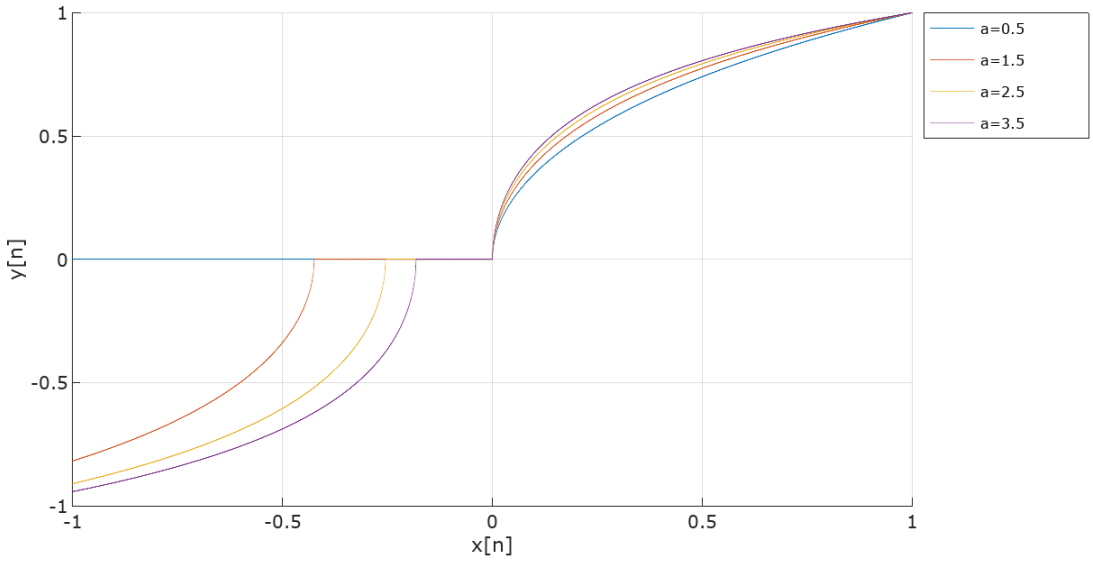

Figura 12 – Curva característica com diversos valores para o ganho de entrada a

A mudança no ganho de entrada nesse efeito define quanto da parte da parte negativa do sinal será descartada. A mudança no valor de b representa uma mudança nos valores associados à valores muito pequenos.

Na figura 13 é possível observar o espectro após aplicação do efeito 2.

Figura 13 – Espectro após aplicação do efeito 2

3.3 Efeito 3

Definido pela equação :

$$y[n] = \tanh(x[n]\pi a + b) \tag{3.4}$$

Onde a pode ter um valor de 1 a 2 , e b de -0.5 a 0.5.

Na imagem 14 é possível observar a curva característica deste efeito , para diferentes valores de a e b constante em 0.

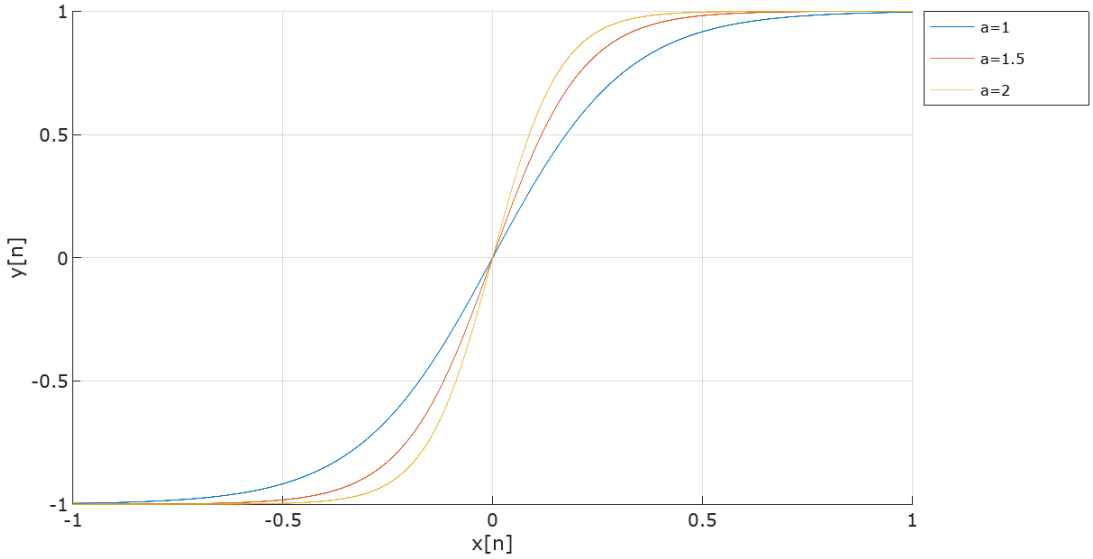

Figura 14 – Curva característica com diversos valores para o ganho de entrada a

A forma dessa curva característica é de corte suave , onde a transição para a região de clipping é suave.Esse tipo de transição cantos arredondados na forma de onda. (REISS; MCPHERSON, 2014)

Na figura 15 é apresentado o espectro após aplicação do efeito 3.

Devido à simetria da função são gerados harmônicos ímpares. Mudar o valor de b gera assimetria na função, gerando harmônicos pares e ímpares. Para adicionar harmônicos pares basta mudar o valor de b.

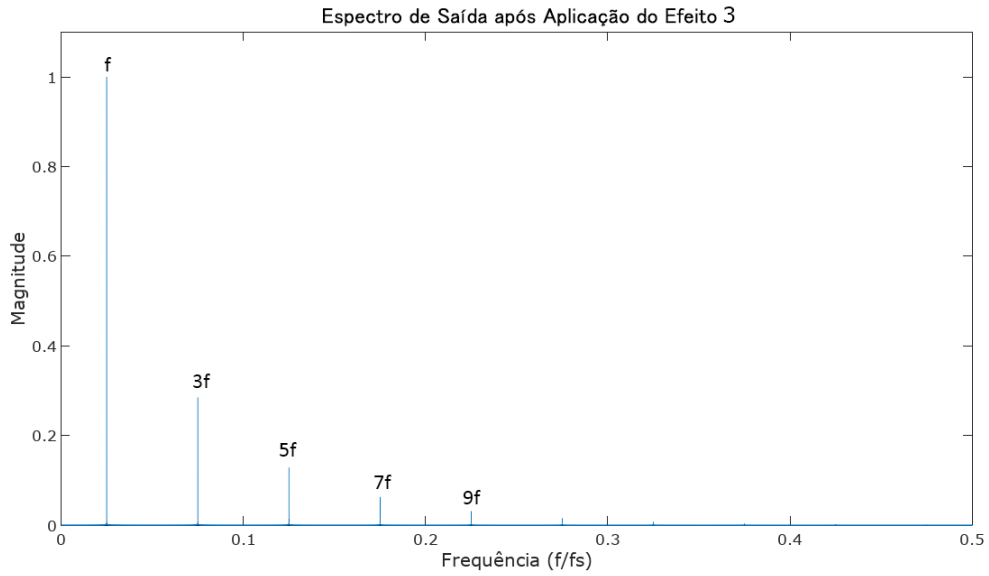

Figura 15 – Espectro após a aplicação do efeito 3 com b=0

3.4 Efeito 4

Definido pela equação :

$$y[n] = 1.6sgn(x[n])\mathbb{R}((x[n]^{0.45})) - 0.6 \tag{3.5}$$

Na imagem 16 é possível observar a curva característica deste efeito.

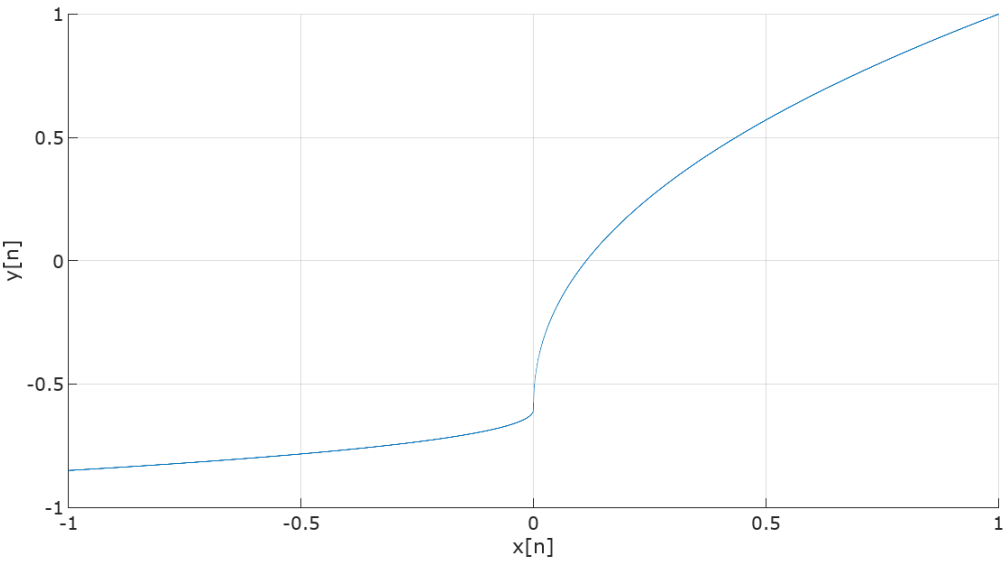

Figura 16 – Curva característica do efeito 4.

Essa função apresenta um comportamento diferente entre valores positivos e negativos. De forma que se um sinal senoidal for utilizado na entrada as parte positivas e negativas apresentaram diferentes formatos de saída.

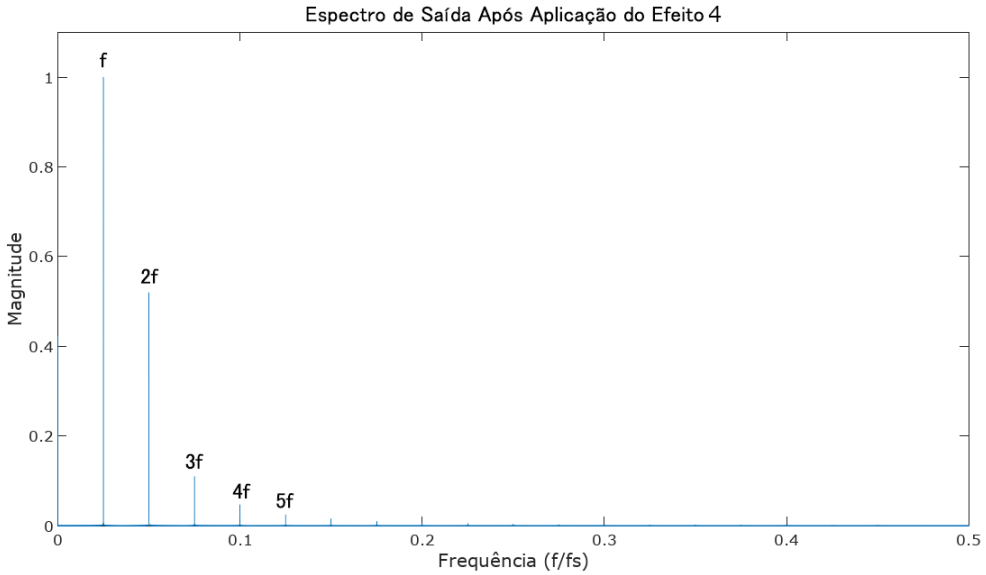

Figura 17 – Espectro após a aplicação do efeito 4

Na figura 17 é apresentado o espectro após aplicação do efeito 4.

4 Metodologia

Após a simulação optou-se por criar um protótipo eletrônico capaz de processar curvas editadas por um usuário. O objetivo do aparelho possibilitar a utilização de curvas características que podem ser alteradas e testadas em tempo real. Um exemplo de curva arbitrária pode ser visualizada na figura 18.

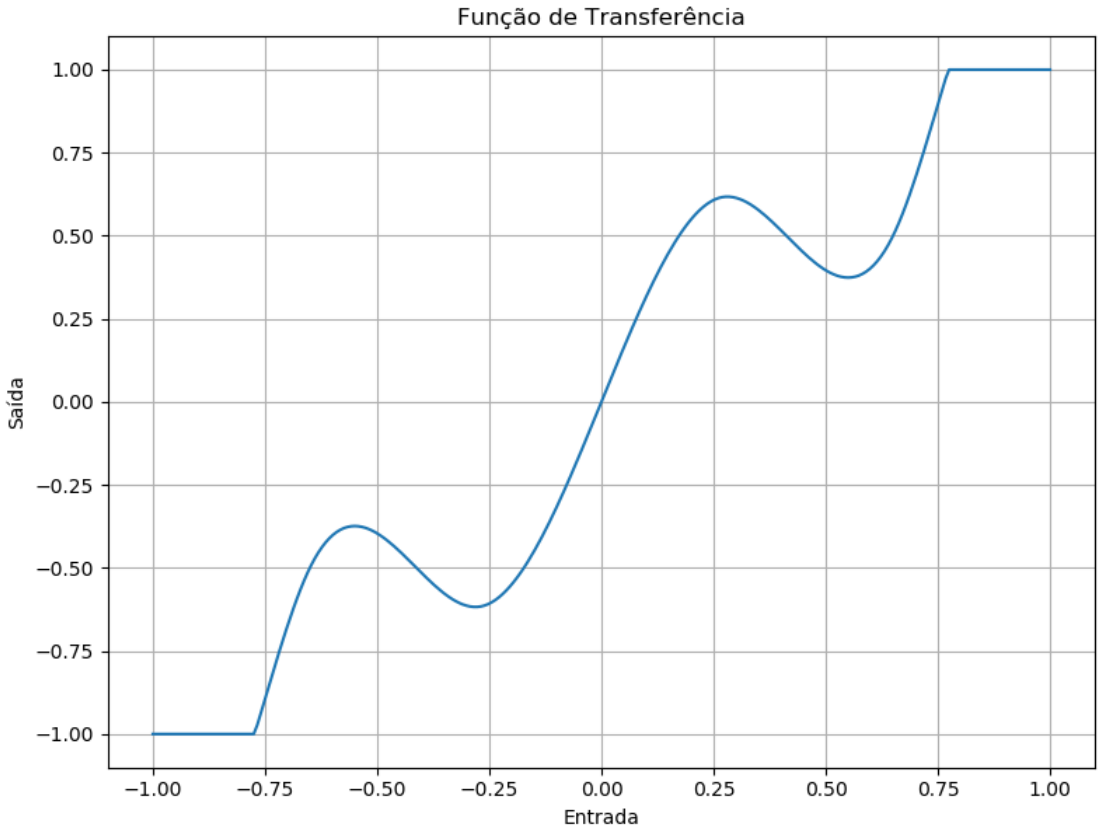

Figura 18 – Curva característica arbitrária.

