Simulador de amplificador de áudio

Paulo Augusto M. F. de Souza
Estudante de Engenharia Eletrônica
Faculdade Gama - FGA UnB
Email: pauloaugustomiguelfonseca@gmail.com

Tiago Martins de Brito
Estudante de Engenharia Eletrônica
Faculdade Gama - FGA UnB
Email: tiagomartinsbrito879@gmail.com

***Resumo*—Com o processamento de sinais de áudio, é possivel modificar as características originais do sinal. Por ser de áudio, pode-se ver o seu comportamento em faixas de frequência diferente e com isso equalizar um sinal da maneira pretendida através de filtros. Também é possível amplificar ele de uma maneira que o torne mais intenso ou menos intenso, modificando a sua amplitude.**

***P*alavras-chaves – Frequência, filtro, amplificador, equalizador.**

I. Introdução

A. Justificativa

Para tratamento de sinais de áudio com equipamentos digitais ou analógicos, ainda há um elevado custo, então como esse projeto tem como motivação tentar reduzir esses custos, mas tendo em vista manter uma certa qualidade. Com esse custo reduzido, poderia ser usado para o aprendizado para quem ainda está aprendendo e não quer gastar muito com equipamentos de áudio.

B. Objetivos

O objetivo deste trabalho, é mostrar como pode ser feita a equallização de áudio, com o Raspberry Pi através da amostragem de um sinal.

C. Requisitos

A priori está sendo estudada a possibilidade de usar um conversor AD, de pelo menos 16 bits e uma frequência de amostragem de 40kHz, ou no lugar dele colocar uma interface de áudio, que tem uma taxa de amostragem de 48kHz e de 24 bits. Isso se deve pois o Raspberry não possui conversor AD, então é preciso usar um externo.

Sensores serão usados para colocar algum tipo de efeito durante o processamento do sinal pelo Raspberry, como por exemplo um sensor de distância para colocar a intensidade do efeito. Também pode ser adicionado um VU-Meter, para mostrar as bandas de frequência do sinal.

Para todos os cálculos e processamentos dos sinais, será usado o Raspberry Pi 3. Ele será usado pois, tem uma grande capacidade de processamento, e com uma boa velocidade. O uso dele permite criar sistemas embarcados para essa aplicação de processamento de sinais de áudio.

II. Revisão Bibliográfica

A. Teorema da amostragem

Para o processamento computacional de um sinal analógico, antes é preciso fazer a amostragem do sinal. Isso consiste em discretizar o sinal, ou seja, torna-lo digital e com isso fazer o seu processamento em um computador, ou no Raspberry Pi. Amostrar é definir para um certo intervalo de tempo, um valor para o sinal. Onde esse ele apenas possui um valor definido em um intervalo de tempo específico.

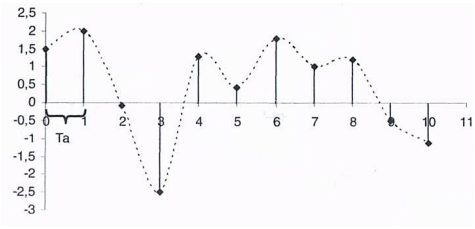

Figura 1. Amostragem

Na Figura 1, é possível ver justamente, onde para cada intervalo de tempo é que se tem uma amostra do sinal, um valor definido apenas em um intervalo de tempo específico. Em intervalos de tempo não definido, o sinal recebe zero como seu valor.

Um meio para as discretização, é usar o critério de Nyquist para amostragem, que diz que a frequência de amostragem deve ser de pelo menos duas vezes a frequência do sinal. Isso se deve para que se possa retomar o sinal original com o mínimo de distorções possíveis. Na Figura 2 é possivel ver justamente esse critério, pois quando a frequência de amostragem é duas vezes menor, há uma sobreposição do sinal, e isso não permite voltar ao sinal original. [4]

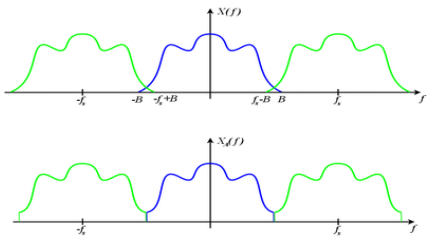

Figura 2. Nyquist

Fonte: https://pt.wikipedia.org/wiki/Teorema_da_amostragem_de_Nyquist-Shannon

B. Processamento de sinais

Em áudio a largura de banda corresponde ao espectro audível, compreendido entre 20 e 20kHz. Então, a partir de

dispositivos de captação sonora, o objetivo é trabalhar com faixas de frequência compreendidas na parte audível. Com isso, se busca então trabalhar com a resposta em frequência dessa faixa. [3]

O processamento desses sinais, podem ser feitos com a aplicação de filtros, que delimitam uma faixa específica para se trabalhar. Esses filtros, que podem ser, passa baixas, passa altas ou passa bandas, servem para fazer a equalização de um sinal de áudio. Além dessa equalização, também podem ser incorporados controles para ajuste da amplitude dos sinais antes e após o processamento. [2]

Filtros passa baixas, são dispositivos que permitem a passagem de um sinal até uma certa frequência máxima. Passa altas, são os que permitem a passagem a partir de uma frequência mínima. E passa bandas, são os que permitem a passagem a partir de uma frequência mínima até uma máxima. As figuras abaixo mostram justamente a resposta em frequência de cada filtro.