Neste trabalho é proposto um protótipo que permite que o usuário possa manipular curvas características de distorção e aplicar em uma guitarra para obter o som desejado.

A estrutura consiste de um microcontrolador, conversor digital analógico, um software escrito em Python e circuito para condicionar o sinal vindo da guitarra. O microcontrolador é utilizado para fazer o processamento a amostragem do sinal. O conversor analógico digital recebe as amostras processadas e converte em um sinal analógico que pode ser amplificado. O programa em Python permite criar a curva desejada e envia as informações necessárias para o microcontrolador.

Uma representação do funcionamento pode ser visualizada na figura 19.

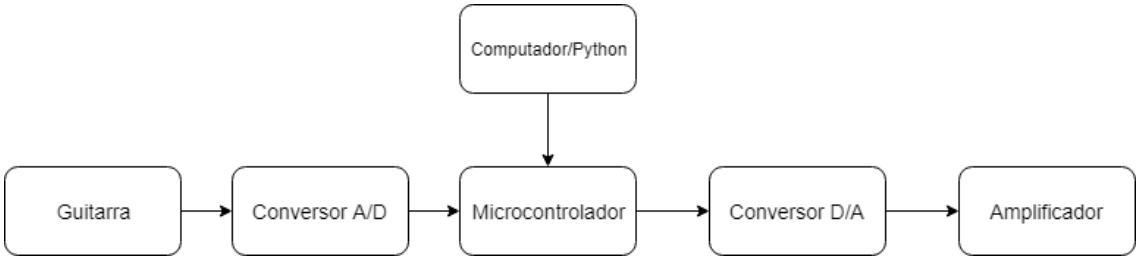

Figura 19 – Diagrama geral de funcionamento

4.1 Manipulação da Curva Característica

O protótipo permite que uma curva característica seja manipulada pelo usuário. Essa entrada de valores é feita em um software de computador.

Os valores de entrada são pontos x e y que são interpolados por uma spline cúbica. Este tipo de interpolação é uma função definida por partes por polinômios. As partes são intervalos entre os pontos de x. Cada parte é definida por um polinômio cubico. Um exemplo pode pontos interpolados dessa forma pode ser visto na figura 20.

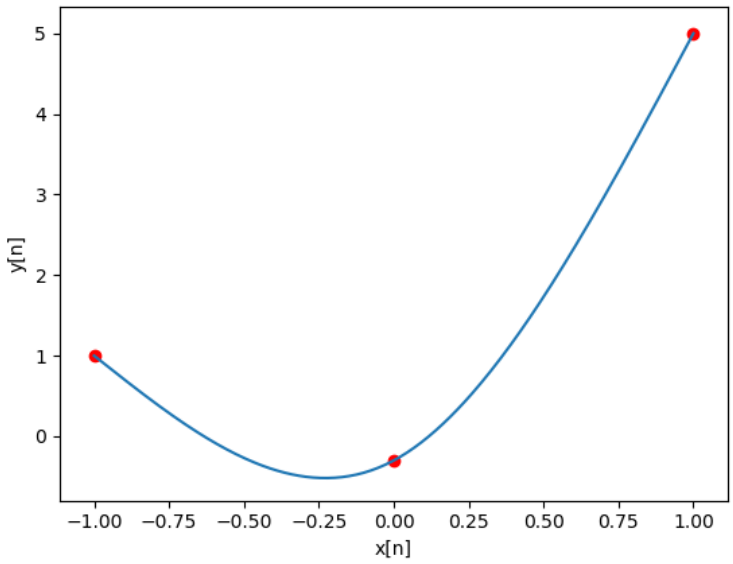

Figura 20 – Três pontos interpolados por uma spline cúbica.

Para aplicar a distorção basta realizar a amostra da guitarra e verificar em qual intervalo o valor da amostra de encontra e então aplicar o polinômio correspondente.

O programa de interface foi escrito em python e utiliza a biblioteca scipy para calcular os coeficientes. Os coeficientes são armazenados em uma matriz c onde $c[k, i]$ é um coeficiente para $(x - x[i])^{3-k}$ no seguimento entre $x[i]$ e $x[i + 1]$. O tamanho da matriz é 4 x $(n - 1)$, onde n é a quantidade de pares de pontos. Estes valores podem ser transferidos para o microcontroladore assim obter o efeito sonoro. (SCIPY.ORG, 2016)

Considerando uma spline unidimensional para um conjunto de n+1 pontos $(y_0, y_1, ..., y_n)$. A i-ésima parte da spline é representa por : (WEISSTEN)

$$Y_i(x) = a_i + b_i x + c_i x^2 + d_i x^3, \tag{4.1}$$

onde $i = 0, ..., n-1$.

É possível obter os valores dos coeficientes com um sistema de equações. Para que se obtenha a quantidade correta de equações necessárias para a quantidade de variáveis faz-se as segundas derivadas nas extremidades iguais a zero. Essa condição de resolução resulta na chamada spline natural. (WEISSTEN)

4.2 Filtro de Aliasing

Efeitos não-lineares geram harmônicos quando aplicados. Os harmônicos que se estenderem além da frequência máxima permitida pela taxa de amostragem retornam para frequências mais baixas, gerando aliasing e prejudicando a qualidade do som. Para amenizar esse efeito um processo de filtragem é aplicado.

O princípio de funcionamento é aumentar a taxa de amostragem do sinal, aplicar a distorção e filtrar os harmônicos. Com a maior de taxa de amostragem o limite de filtragem se estende e permite que harmônicos antes se tornariam aliasing sejam filtrados.

A primeira etapa é a sobreamostragem conforme a equação 2.10. Fazendo uma subamostragem de ordem seis, obtemos o sinal original com cinco zeros entre cada amostra e com taxa de amostragem multiplicada por seis.

A próxima etapa é filtrar a imagem espectral gerada pela inserção de zeros. A equação 2.11 representa um filtro ideal aplicado no domínio da frequência. Para a aplicação foi utilizado um filtro fir de ordem 47. Neste caso há uma grande quantidade de zeros, fazendo que muitas multiplicações não sejam necessárias de modo que o filtro fir é preferível ao iir em questão de quantidade de operações. A figura 21 mostra o gráfico mostra a resposta em frequência do filtro ideal e o real.

A seguir é aplicado o efeito como descrito na seção anterior. Outro filtro é aplicado para evitar aliasing após a subamostragem. O filtro iir implementado tem frequência de corte em torno de 20kHz.

A etapa final é a de subamostragem onde se reduz a taxa de amostragem. Neste caso a taxa de amostragem original era de 16kHz. Considerando que depois a distorção o som da guitarra pode ter frequências em todo o espectro audível optou-se por reduzir a taxa de amostragem do sinal subamostrado para 48kHz. Uma a cada duas amostras do processo de sobreamostragem são enviadas ao DAC de modo a obter essa taxa de amostragem.

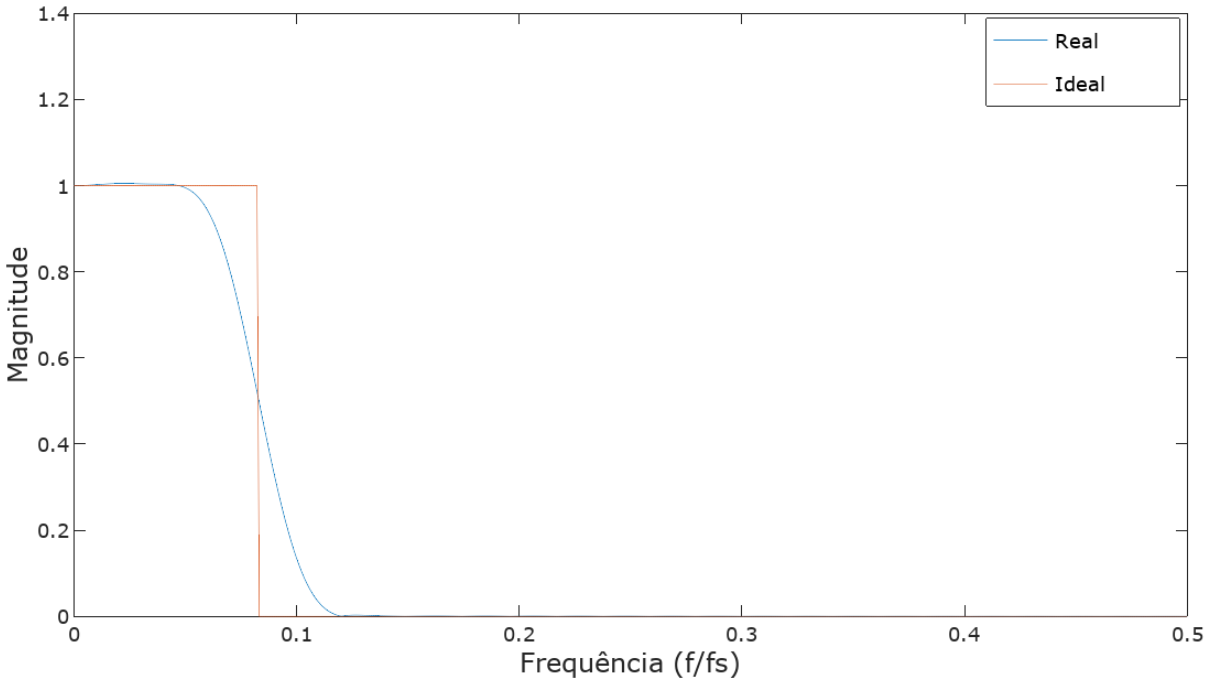

Figura 21 – Comparação entre o filtro ideal e o real.

4.3 Circuito Condicionador

Esta é a etapa em que os sinal da guitarra se torna apropriado para a conversão AD. São feitos ajustes no ganho dc, filtragem e inversão para o conversor AD que é diferencial.

A figura 22 mostra a versão final projetada.

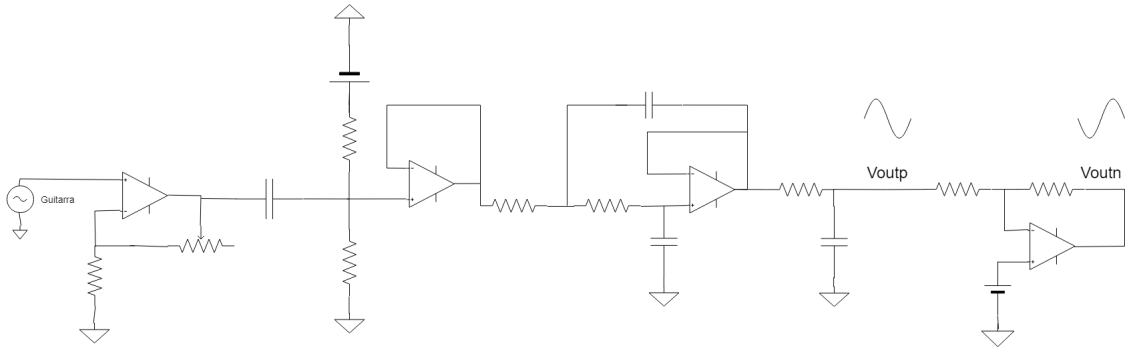

Figura 22 – Circuito condicionador

A figura 23 mostra o sinal de entrada e de saída:

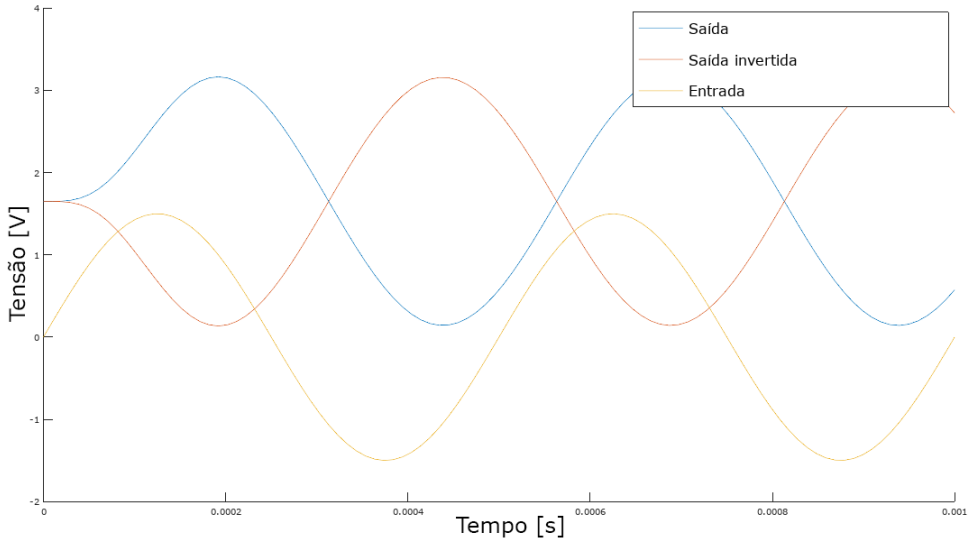

Figura 23 – Entrada e saída do circuito condicionador.

O comportamento em frequência pode ser visto a seguir:

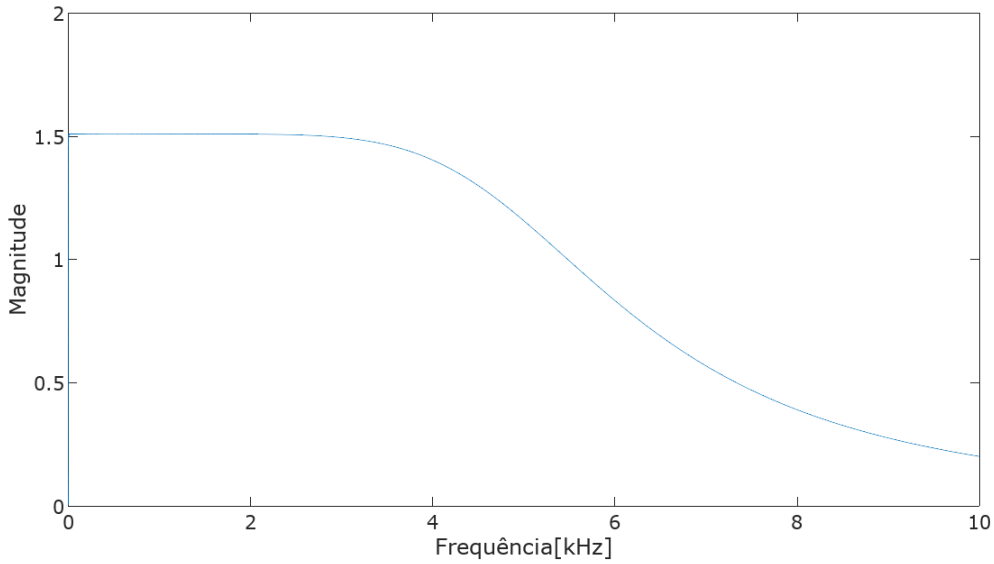

Figura 24 – Resposta em frequência do circuito condicionador.

4.3.1 Ganho AC

Como é possível observar na figura 8 o grau de distorção é dependente da amplitude. Para permitir uma variação na intensidade do efeito foi adicionada uma etapa de ganho que aumenta a amplitude da guitarra. A topologia utilizada foi a do amplificador não-inversor, representado na figura 25.

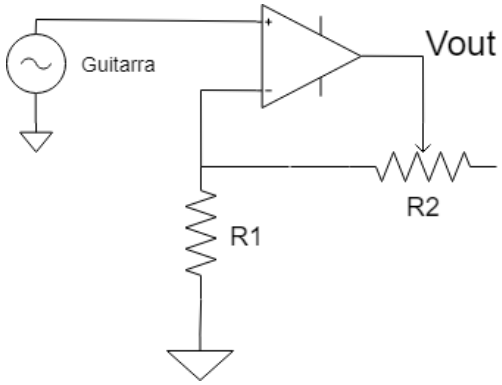

Figura 25 – Filtro de ganho AC.

A equação que representa o comportamento do circuito:

$$A = 1 + \frac{R_2}{R_1} \tag{4.2}$$

Fazendo R_2 sendo um potenciômetro de $50K\Omega$ e R_1 sendo uma resistência de $20K\Omega$ obtém-se um ganho máximo de 3.5 e mínimo de aproximadamente 1.

4.3.2 Ganho DC

O conversor AC aceita tensões de 0 a 3,3V e a guitarra tem uma tensão de saída aproximadamente de -1,5 a 1.5 V (valores observados experimentalmente com uma guitarra com captador passivo). Para que a conversão ocorra corretamente é necessário que o intervalo de tensão seja adaptado para o alcance do conversor AD.

Esta parte do circuito tem está representado na figura 26

Ele exerce duas funções, remover qualquer nível DC proveniente do instrumento e adicionar um nível DC em $3.3/2V$.

O comportamento pode ser analisado utilizando o princípio da superposição. Se colocarmos a fonte AC em repouso o capacitor passa a estar conectado com o ground. Assim o capacitor irá carregar e se tornar um circuito aberto, de forma que sobra apenas um divisor de tensão em $3.3/2V$. Se colocarmos a fonte DC em repouso obtemos um capacitor em série com dois resistores em paralelo. Esta configuração é de um filtro passa alta com frequência de corte em $1/\pi RC$, considerando que os dois resistores são iguais. Escolhendo $C = 68nF$ e $R = 5.8M\Omega$ obtemos um corte até algo em torno de $1.67Hz$. Somando a contribuição das duas fonte obtemos um sinal com dc filtrado com $1.65V$ de nível DC.

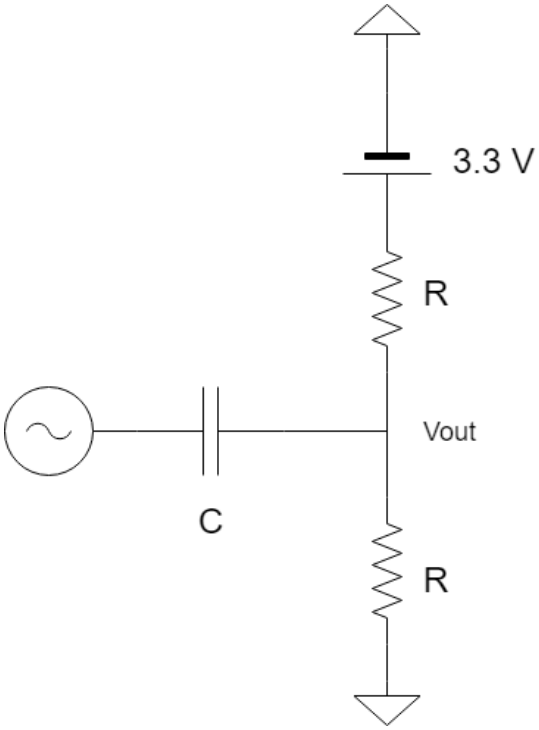

Figura 26 – Filtro de ganho DC.

4.3.3 Filtro Passa-Baixa

Segundo o teorema de Nyquist é necessário que a frequência máxima do sinal a ser amostra seja no máximo metade da frequência de amostragem. Por isso foi utilizado um filtro para remover essas frequências não desejadas.

Segundo (CASE, 2011) não há muitos componentes de frequência importantes para a guitarra não-distorcida acima dos $5kHz$. Por isso foi projetado um filtro ativo de terceira ordem com frequência de corte em torno de $5500kHz$.

Ele consiste de um filtro de topologia Sallen-Key de segunda ordem com ganho unitário em série com um filtro de primeira ordem, representado na figura 27.

O comportamento de um filtro Butterworth de terceita ordem com frequência de corte $\omega_c = 1rad/s$ é (DORF; SVOBODA, 2013):

$$A(s) = \frac{1}{(1+s+s^2)(1+s)} \tag{4.3}$$

A equação que define o comportamento do filtro de segunda ordem é a seguinte(MANCINI, 2003) :

$$A(s) = \frac{1}{1+as+bs^2} \tag{4.4}$$

O comportamento do filtro depende dos coeficientes a e b. O filtro Butterworth foi

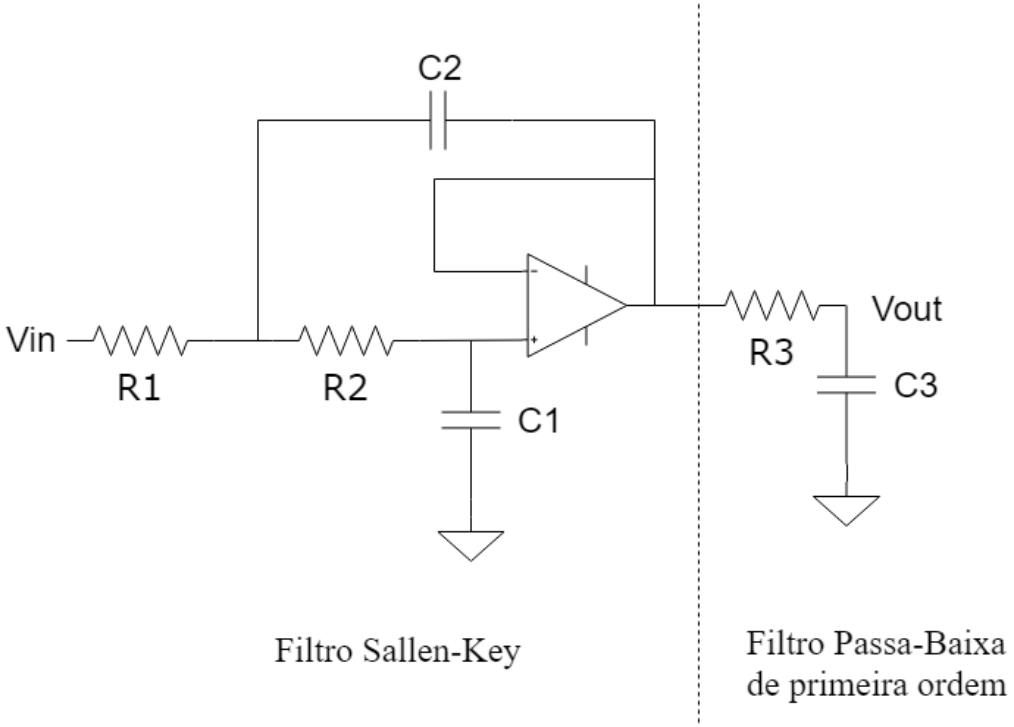

Figura 27 – Filtro passa-baixas de terceira ordem.

selecionado pela planaridade na banda de passagem. Para essa topologia $a = \omega_c C_1(R_1 + R_2)$ e $b = \omega_c^2 R_1 R_2 C_1 C_2$. (MANCINI, 2003)

Com base nas equações foram selecionados os componentes $R_1 = 34k\Omega$, $R_2 = 14k\Omega$, $R_3 = 10k\Omega$, $C_1 = 0.6nF$, $C_2 = 3.3nF$, $C_3 = 3.3nF$. Obtendo assim um filtro com atenuação de $3dB$ em $5.3kHz$.

4.3.4 Inversor de amplitude

A última etapa necessária é inverter o sinal obtido pois o conversor AD é diferencial. Para isso foi utilizado um amplificador inversor, representado na figura 28.

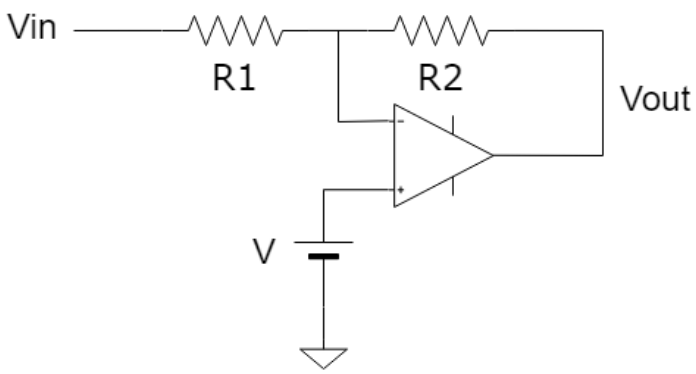

Figura 28 – Amplificador inversor.

Com a saída sendo representada pela seguinte equação:

$$V_{out} = V\left(\frac{R_2}{R_1} + 1\right) - \frac{R_2}{R_1} Vin \tag{4.5}$$

Neste caso o nível DC deve permanecer igual, para isso V deve ser igual ao nível DC de V_in. É possível considerar V_{in} como sua composição AC e DC $V_{in} = V_{dc} + V_{ac}$. Fazendo $R_1 = R_2$ e $V_{dc} = V$ é obtido um sinal de mesma amplitude DC e amplitude AC invertida.

4.4 Microcontrolador

O microcontrolador utilizado foi o C2000 Delfino MCU F28379D. Foi escolhido este modelo pelo seu processador de 200MHz e conversor AD interno de 16 bits. Este clock permite que as operações de filtragem sejam processadas em no intervalo de tempo disponível. O conversor AD interno acelera o processo de amostragem pois não necessita de um processo externo de comunicação

O programa utilizado para programar foi o Code Composer Studio 8.1.0. Foi utilizado o C2000Ware que contém drivers e bibliotecas que aceleram o processo de desenvolvimento.

O código possui duas partes principais, há um processo inicial de configuração e após se inicial um ciclo que se repete enquanto o aparelho estiver ligado.

Na configuração inicial se configura a função de pinos utilizados, é habilitado a comunicação via SCI pelo cabo USB com o computador, é configurado um timer e o ADC.

Um esquemático que representa o comportamento geral está representado na figura 29.

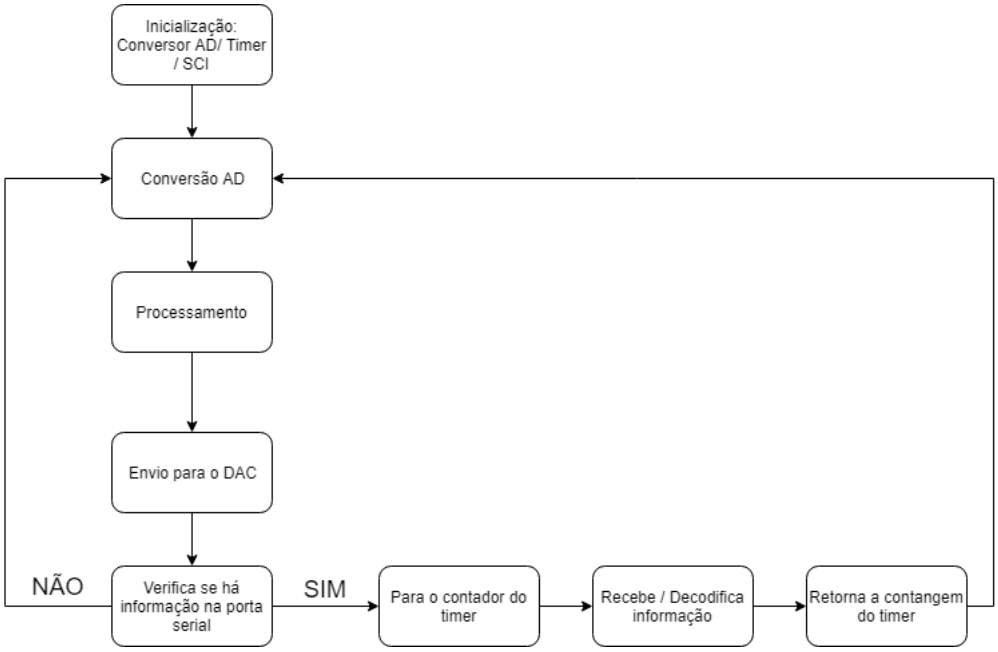

Figura 29 – Diagrama de rotina executada pelo microcontrolador.

4.4.1 Conversor AD

O conversor analógico digital disponível nesta Launchpad possui funcionamento diferencial. São utilizados dois pinos para realizar a medida, sendo um a entrada positiva e o outro a entrada negativa. O resultado obtido é então a diferença de tensão entre os dois pinos. Com 16 bits de precisão é obtido um valor digital entre 0 e 65535.