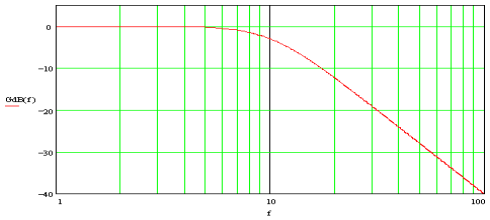

Figura 3. Passa baixas

Fonte:
http://www.alvaroneiva.site.br.com/filteq2_arquivos/image011.gif

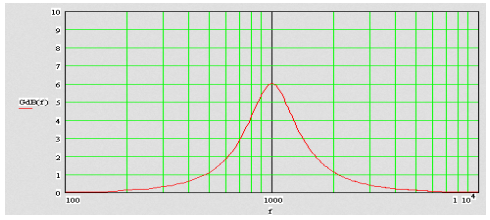

Figura 4. Passa banda

Fonte:
http://www.alvaroneiva.site.br.com/filteq2_arquivos/image015.gif

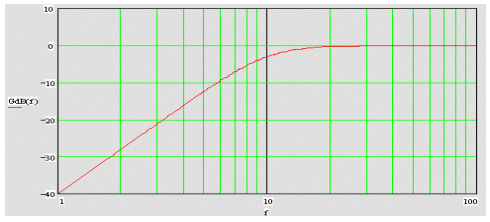

Figura 5. Passa alta

Fonte:
http://www.alvaroneiva.site.br.com/filteq2_arquivos/image013.gif

III. Desenvolvimento

A primeira parte do desenvolvimento, foi conectar a interface de áudio Roland DuoCapture UA11 ao Linux. Essa conectividade foi possível com a entrada P10 da interface, pois quando tentamos usar o P2 não foi possível gravar um arquivo de áudio. A gravação, foi possível usando um software para esse tipo de tarefa, o Ardour, e também um que já vem na Raspberry Pi, o Alsamixer.

Após a gravação, é possível ouvir o áudio gravado, com o Ardour, na saída da interface. Quando a gravação é feita usando o Alsamixer é possível ouvir o som na saída P2 da Raspberry Pi ou via o audio da saída HDMI.

Após a leitura, o próximo passo então é fazer o processamento desse sinal de áudio, adicionar efeitos e aplicar filtros para tratamento sinal recebido na Raspberry. Esses filtros podem ser desenvolvidos na Raspberry e com isso modificar o sinal para que possa ser percebido as diferenças na saída. Com isso, embarcar o sistema utilizando alguns sensores, como o de presença até o momento mencionado.

A. Descrição de Hardware

A ligação da interface com a Raspberry Pi é feita através do cabo USB que permite a sua comunicação. O instrumento é conectado na interface de áudio para que seja feito uma conversão AD para que seja possível a leitura e gravação dos dados.

A conecção do instrumento pode ser feita com o conector de entrada P10 da interface permitindo assim que ela faça a conversão AD. Como essa interface possui uma taxa de amostragem de 48KHz, a qualidade do sinal audível é muito boa, assim é possível fazer uma análise mais robusta do áudio e consequentemente o seu processamento, enfatizando algumas informações extraidas através de filtros, permitindo que sinais semelhantes ao de um simulador de amplificador de guitarra sejam executados como saída do sistema.

Figura 6. Interface de áudio

Fonte: https://www.roland.com/us/products/duo-capture/

A Figura 6 traz a interface de áudio usada para nosso projeto. Nela contém duas entradas e duas saídas, assim é

possível conectar mais de um equipamento para gravação. E atrás dela possui o conector para se ligar o cabo USB para ser feita a conexão com o computador ou a Raspberry Pi.

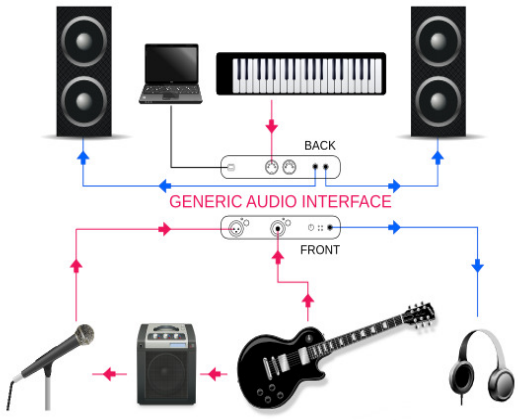

Figura 7. Conecção da interface e o PC

Fonte: http://libremusicproduction.com/sites/default/files/articles/Bedroom%20musician%20setup.png

Na Figura 7 é possível ver como ele pode ser ligado ao computador ou ao Raspberry Pi. Esse tipo de interface permite que não apenas um tipo de instrumento ou microfone possa ser ligado a ela.

Esse tipo de implementação de hardware permite que se faça então a leitura do PC ou da Raspberry do sinal que pode ser tratado e então enviado para saídas de áudio e então ser escutado por uma pessoa através de um fone ou auto falante.

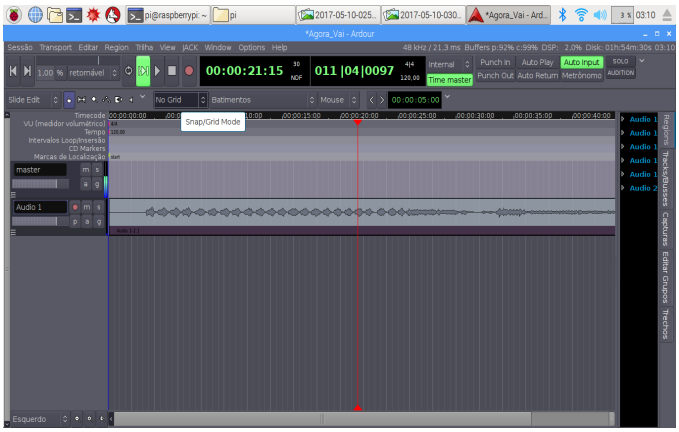

Figura 8. Teste de Gravação

Fonte: extraído diretamente pela Raspberry Pi 3-B

Na figura 8 foi realizado uma gravação de teste, usando o software Ardour implementado na raspberry pi 3-B, cujo foi instalado na placa.

Para a gravação foi utilizado um mini teclado (instrumento) SA-46 da CASIO, a gravação saiu como o esperado e foi realizada com o ganho em 50% tanto do instrumento (OUTPUT) quanto da interface de audio (da INPUT).

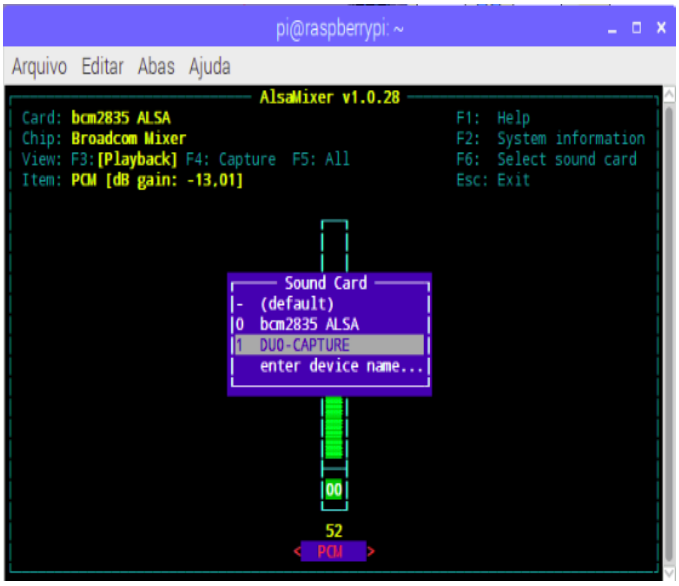

Figura 9. Conecção da interface e o PC

Fonte: extraído diretamente pela Raspberry Pi 3-B

Na figura 9 mostra que a interface UA-11 Duo-Capture da Roland foi reconhecida e selecionada para gravação. Abaixo na figura 10 é possível observar os comandos utilizados para gravação “arecord -D plughw:1,0 test.wav” e "ctrl+c"usado para parar a gravação, posteriormente o comando “aplay test.wav” para ouvir a gravação através do JACKda raspiberry ou da saída de áudio via HDMI.