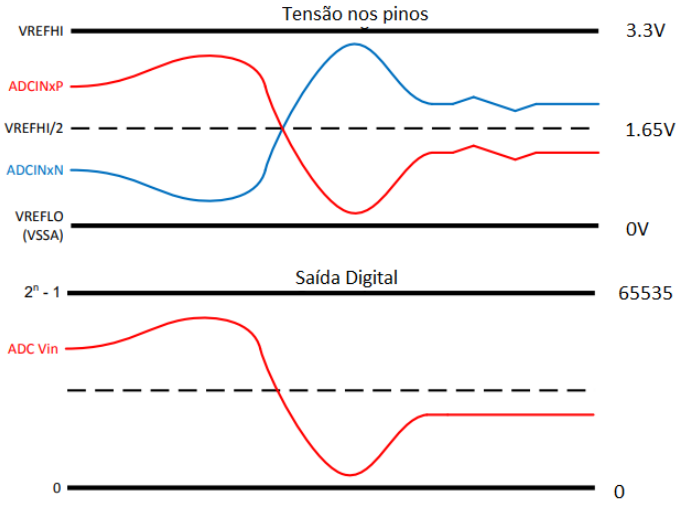

Figura 30 – Modo de sinalização diferencial. Fonte: Adaptado de TEXAS INSTRUMENTS (2013)

Na configuração é dita a fonte do clock, o par de pinos diferencias e como e feita iniciação de um novo ciclo de conversão, que é configurada para se iniciar via coman-

dando de software. Cada conversão ocorre quando requerido via software, o que ocorre em sincronia com as interrupções geradas pelo timer.

4.4.2 Timer

O timer é uma forma de manter tarefas que necessitam se repetir periodicamente. É baseado em um clock, é feita um incremento em cada ciclo, e quando um valor é alcançada uma flag é configurada em nível lógico alto. O número limite de contagem define o período entre cada flag. Dessa forma pode-se alcançar a taxa de amostragem desejada, pois a cada período pode-se iniciar uma nova conversão AD.

O timer foi configurado para gerar interrupções a cada $62.5\mu s$, ou seja com uma frequência de $16kHz$ capaz de amostrar corretamente o espectro significativo da guitarra.

4.4.3 Comunicação Serial

A microcontrolador possui um meio de comunicação para o computador via Serial Communication Interface (SCI). Foi utilizado para receber os coeficientes dos polinômios e os pontos do eixo x. Esses dados são utilizados para aplicar a distorção.

Os dados recebidos são enviados por uma porta serial aberta pela pySerial. Os coeficientes são dependentes do formato da curva e mudam de quantidade conforme varia a quantidade de pontos no eixo. Pela característica dinâmica esses valores são armazenados em um vetor alocado dinamicamente na memória do microcontrolador.

A mensagem recebida possui os coeficientes truncados e separados pela letra d. Uma sequência de valores é recebida e armazenada em uma string, quando se recebe a letra d a sequência é convertida em float e armazenada em um vetor e outra sequência se inicia até o fim da recepção.

4.5 Conversor Digital Analógico

Para obter o som após o processamento digital é necessário fazer a conversão de valores numéricos para tensão. Para isso é utilizado um conversor digital-analógico(DAC).

A primeira opção era de utilizar o conversor DAC8552 que possui comunicação SPI e aceita clock de até 30MHz, que permite uma comunicação rápida e que otimiza o tempo disponível para processamento. Por problemas de funcionamento optou-se por utilizar o PCM5102a que possui comunicação I2S. Dessa forma foi necessário que fosse inserido um microcontrolador que possui suporte para a comunicação I2S.

Após amostragem e processamento na F28379D a amostra é enviada para o microcontrolador ESP8266 em pacotes de quatro bits para minimizar o atraso introduzido

por esse processo intermediário.

Na figura 31 representa a ligação destes componentes com o amplificador.

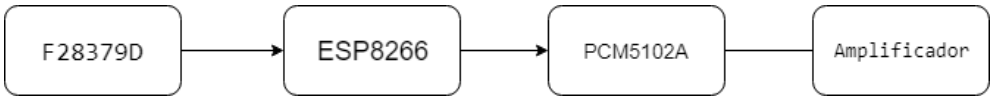

Figura 31 – Diagrama de envio para DAC

O envio é feito para apenas um canal e o outro é mantido em nível zero.

A comunicação possui três pinos, LRCLK, BCK e DATA. LRCLK é o pino responsável por comunicar para que canal a mensagem será enviada. BCK é usado para indicar quando realizar a leitura de um bit disponível no pino DATA. A ordem de envio é com o bit mais significativo enviado primeiro.

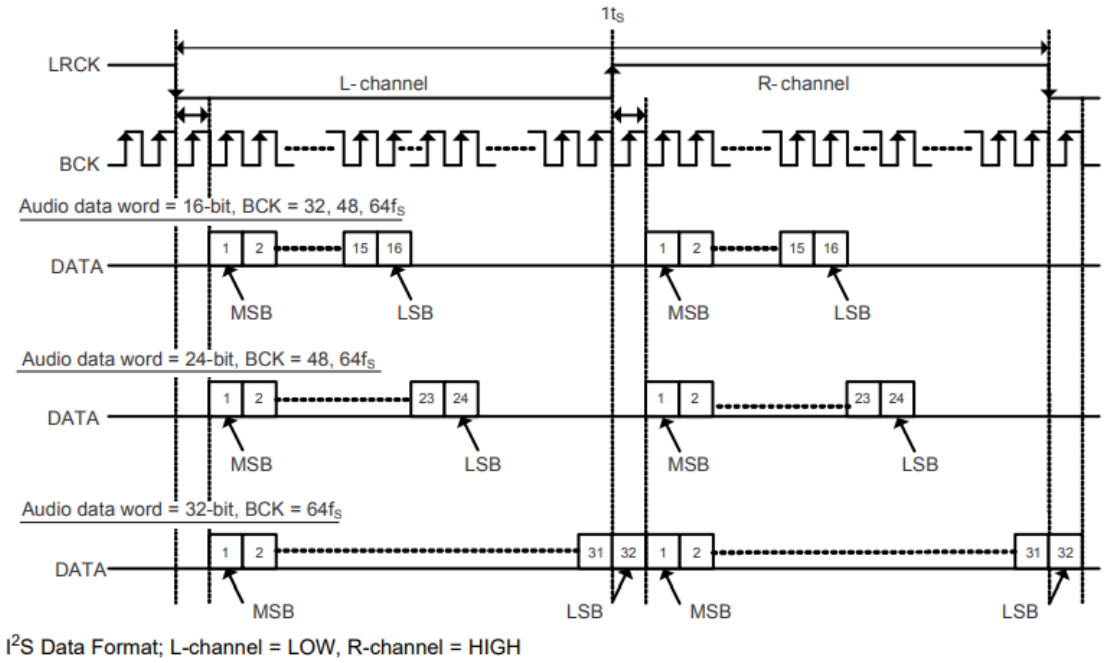

Figura 32 – Comunicação I^2S. Fonte: TEXAS INSTRUMENTS (2012)

4.6 Interface

A interface foi criada em Python com auxílio das bibliotecas Tkinter, Matplotlib e Scipy. A comunicação serial é feita com o pySerial.

A curva a ser editada fica em uma instância do widget Canvas do Tkinter. É uma estrutura que permite exibir gráficos e possui métodos que permitem acessar a posição de movimento e cliques do ponteiro do mouse.

A curva característica é desenhada com interpolação do tipo Spline. A entrada do usuário são pontos que serão então interpolados e mostrados na janela gráfica. Os pontos são entrada do usuário, eles podem ser apagados e arrastados para moldar a curva.

O programa é inicializado com dois vetores, um para os pontos no eixo x e no eixo y. Os valores são -1 e 1 para os dois, pois assim se inicia com o sinal saindo não distorcido. Pontos são então adicionados ou removidos com controle do usuário.

Os efeitos podem ser salvos ou carregados com os botões disponíveis. Os arquivos são salvos em formato .txt e contém os pares de pontos em x e y. Os pontos são limitados entre -1 e 1 para manter os valores dentro do esperado dos limites estabelecidos. Se a interpolação passar deste limite é feito um corte.

A janela gráfica pode ser vista na figura 33.

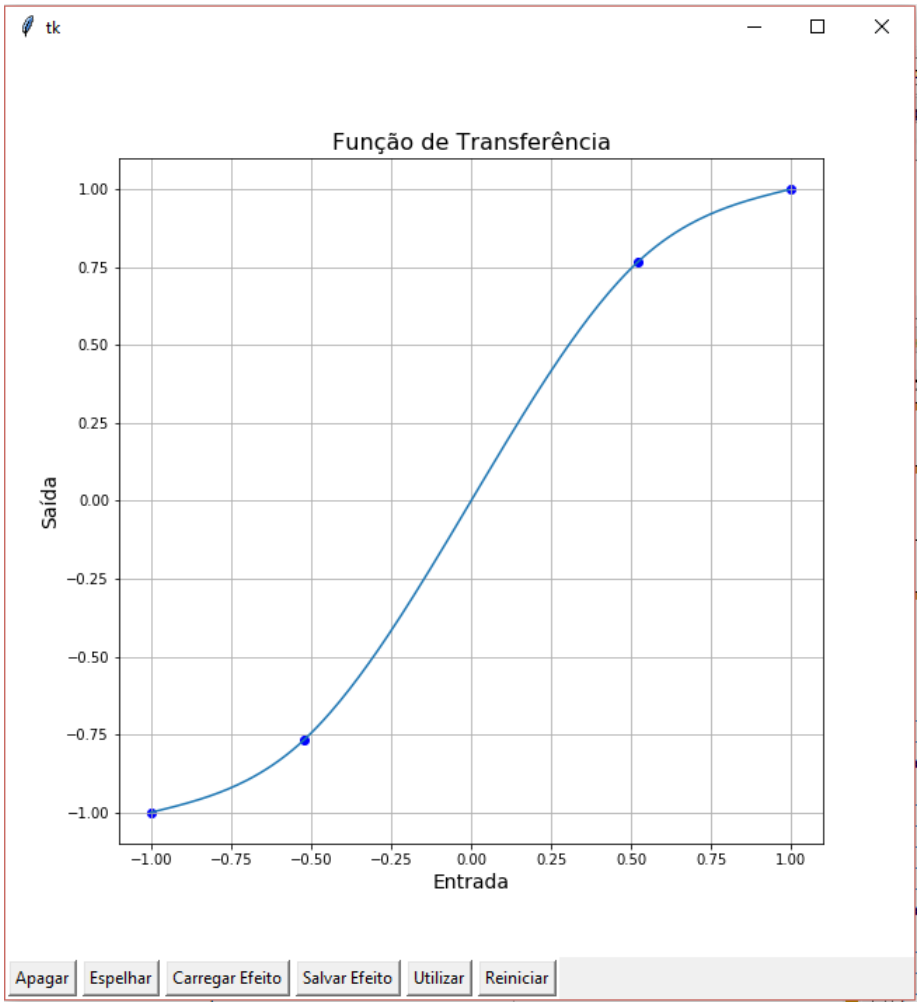

Figura 33 – Interface de usuário.

Para obter um curva característica simétrica há um botão responsável por espelhar os pontos positivos do eixo x.

Com o botão 'utilizar' é formada uma mensagem com a quantidade de pontos no

eixo x, os pontos correspondentes e os coeficientes dos polinômios, que são escritos na porta serial e então recebidos pelo microcontrolador.

4.7 Placa de Circuito Impresso

Para conectar os componentes foi projetada uma placa de circuito. O resultado está na imagem 34.

Foram utilizados dois conectores jack p-10 para receber a guitarra e para a saída para o amplificador. Os amplificadores operacionais utilizados foram dois CIs LM324N. Há um conector p4 para conectar uma fonte de 5V, utilizada para fornecer a tensão negativa para os amplificadores. Diversas associações de componentes foram feitas para obter valores calculador teoricamente.

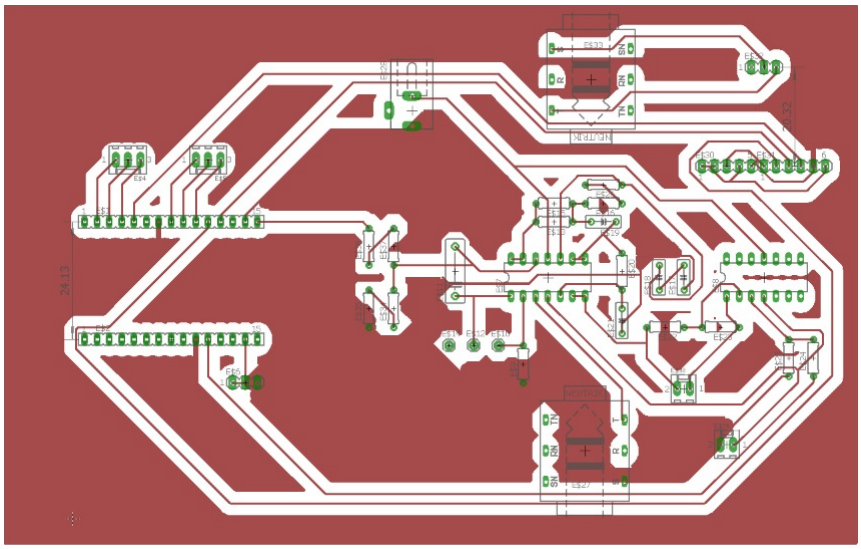

Figura 34 – Conexões da placa de circuito impresso.

O microntralador principal é conectado com o esp8266 por meio de jumpers conectados em pinos soldados na placa.

Na figura 35 o protótipo montado.

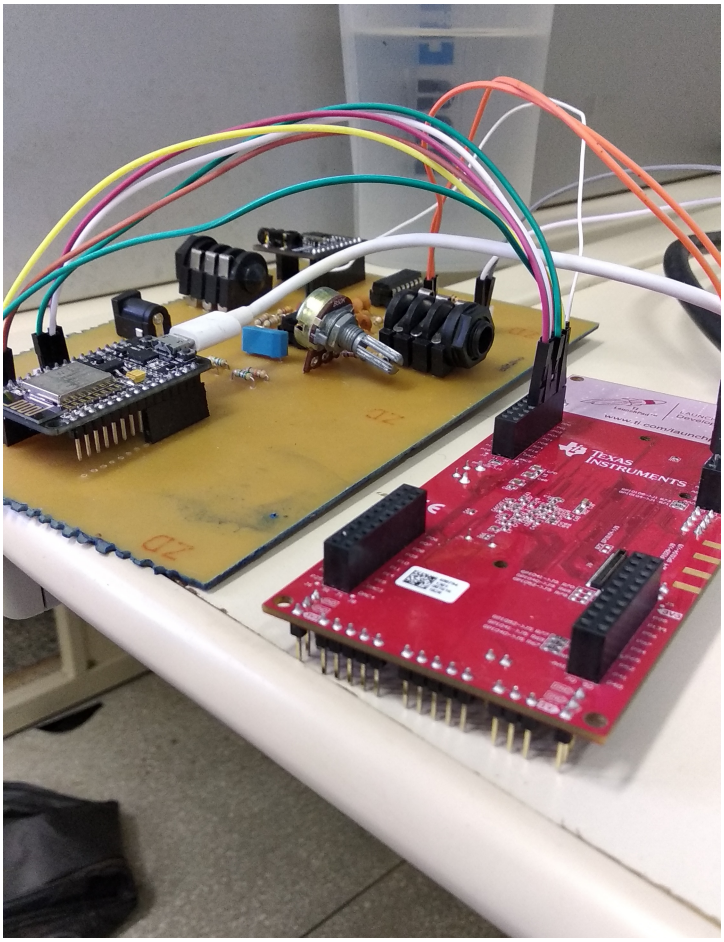

Figura 35 – Protótipo construído.

5 Resultados

Para analisar os resultados foram escolhidos quatro curvas arbitrarias analisando a mudança sonora e um teste com o processo aplicado em uma senoide pra fins de comparação com o esperado idealmente em termos de amplitude dos harmônicos gerados e performance do filtro anti-aliasing. Observando os espectros obtidos é possível analisar se os filtros digitais funcionam como esperado.

O áudio resultante foi gravado pela porta P2 com o programa Audacity 2.2.2.

Para analisar os harmônicos gerados foi utilizado um espectro de entrada padrão sendo uma senoide em 250Hz.

O eixo é normalizado em relação à frequência de amostragem f_s. O eixo está limitado até $0.5f_s$ para melhor visualização.

Arquivos de áudio estão disponíveis em https://soundcloud.com/vinicius-zschitschick-de-oliveira/sets/tcc. Os arquivos terminados em software se referem aos sons obtidos na simulação. Os arquivos terminados em hardware foram gravados diretamente no protótipo construído.

5.1 Efeito 1

Efeito baseado em um dos resultados obtidos na seção de simulação que possui uma curva característica com comportamento de corte suave. Pode ser observado na figura 36.

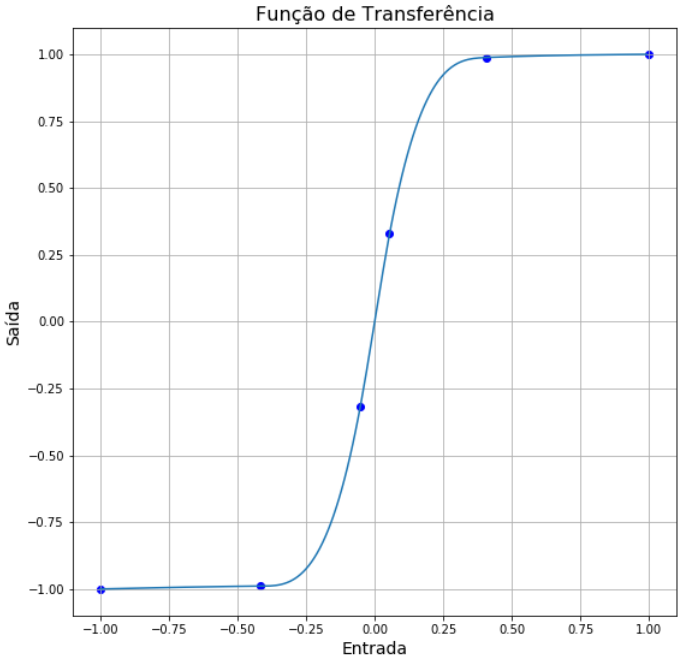

Figura 36 – Curva característica do efeito 1

Na figura 37 há a comparação entre uma senoide após aplicação do efeito idealmente e obtido pelo protótipo.

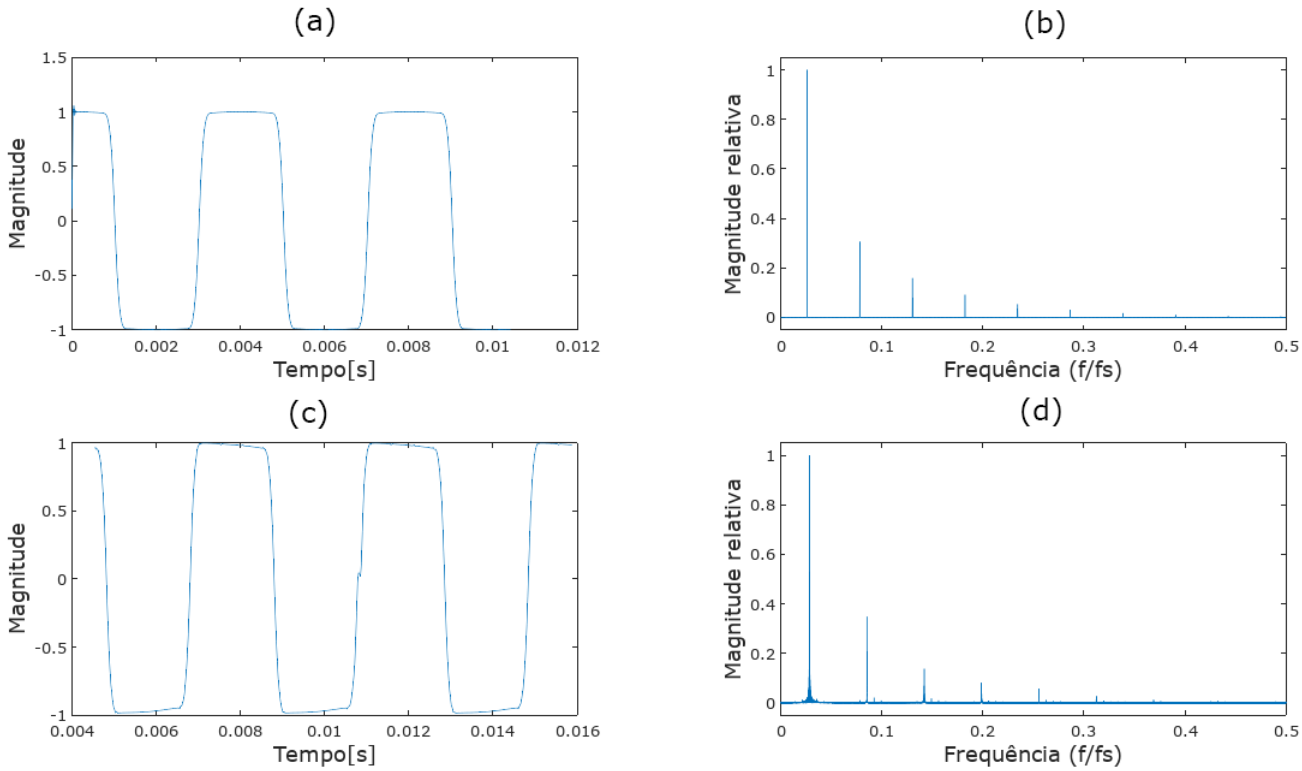

Figura 37 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 1. Em (a) o formato teórico ideal, em (b) o espectro correspondente, em (c) o formato obtido e em (d) o espectro correspondente.

No espectro obtido pode-se observar que para curvas mais básicas a amplitude dos harmônicos se assemelham muito ao esperado. O aliasing gerado é desprezível em comparação com os harmônicos principais. Como esperado de uma curva não muito não-linear o grau de distorção no som da guitarra é baixo.

5.2 Efeito 2

Curva característica na figura 38.

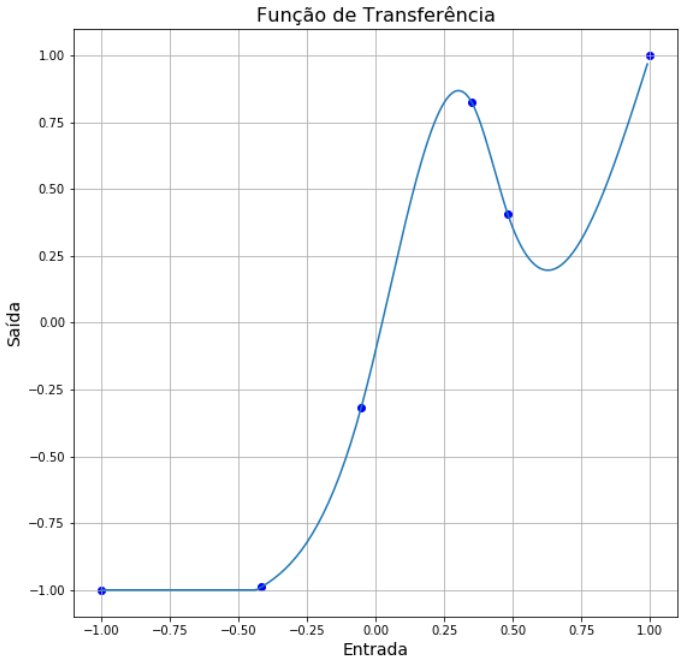

Figura 38 – Curva característica do efeito 2

Na figura 39 a comporação do espectro ideal e do gerado pelo protótipo.

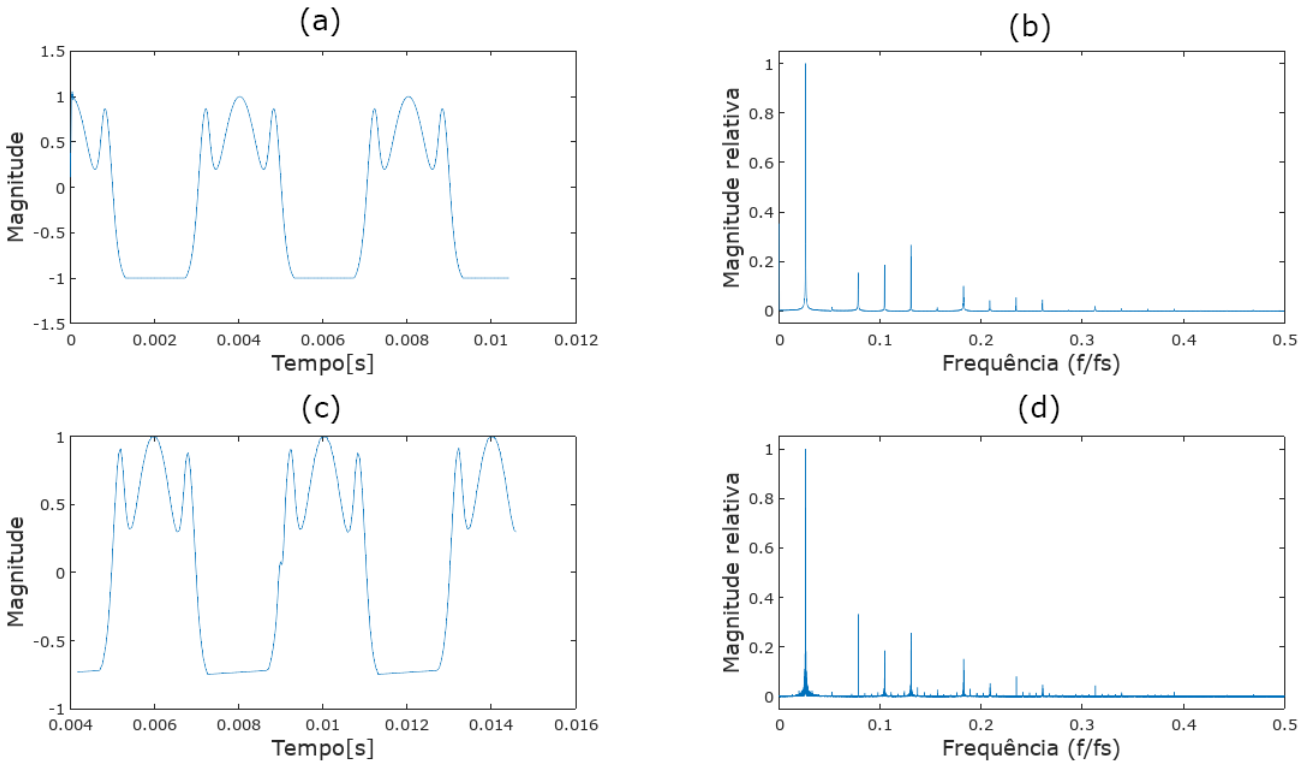

Figura 39 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 2. Em (a) o formato teórico ideal, em (b) o espectro correspondente, em (c) o formato obtido e em (d) o espectro correspondente.

O efeito 2 possui um certo grau de não-linearidade. A amplitude dos harmônicos não seguem exatamente o modelo ideal mas possuem a mesma tendência. É possível observar alguns componentes de aliasing, mas pequenos em relação aos componentes principais. A sonoridade é altamente distorcida e gera harmônicos em quase todo espectro audível. A sonoridade pode-se ser aplicada em gêneros específicos de música.

5.3 Efeito 3

Curva característica na figura 40.

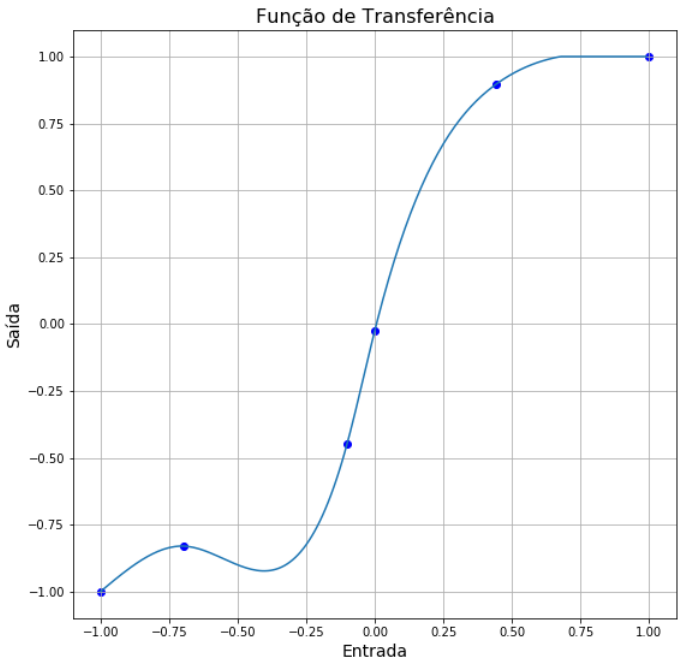

Figura 40 – Curva característica do efeito 3

Os harmônicos do efeito 3 apresentaram pequenas diferenças de amplitude, também apresenta poucos componentes de aliasing.

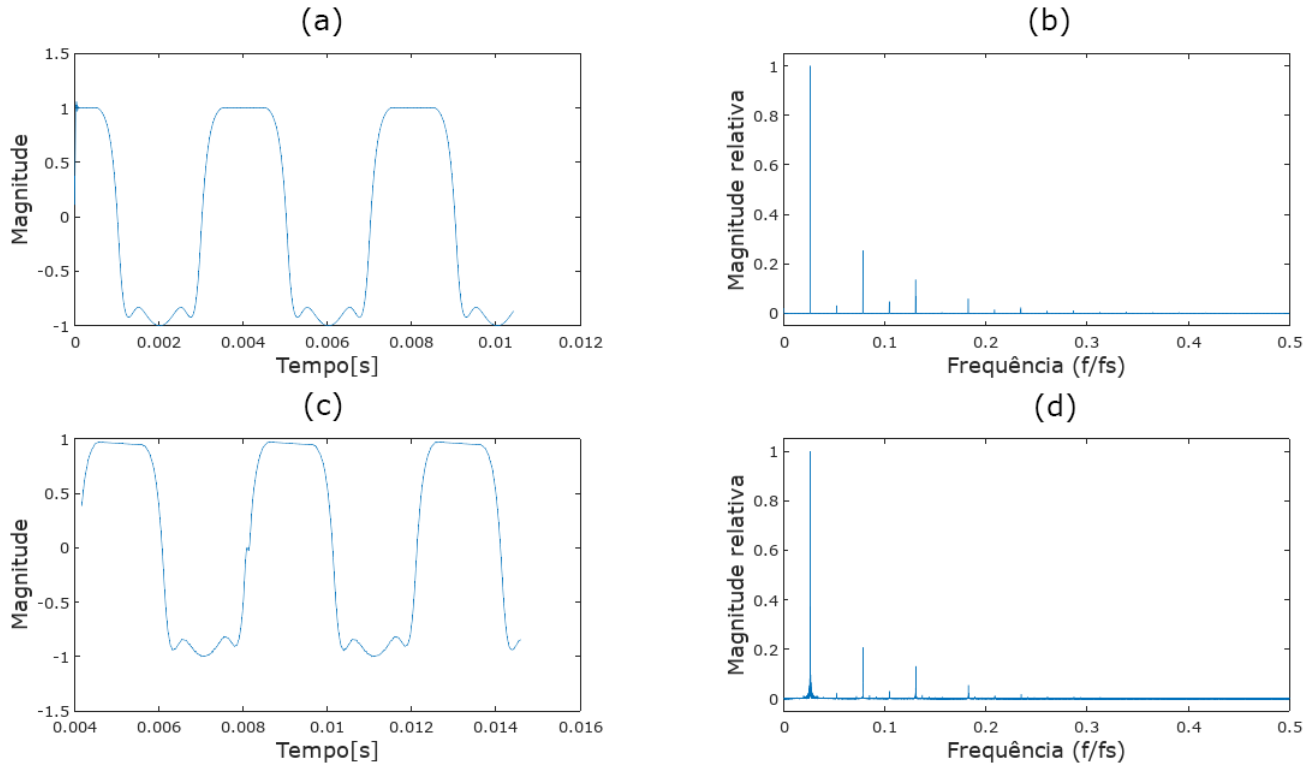

Figura 41 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 3. Em (a) o formato teórico ideal, em (b) o espectro correspondente, em (c) o formato obtido e em (d) o espectro correspondente.

5.4 Efeito 4

Curva característica na figura 42.

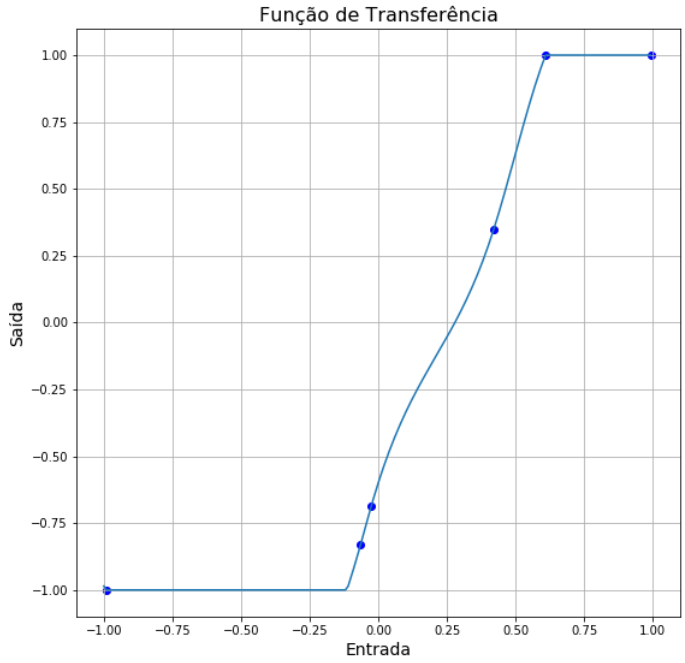

Figura 42 – Curva característica do efeito 4

Este efeito possui comportamento de um corte abrupto assimétrico.

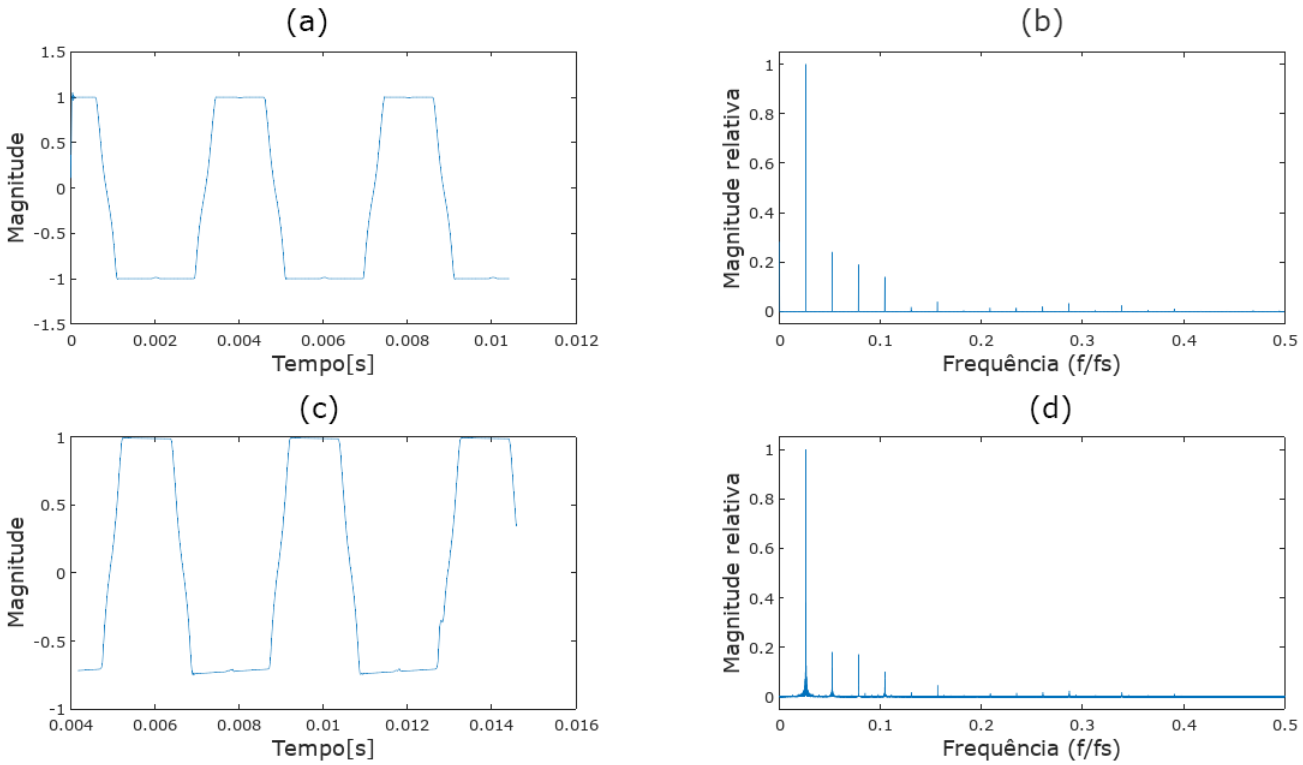

Figura 43 – Comparação entre espectro e forma de onda ideais e espectro obtido para o efeito 4. Em (a) o formato teórico ideal, em (b) o espectro correspondente, em (c) o formato obtido e em (d) o espectro correspondente.

De modo geral o protótipo funcionou como esperado com alguns pontos que são

possíveis de se ajustar. Para melhorar a qualidade do filtro anti-aliasing pode-se aumentar o grau de sobreamostragem. Há dois problemas que precisam ser melhorados para atingir uma qualidade de som. Existe a presença de ruído introduzido pelo circuito e uma clique periódico que não se descobriu a origem.

Uma possível função seria um filtro passa-baixa ajustável na saída, permitindo um corte suave de frequências mais altas e diminuindo a aspereza sonora em alguns casos.

Durante os testes e comparando o som com o da simulação observou-se que o efeito não depende apenas da curva, mas depende também do volume da guitarra, da posição do potenciômetro, da força da palhetada e posição no braço do instrumento. Há também distorção de intermodulação que faz com que seja necessário testar o efeito com notas melódicas e harmônicas.

6 Conclusões

Foram feitas simulações para analisar como funções trigonométricas e hiperbólicas funcionam como função de transferência para arquivos de música. Com as simulações foi possível observar que algumas das funções podem funcionar como curva característica de efeitos não-lineares e que geram diferentes harmônicos entre si, de forma a mudança no timbre é diferente para cada efeito.

Com os resultados das simulações foi feito um protótipo que permite editar curvas características que são aplicadas em uma guitarra com auxílio de um microcontrolador em tempo real.

Foram obtidas diversas curvas que mostram a possibilidade de explorar diversas sonoridades, porém com resultados similares a distorções já existentes. Diversos pontos podem ser melhoradas no protótipo, mas foi possível demonstrar o conceito proposto.

Referências

ANSELMI, J. J. *Ride the Feedback: A Brief History of Guitar Distortion*. 2017. Disponível em: <https://noisey.vice.com/en_ca/article/wn7ja9/ride-the-feedback-a-brief-history-of-guitar-distortion>. Citado na página 11.

CASE, A. *Mix smart: Pro audio tips for your multitrack mix*. [S.l.]: Focal Press, 2011. Citado na página 39.

DORF, R.; SVOBODA, J. *Introduction to Electric Circuits, 9th Edition*. [S.l.]: John Wiley and Sons, Incorporated, 2013. Citado na página 39.

FARNELL, A. *Designing sound*. [S.l.]: Mit Press, 2010. Citado na página 17.

GRAHAM, R. *A Foundation for Electronic Music*. 2nd. ed. [S.l.]: Roland Corporation, 1978. Citado 3 vezes nas páginas 13, 14 e 15.

MANCINI, R. *Op amps for everyone: design reference*. [S.l.]: Newnes, 2003. Citado 2 vezes nas páginas 39 e 40.

POHLMANN, K. *Principles of Digital Audio*. [S.l.]: McGraw-Hill, 2011. Citado 5 vezes nas páginas 13, 14, 15, 16 e 17.

REISS, J. D.; MCPHERSON, A. *Audio effects: theory, implementation and application*. [S.l.]: CRC Press, 2014. Citado 12 vezes nas páginas 11, 16, 17, 18, 19, 20, 22, 23, 24, 25, 28 e 30.

RUSS, M. *Sound synthesis and sampling*. [S.l.]: Focal Press, 2012. Citado 2 vezes nas páginas 16 e 24.

SCIPY.ORG. *scipy.interpolate.CubicSpline*. 2016. Disponível em: <https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.interpolate.CubicSpline.html>. Citado na página 34.

SELF, D. et al. *Audio engineering: Know it all*. [S.l.]: Newnes, 2009. v. 1. Citado 2 vezes nas páginas 13 e 17.

TEXAS INSTRUMENTS. *PCM510xA 2.1 VRMS, 112/106/100 dB Audio Stereo DAC with PLL and 32-bit, 384 kHz PCM Interface*. [S.l.], 2012. Citado na página 44.

TEXAS INSTRUMENTS. *TMS320F2837xD Dual-Core Delfino^{TM} Microcontrollers*. [S.l.], 2013. Citado na página 42.

WEISSTEN, E. W. *Cubic Spline*. Disponível em: <http://mathworld.wolfram.com/CubicSpline.html>. Citado na página 35.

WEISSTEN, E. W. *Fourier Series*. Disponível em: <http://mathworld.wolfram.com/FourierSeries.html>. Citado na página 15.

ZÖLZER, U. *Digital audio signal processing*. [S.l.]: John Wiley & Sons, 2008. Citado 2 vezes nas páginas 22 e 23.

ZÖLZER, U. *DAFX: digital audio effects*. [S.l.]: John Wiley & Sons, 2011. Citado na página 18.

Apêndices

APÊNDICE A – Código da interface

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from matplotlib.backend_bases import MouseEvent
from tkinter import *
from scipy.interpolate import CubicSpline
import serial
import numpy as np



class Curva:

    def __init__(self,  window):
        self.window = window
        self.x=[-1,1]
        self.y=[-1,1]
        self.plot()
        self.click_holding=False
        self.x_selecao=[]

        toolbar = Frame(window,bg='white')

        # Cria botoes, associa metodos e mostra na janela

        self.b = Button(toolbar, text="Apagar", command=self.apagar_ponto)
        self.b2 = Button(toolbar, text="Espelhar", command=self.mirror)
        self.b3 = Button(toolbar, text="Carregar Efeito", command=self.load
            )
        self.b4 = Button(toolbar, text="Salvar Efeito", command=self.save)
        self.b5 = Button(toolbar, text="Utilizar", command=self.save_placa)
        self.b6 = Button(toolbar, text="Reiniciar", command=self.reinit)

        self.b.pack(side=LEFT, padx=2, pady=2)
        self.b2.pack(side=LEFT, padx=2, pady=2)
        self.b3.pack(side=LEFT, padx=2, pady=2)
        self.b4.pack(side=LEFT,padx=2,pady=2)
        self.b5.pack(side=LEFT,padx=2,pady=2)
        self.b6.pack(side=LEFT,padx=2,pady=2)

        toolbar.pack(side=LEFT, fill=X)
```

```
def plot (self):

    self.fig = Figure(figsize=(9,9))

    self.a = self.fig.add_subplot(111)
    self.a.scatter(self.x,self.y,color='blue')    # Plota os pontos

    self.a.grid()
    cs=CubicSpline(self.x,self.y,bc_type='natural') # Interpolação cú
        bica dos dados atuais
    xs = np.arange(-1, 1, 0.01) # Pontos para plotar o gráfico
    self.a.plot(xs,np.clip(cs(xs),-1,1))

    self.a.set_title ("Função de Transferência", fontsize=16) # Títulos
    self.a.set_ylabel("Saída", fontsize=14)
    self.a.set_xlabel("Entrada", fontsize=14)

    self.canvas = FigureCanvasTkAgg(self.fig, master=self.window)
    self.canvas.get_tk_widget().pack()
    self.canvas.draw() # Plota o gráfica na janela gráfica


    self.fig.canvas.callbacks.connect('button_press_event',self.clica)

    self.fig.canvas.callbacks.connect('button_release_event', self.
        solta)

def solta(self,event): # Função chamada quando o clique no mouse é
    solto, lê a posição onde o ponto foi solto e grava no vetor de
    pontos
    self.click_holding=False
    self.fig.canvas.callbacks.disconnect('motion_notify_event')
    if ( event.xdata != None and event.ydata != None ):

        if (abs(event.xdata)<=1 and abs(event.ydata)<=1):

            self.atualiza_vetores(event.xdata,event.ydata)
            self.x_selecao=[event.xdata,event.ydata]
            self.atualiza_grafico_definitivo()

def clica(self,event): # Função chamada quando em cliques do mouse , se
    o clique for em cima de um ponto existente o ponto é substituído
    self.click_holding=True
    self.fig.canvas.callbacks.connect('motion_notify_event', self.
        segura)
```

```
        for i in range(len(self.x)):
            if event.xdata>0:
                if ((self.x[i]*0.85))<(event.xdata)<(self.x[i]*1.15):
                    self.x.pop(i)
                    self.y.pop(i)

                    break
            if event.xdata<0:
                if ((self.x[i]*0.85))>(event.xdata)>(self.x[i]*1.15):
                    self.x.pop(i)
                    self.y.pop(i)

                    break


    def segura(self,event): # Atualiza o gráfico com pontos temporários
        enquanto não escolhido um ponto definitivo

        if (self.click_holding):
            if ( event.xdata != None and event.ydata != None ):

                if (abs(event.xdata)<=1 and abs(event.ydata)<=1):
                    self.atualiza_grafico_temp(event.xdata,event.ydata)


    def atualiza_grafico_temp(self,a,b): # mantem um gráfico temporário
        enquanto há um clique que não foi solto

        x_temp=self.x.copy()
        y_temp=self.y.copy()
        x_temp.append(a)
        x_temp.sort()
        indice=x_temp.index(a)
        y_temp.insert(indice,b)

        self.a.clear()
        self.a.set_title ("Função de Transferência", fontsize=16)
        self.a.set_ylabel("Saída", fontsize=14)
        self.a.set_xlabel("Entrada", fontsize=14)
        self.a.scatter(x_temp,y_temp,color='blue')
        self.a.grid()
        cs=CubicSpline(x_temp,y_temp,bc_type='natural')
        xs = np.arange(-1, 1, 0.05)
        self.a.set_ylim((-1.1, 1.1))
        self.a.set_xlim((-1.1, 1.1))
        self.a.plot(xs,np.clip(cs(xs),-1,1))
```

```
        self.canvas.draw()

    def atualiza_grafico_definitivo(self): # Atualiza o gráfico com o vetor
        de pontos atuais

        self.a.clear()
        self.a.set_title ("Função de Transferência", fontsize=16)
        self.a.set_ylabel("Saída", fontsize=14)
        self.a.set_xlabel("Entrada", fontsize=14)
        self.a.scatter(self.x,self.y,color='blue')
        self.a.grid()
        cs=CubicSpline(self.x,self.y,bc_type='natural')
        xs = np.arange(-1, 1, 0.01)
        if self.x_selecao!=[]: # Vetor com seleção atual para ser plotado
            na cor vermelha
             self.a.scatter(self.x_selecao[0],self.x_selecao[1],color='red')
        self.a.set_ylim((-1.1, 1.1))
        self.a.set_xlim((-1.1, 1.1))
        self.a.plot(xs,np.clip(cs(xs),-1,1))
        self.canvas.draw()

    def atualiza_vetores(self,a,b): # insere os novos valores nos vetores

        self.x.append(a)
        self.x.sort()
        indice=self.x.index(a)
        self.y.insert(indice,b)

    def apagar_ponto(self): # apaga a seleção atual
        indice=self.x.index(self.x_selecao[0])
        self.x.pop(indice)
        self.y.pop(indice)
        self.x_selecao=[]
        self.atualiza_grafico_definitivo()

    def mirror(self): # espelhao os pontos do eixo x positivo
        a=self.x.copy()
        for item in a:
            if item<0:
                self.y.pop(self.x.index(item))
                self.x.remove(item)
        self.x=[-item for item in self.x[::-1]]+self.x
        self.y=[-item for item in self.y[::-1]]+self.y
        self.x_selecao=[]
        self.atualiza_grafico_definitivo()

    def load(self): # Procura um arquivo com efeito e carrega
```

```
        self.window.filename =  filedialog.askopenfilename(initialdir = "/"
            ,title = "Escolher efeito",filetypes = (("Arquivos txt","*.txt")
            ,("all files","*.*")))

        x=[]
        y=[]
        l=1
        try:
            with open(window.filename,'r') as f:

                for item in f:
                    atual=item[:-1]
                    if atual=='y':
                        l=0
                        continue
                    if l :
                        x.append(float(atual))
                    else:
                        y.append(float(atual))
            self.x=x
            self.y=y
            self.atualiza_grafico_definitivo();
        except:
            pass

    def save(self): # escolhe uma pasta e grava o efeito em um arquivo txt
        self.window.filename =  filedialog.asksaveasfilename(initialdir = "
            /",title = "Escolher efeito",filetypes = (("Arquivos txt","*.txt
            "),("all files","*.*")))

        try:
            with open(self.window.filename,'w') as f:

                for item in self.x:
                    f.write('%f\n' % item)

                f.write('%s\n' % 'y')

                for item in self.y:
                    f.write('%f\n' % item)
        except:
            pass


    def save_placa(self): # Cria a mensagem e envia para o microcontrolador
        os coeficientes da spline
```

```
        cs=CubicSpline(self.x,self.y,bc_type='natural')
        #print(sum(cs.c.flatten()))
        try:
            ser = serial.Serial('COM9', 115200)
        except:
            pass


        if(ser.isOpen()):
            msg=''
            msg+=str(len(self.x))+'d'

            lista_aux=[str(round(item,6))+'d' for item in self.x]
            for item in lista_aux:
                msg+=item

            lista_aux=[str(round(item,8))+'d' for item in cs.c.flatten()]

            for item in lista_aux:
                msg+=item

            ser.write(bytes(msg,'utf-8'))

    def reinit(self): # Volta a para os vetores de guitarra limpa

        self.x=[-1,1];
        self.y=[-1,1];
        self.x_selecao=[]
        self.atualiza_grafico_definitivo();


window= Tk()
start= Curva(window)
window.mainloop()
```

APÊNDICE B – Código microcontrolador

```
//
// Headers incluídos
//
#include "driverlib.h"
#include "device.h"
#include <stdlib.h>
#include "stdio.h"


// Prototipo de funçoes
//
void initADCs(void); // Inicializa o conversor AD
__interrupt void cpuTimer0ISR(void);  // Serviço de interrupção do timer
void initCPUTimers(void); // Inicializa o timer
void configCPUTimer(uint32_t, float, float); // Configura o periodo do
    timer
//
// Variaveis Globais
//

uint16_t loopCounter = 0;
float a,somateste=0;
uint16_t byte_recebido;
uint16_t adc_amostra=0;
float dsp_amostra=0;
char amostrar=0;
char amostrar_teste=0;
char phase=0;
char atual=0;
//
// Main
//
void main(void)
{

    float x_oldf2[3]; // Vetor para armazenar amostras de entrada
        anteriores
    float y_oldf2[3]; // Vetor para armazenar amostra de saída anteriores


    float x1,x2=0,x3=0,x4=0,x5=0,x6=0,x7=0,x8=0; // Amostras anteriores
        para filtro FIR
```

```
float b[48]={ -3.5561e-004, -8.9862e-004, -1.4012e-003, -1.7171e-003,
    -1.5647e-003, -6.2813e-004, 1.2452e-003, 3.8314e-003,
                    6.4131e-003, 7.8589e-003, 6.9351e-003, 2.7936e-003,
                        -4.5105e-003, -1.3667e-002, -2.2103e-002,
                        -2.6408e-002,
                    -2.3143e-002, -9.8422e-003, 1.4065e-002, 4.6732e
                        -002, 8.3962e-002, 1.1993e-001, 1.4838e-001,
                        1.6410e-001,
                    1.6410e-001, 1.4838e-001, 1.1993e-001, 8.3962e-002,
                         4.6732e-002, 1.4065e-002, -9.8422e-003, -2.3143
                        e-002,
                    -2.6408e-002, -2.2103e-002, -1.3667e-002, -4.5105e
                        -003, 2.7936e-003, 6.9351e-003, 7.8589e-003,
                        6.4131e-003,
                    3.8314e-003, 1.2452e-003, -6.2813e-004, -1.5647e
                        -003, -1.7171e-003, -1.4012e-003, -8.9862e-004,
                        -3.5561e-004  };
float upsamples[ordem_interp]; // Coeficientes filtro FIR

int xpos=10;
float saida; //resultado filtro fir
float poli_result=0; // resultado da aplicação do polinomio
float amostra=0;
float amostra_float=0.0;
int i=0,j=0; // Contadores

for(i=0;i<ordem_iir;i++){

          x_oldf2[i]=0; // inicializa os vetores
         y_oldf2[i]=0;
    }

int x_size=2; // Quantidade de pontos no eixo x

float * x_val; // Valores dos pontos no eixo x
x_val=malloc((x_size) * sizeof(float));
x_val[0] = -1;
x_val[1] = 1;
float *coeffs;   // Array para armazenar as constantes dos polinomios
coeffs = malloc((x_size-1) * 4 * sizeof(float));
coeffs[0] = 0;
coeffs[1] = 0;
coeffs[2] = 1;
coeffs[3] = -1;
```

```
//
// Configura PLL, desabilita o Watchdog timer, habilita o clock de
    perifericos.
//
Device_init();
//
// Disable pin locks and enable internal pullups.
//
Device_initGPIO();
GPIO_setPinConfig(DEVICE_GPIO_CFG_LED1);
GPIO_setMasterCore(DEVICE_GPIO_PIN_LED1, GPIO_CORE_CPU1);
GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);

GPIO_setDirectionMode(6, GPIO_DIR_MODE_OUT);     // GPIO6 = output
//
// GPIO42 é o pino SCI Rx. Comunicação serial o computador
//
GPIO_setMasterCore(42, GPIO_CORE_CPU1);
GPIO_setPinConfig(GPIO_42_SCITXDA);
GPIO_setDirectionMode(42, GPIO_DIR_MODE_IN);
GPIO_setPadConfig(42, GPIO_PIN_TYPE_STD);
GPIO_setQualificationMode(42, GPIO_QUAL_ASYNC);
//
// GPIO43 é o pino SCI Tx.
//
GPIO_setMasterCore(43, GPIO_CORE_CPU1);
GPIO_setPinConfig(GPIO_43_SCIRXDA);
GPIO_setDirectionMode(43, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(43, GPIO_PIN_TYPE_STD);
GPIO_setQualificationMode(43, GPIO_QUAL_ASYNC);

//
// Inicializa os pinos de comunicação
//
GPIO_setPadConfig(5, GPIO_PIN_TYPE_PULLUP);
GPIO_setPinConfig(GPIO_5_GPIO5);
GPIO_setDirectionMode(5, GPIO_DIR_MODE_IN);

GPIO_setPadConfig(0, GPIO_PIN_TYPE_PULLUP);
GPIO_setPinConfig(GPIO_0_GPIO0);
GPIO_setDirectionMode(0, GPIO_DIR_MODE_OUT);

GPIO_setPadConfig(1, GPIO_PIN_TYPE_PULLUP);
GPIO_setPinConfig(GPIO_1_GPIO1);
GPIO_setDirectionMode(1, GPIO_DIR_MODE_OUT);
```

```
GPIO_setPadConfig(2, GPIO_PIN_TYPE_PULLUP);
GPIO_setPinConfig(GPIO_2_GPIO2);
GPIO_setDirectionMode(2, GPIO_DIR_MODE_OUT);

GPIO_setPadConfig(3, GPIO_PIN_TYPE_PULLUP);
GPIO_setPinConfig(GPIO_3_GPIO3);
GPIO_setDirectionMode(3, GPIO_DIR_MODE_OUT);

GPIO_setPadConfig(4, GPIO_PIN_TYPE_PULLUP);
GPIO_setPinConfig(GPIO_4_GPIO4);
GPIO_setDirectionMode(4, GPIO_DIR_MODE_OUT);
GPIO_writePin(4, 0);


GPIO_setPinConfig(DEVICE_GPIO_CFG_LED1);
GPIO_setMasterCore(DEVICE_GPIO_PIN_LED1, GPIO_CORE_CPU1);
GPIO_setPadConfig(DEVICE_GPIO_PIN_LED1, GPIO_PIN_TYPE_STD);
GPIO_setDirectionMode(DEVICE_GPIO_PIN_LED1, GPIO_DIR_MODE_OUT);
GPIO_writePin(DEVICE_GPIO_PIN_LED1, 0);
//
// Initialize interrupt controller and vector table.
//
Interrupt_initModule();
Interrupt_initVectorTable();
initADCs();

Interrupt_register(INT_TIMER0, &cpuTimer0ISR);

initCPUTimers();

configCPUTimer(CPUTIMER0_BASE, DEVICE_SYSCLK_FREQ, (62.5)/2.0); //

CPUTimer_enableInterrupt(CPUTIMER0_BASE);

Interrupt_enable(INT_TIMER0);

//habilita interrupção globais(INTM) e realtime interrupt (DBGM)
EINT;
ERTM;

//
// Configura o  SCIA.
//
SCI_performSoftwareReset(SCIA_BASE);

SCI_setConfig(SCIA_BASE, DEVICE_LSPCLK_FREQ, 230400, (SCI_CONFIG_WLEN_8
    |
```

```
                                                    SCI_CONFIG_STOP_ONE
                                                          |
                                                    SCI_CONFIG_PAR_NONE
                                                        ));
    SCI_resetChannels(SCIA_BASE);
    SCI_resetRxFIFO(SCIA_BASE);
    SCI_resetTxFIFO(SCIA_BASE);
    SCI_clearInterruptStatus(SCIA_BASE, SCI_INT_TXFF | SCI_INT_RXFF);
    SCI_enableFIFO(SCIA_BASE);
    SCI_enableModule(SCIA_BASE);
    SCI_performSoftwareReset(SCIA_BASE);

#ifdef AUTOBAUD
    SCI_lockAutobaud(SCIA_BASE);
#endif


    char buff[20]; // Variavel para armazenar os bytes recebidos

    char receber=1;
    GPIO_writePin(DEVICE_GPIO_PIN_LED1, 1);
    CPUTimer_startTimer(CPUTIMER0_BASE);

    while(1){

    if(amostrar){

        phase++;
        phase=phase&0x00FF;
        adc_amostra  = (uint16_t)(sine[phase]);

        amostrar=0;

        while(ADC_getInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1) == false)
            {} //Espera o fim da conversão AD

        ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
        adc_amostra = ADC_readResult(ADCARESULT_BASE, ADC_SOC_NUMBER0);
        dsp_amostra = (float)adc_amostra;
        dsp_amostra = (dsp_amostra/65535)*2-1; // Ajusta a faixa de valores
            de 0 a 65535 para -1 a 1
        x1=dsp_amostra;

        // Sobreamostragem e passa-baixas FIR

        upsamples[0]=b[0]*x1+b[6]*x2+b[12]*x3+b[18]*x4+b[24]*x5+b[30]*x6+b
            [36]*x7+b[42]*x8;
```

```
upsamples[1]=b[1]*x1+b[7]*x2+b[13]*x3+b[19]*x4+b[25]*x5+b[31]*x6+b
    [37]*x7+b[43]*x8;
upsamples[2]=b[2]*x1+b[8]*x2+b[14]*x3+b[20]*x4+b[26]*x5+b[32]*x6+b
    [38]*x7+b[44]*x8;
upsamples[3]=b[3]*x1+b[9]*x2+b[15]*x3+b[21]*x4+b[27]*x5+b[33]*x6+b
    [39]*x7+b[45]*x8;
upsamples[4]=b[4]*x1+b[10]*x2+b[16]*x3+b[22]*x4+b[28]*x5+b[34]*x6+b
    [40]*x7+b[46]*x8;
upsamples[5]=b[5]*x1+b[11]*x2+b[17]*x3+b[23]*x4+b[29]*x5+b[35]*x6+b
    [41]*x7+b[47]*x8;

x8=x7;
x7=x6;
x6=x5;
x5=x4;
x4=x3;
x3=x2;
x2=x1;


// Procura em que intervalo as amostras se encontram e aplica a
    distorção com metodo de Horner

for(i=0;i<6;i++){

upsamples[i]=upsamples[i]*6;

for(j=0;j<x_size-1;j++){
    if(x_val[j]<=upsamples[i] && x_val[j+1]>=upsamples[i]){
        xpos=j;
        atual=xpos;
        }
    }

    amostra=upsamples[i]-x_val[xpos];
    a=amostra;
    poli_result=coeffs[xpos]; // Aplica o polinômio com o método de
        Horner
    poli_result=poli_result*amostra+coeffs[x_size-1+xpos];
    poli_result=poli_result*amostra+coeffs[2*(x_size-1)+xpos];
    poli_result=poli_result*amostra+coeffs[3*(x_size-1)+xpos];

    if(poli_result>1.0){
        poli_result=1.0;
    }
    if(poli_result<-1.0){
        poli_result=-1.0;
```

```
}

upsamples[i]=poli_result;
a=poli_result;
}

for(i=0;i<6;i++){

amostra=upsamples[i];
saida=amostra*(0.098531)+0.295593*x_oldf2[0]+(0.295593)*x_oldf2
    [1]+(0.098531)*x_oldf2[2]-(-0.577241)*y_oldf2[0]-(0.421787)*
    y_oldf2[1]-(-0.056297)*y_oldf2[2];

x_oldf2[2]=x_oldf2[1];
x_oldf2[1]=x_oldf2[0];
x_oldf2[0]=amostra;

y_oldf2[2]=y_oldf2[1];
y_oldf2[1]=y_oldf2[0];
y_oldf2[0]=saida;

upsamples[i]=saida;
}

for(i=0;i<6;i=i+2){

adc_amostra  = (uint16_t)(((((upsamples[i])/2+1)/2.0)*65535));
GPIO_writePin(0, adc_amostra & 1 );
GPIO_writePin(1, adc_amostra & 2 );
GPIO_writePin(2, adc_amostra & 4 );
GPIO_writePin(3, adc_amostra & 8 );

GPIO_writePin(4, 1);  // Inicia a Transmissão    ; Pino D6
    nodemcu
while(GPIO_readPin(5)==0){} // Aguarda o sinal de fim de
    leitura

GPIO_writePin(0, adc_amostra & 16 );
GPIO_writePin(1, adc_amostra & 32 );
GPIO_writePin(2, adc_amostra & 64 );
GPIO_writePin(3, adc_amostra & 128 );
GPIO_writePin(4, 0);

while(GPIO_readPin(5)!=0){}

GPIO_writePin(0, adc_amostra & 256 );
GPIO_writePin(1, adc_amostra & 512 );
```

```
            GPIO_writePin(2, adc_amostra & 1024 );
            GPIO_writePin(3, adc_amostra & 2048 );

            GPIO_writePin(4, 1);

            while(GPIO_readPin(5)==0){} // Aguarda o sinal de fim de
                leitura

            GPIO_writePin(0, adc_amostra & 4096 );
            GPIO_writePin(1, adc_amostra & 8192 );
            GPIO_writePin(2, adc_amostra & 16384 );
            GPIO_writePin(3, adc_amostra & 32768 );
            GPIO_writePin(4, 0);

            while(GPIO_readPin(5)!=0){}
        }
    }

        if(SCI_getRxFIFOStatus(SCIA_BASE) != SCI_FIFO_RX0) // Verifica se h
            á um valor disponível no buffer
            //GPIO_writePin(DEVICE_GPIO_PIN_LED1, 0);
            //loopCounter++;
                                                                // e
                inicia o processo de leitura dos valores enviados
        //CPUTimer_stopTimer(CPUTIMER0_BASE);
            {
                for(i=0;i<12;i++) // Recebe valores até receber um valor nã
                    o-numérico . Primeiro valor recebido representa a
                    quantidade de pontos no eixo x
                    {
                        if(receber)
                        {
                            byte_recebido = SCI_readCharBlockingFIFO(SCIA_
                                BASE);

                            if(byte_recebido!=100){ // Verifica se é um
                                valor numérico pelo valor ASCII
                                buff[i]=byte_recebido;
                            }
                            else{
                                receber=0;
                                buff[i]=0;
                                }
                        }
                        else
                        {
                            buff[i]=0;
```

```
            }


        }

    x_size=atoi(buff);//Converte o valor armazena em String em
        um valor inteiro

    receber=1;

    x_val = realloc(x_val,(x_size) * sizeof(float)); // Array
        para armazenar os valores do eixo x

    for(i=0;i<x_size;i++){
        for(j=0;j<12;j++) // Recebe valores até receber um
            valor não-numérico
        {
        if(receber)
            {
            byte_recebido = SCI_readCharBlockingFIFO(SCIA_BASE)
                ;

            if(byte_recebido!=100){ // Verifica se é um valor
                numérico ou a letra 'd' pela tabela ASCII
                buff[j]=byte_recebido;
                }
            else{
                receber=0;
                buff[j]=0;
                }
            }
            else
            {
            buff[j]=0;
            }
            }
        a=atof(buff); // converte a string para float
        x_val[i]=a;

        receber=1;
        loopCounter++;
    }

    coeffs = realloc(coeffs,(x_size-1) * 4 * sizeof(float)); //
        Array para armazenar os coeficientes da Spline

    for(i=0;i<4*(x_size-1);i++){
```

```
                for(j=0;j<20;j++) // Recebe valores até receber um valor nã
                    o-numérico
                  {
                    if(receber)
                    {
                    byte_recebido = SCI_readCharBlockingFIFO(SCIA_BASE);

                    if(byte_recebido!=100){ // Verifica se é o separador 'd'
                    buff[j]=byte_recebido;
                        }
                        else{
                            receber=0;
                            buff[j]=0;
                            }
                    }
                    else
                    {
                    buff[j]=0;
                    }
                    }
                    //a=atof(buff);
                    coeffs[i]=atof(buff);
                    somateste=somateste+coeffs[i];
                    receber=1;
                    a=coeffs[i];

                    }
            }
    }
}


void initADCs(void)
{

    ADC_setPrescaler(ADCA_BASE, ADC_CLK_DIV_1_0); // Divisão do clock de
        entrada
    ADC_setMode(ADCA_BASE, ADC_RESOLUTION_16BIT, ADC_MODE_DIFFERENTIAL); //
         Resolução e modo
    ADC_setInterruptPulseMode(ADCA_BASE, ADC_PULSE_END_OF_CONV);
    ADC_enableConverter(ADCA_BASE); // Habilita o conversor
    DEVICE_DELAY_US(1000);
    ADC_setupSOC(ADCA_BASE, ADC_SOC_NUMBER0, ADC_TRIGGER_SW_ONLY,ADC_CH_
        ADCIN2_ADCIN3, 64); // Pinos, fonte de trigger software e tempo e
        quantidade de ciclos de clock
    ADC_setInterruptSource(ADCA_BASE, ADC_INT_NUMBER1, ADC_SOC_NUMBER0);
    ADC_enableInterrupt(ADCA_BASE, ADC_INT_NUMBER1);
```

```
    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);

}

void
initCPUTimers(void)
{
    //
    // Inicializa o periodo do timer para o máximo
    //
    CPUTimer_setPeriod(CPUTIMER0_BASE, 0xFFFFFFFF);

    //
    // Divisão do clock por 1 (SYSCLKOUT)
    //
    CPUTimer_setPreScaler(CPUTIMER0_BASE, 0);

    //
    // Para o timer
    //
    CPUTimer_stopTimer(CPUTIMER0_BASE);
    //
    // Recarrega o contador do timer
    //
    CPUTimer_reloadTimerCounter(CPUTIMER0_BASE);

}

void
configCPUTimer(uint32_t cpuTimer, float freq, float period)
{
    uint32_t temp;

    //
    // Initialize timer period:
    //
    temp = (uint32_t)(freq / 1000000 * period);
    CPUTimer_setPeriod(cpuTimer, temp);

    CPUTimer_setPreScaler(cpuTimer, 0);

    CPUTimer_stopTimer(cpuTimer);
    CPUTimer_reloadTimerCounter(cpuTimer);

    CPUTimer_enableInterrupt(cpuTimer);

}
```

```
__interrupt void
cpuTimer0ISR(void)
{

    amostrar=1;
    ADC_forceSOC(ADCA_BASE, ADC_SOC_NUMBER0);

    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}
```

APÊNDICE C – Código esp8266

```
#include <Arduino.h>
#include "ESP8266WiFi.h"
#include <i2s.h>
#include <i2s_reg.h>
#define GPIO_IN ((volatile uint32_t*) 0x60000318)

uint16_t amostra=0; // 16 bits sem sinal
uint16_t nib1,nib2,nib3,nib4;
int16_t data;

void setup() {

  i2s_begin(); // inicializa o i2s e associa os pinos internos
  system_update_cpu_freq(160); // Coloca a frequência em 160MHz

  DR_REG_I2S_BASE&(~I2S_MSB_RIGHT);
  i2s_set_rate(48000);

  pinMode(D1, INPUT);
  pinMode(D2, INPUT);
  pinMode(D3, INPUT);
  pinMode(D5, INPUT);
  pinMode(D6, INPUT);
  pinMode(D7, OUTPUT);
  digitalWrite(D7,LOW);

}

void loop() {

  if(GPIP(D6) and (GPIP(D7)==0)){ //GPIP lê o pino
      amostra=0;
      nib3=((uint16_t)*GPIO_IN)&0x4031; // Lê o registrador que contem o
          estado atual dos pinos e faz uma mascara para obter os bits de
          interesse
      GPOS = (1 << D7);  // Sinaliza o fim da leitura  LOW        5
      while(GPIP(D6)!=0){} // Espera novos dados no barramento

      nib4=((uint16_t)*GPIO_IN)&0x4031;
      GPOC = (1 << D7);  // Sinaliza o fim da leitura

      while(GPIP(D6)!=1){} // Espera novos dados no barramento
```

```
        nib1=((uint16_t)*GPIO_IN)&0x4031;
        GPOS = (1 << D7);  // Sinaliza o fim da leitura

        while(GPIP(D6)!=0){} // Espera novos dados no barramento
        nib2=((uint16_t)*GPIO_IN)&0x4031;
        GPOC = (1 << D7);  // Sinaliza o fim da leitura

        while(GPIP(D6)!=1){}

        // D3 = BIT1 = 1 ; D1 = BIT6 = 32; D2 = BIT5 = 16;  D5 = BIT15 =
            16384

        amostra|=((nib3&32)==32)<<8;
        amostra|=((nib3&16)==16)<<9;
        amostra|=((nib3&1)==1)<<10;
        amostra|=((nib3&16384)==16384)<<11;

        amostra|=((nib4&32)==32)<<12;
        amostra|=((nib4&16)==16)<<13;
        amostra|=((nib4&1)==1)<<14;
        amostra|=((nib4&16384)==16384)<<15;

        amostra|=((nib1&32)==32);
        amostra|=((nib1&16)==16)<<1;
        amostra|=((nib1&1)==1)<<2;
        amostra|=((nib1&16384)==16384)<<3;

        amostra|=((nib2&32)==32)<<4;
        amostra|=((nib2&16)==16)<<5;
        amostra|=((nib2&1)==1)<<6;
        amostra|=((nib2&16384)==16384)<<7;

        data=amostra-0x8000;
        i2s_write_lr(data,0);
        ESP.wdtFeed();//Alimenta o Watchdog.

    }
}
```