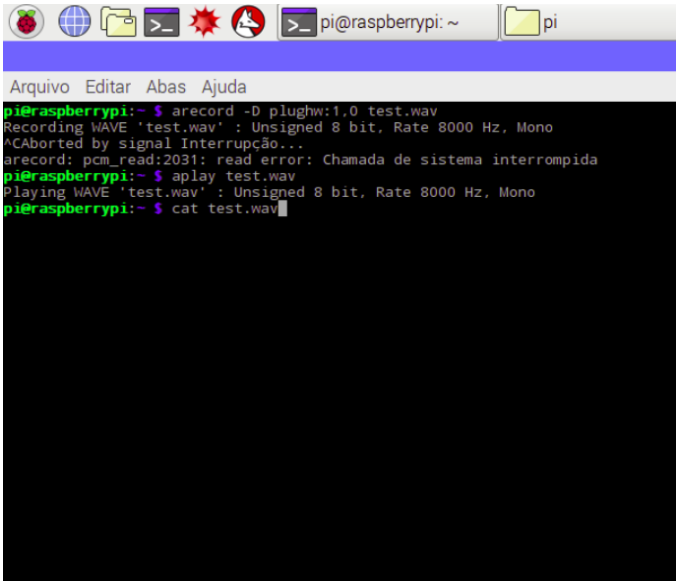

Figura 10. Conecção da interface e o PC

Fonte: extraído diretamente pela Raspberry Pi 3-B

B. Descrição de Software

Implementação do Filtro passa-baixas

Foi realizada uma tentativa de implementar um filtro passa baixa usando média móvel. Para isso abriu-se um arquivo de áudio dentro de um programa em C, copiou-se as amostras deste arquivo e essas amostras foram coladas em um outro arquivo de audio com o filtro implementado. Contudo esse tipo de implementação não é a esperada para o projeto que visa uma tentativa de fazer o processamento do sinal em tempo real. Esse tipo de tentativa foi feita pelo motivo de não conseguir ler as amostras vindas da interface de audio via usb. O código usado para o filtro passa-baixas com médias móveis é da Figura 11:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

        FILE *fp;
        FILE *fp1;
        char c[10];
        int i = 0;
        float media = 0.0;
        fp = fopen("Corvette_pass.wav", "r");
        fp1 = fopen("out.wav", "w");

        if (!fp){ printf("deu ruim"); exit(0);}
        if(!fp1){printf("deu ruim"); exit(1);}
        while((c[i] = fgetc(fp)) != EOF){
                media += c[i];
                i++;
                if(i==10){
                        media = media/10.0;
                        putc(media, fp1);
                        media = 0.0;
                        i = 0;}
                        }


        fclose(fp);
        fclose(fp1);

        return 0;

}
```

Figura 11. Código usado como tentantiva de implementação

O código começa abrindo um arquivo de áudio (.wav), apenas para leitura e outro para escrita. Foi feito a leitura desse arquivo posição por posição e então gravado em um caractere "c". Realizou-se a soma desses caracteres para calcular a média entre eles com 10 caracteres. Essa foi a lógia usada para fazer um filtro com médias móveis. No entanto, o problema apresentado aqui é que se faz necessário gravar primeiro o arquivo de áudio (um sinal de áudio) e depois processar o arquivo. Mas o propósito do nosso projeto é tentar implementar esse processo em tempo real, criando assim um equalizador.

Como uma forma de se comunicar com a interface, e como tudo em Linux funciona com arquivos, ao se olhar para a pasta /dev, se verificou que o arquivo responsável pela interface foi o hidraw0. Contudo esse tipo de arquivo, hidraw, possui um jeito diferente de se trabalhar, pois para ele existe até uma biblioteca própria, o *#include <linux/hidraw>*. Com o código utilizado abaixo, se tentou uma realização de abertura do arquivo e uma escrita e leitura.

```c
/* Linux */
#include <linux/types.h>
#include <linux/input.h>
#include <linux/hidraw.h>

/*
 * Ugly hack to work around failing compilation on systems that don't
 * yet populate new version of hidraw.h to userspace.
 *
 * If you need this, please have your distro update the kernel headers.
 */
#ifndef HIDIOCSFEATURE
#define HIDIOCSFEATURE(len)    _IOC(_IOC_WRITE|_IOC_READ, 'H', 0x06, len)
#define HIDIOCGFEATURE(len)    _IOC(_IOC_WRITE|_IOC_READ, 'H', 0x07, len)
#endif

/* Unix */
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

/* C */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

const char *bus_str(int bus);


int main(int argc, char **argv)
{
        int fd;
        int i, res, desc_size = 0;
        char buf[256];
        struct hidraw_report_descriptor rpt_desc;
        struct hidraw_devinfo info;

        /* Open the Device with non-blocking reads. In real life,
           don't use a hard coded path; use libudev instead. */
        fd = open("/dev/hidraw0", O_RDWR|O_NONBLOCK);

        if (fd < 0) {
                perror("Unable to open device");
                return 1;
        }

        memset(&rpt_desc, 0x0, sizeof(rpt_desc));
        memset(&info, 0x0, sizeof(info));
        memset(buf, 0x0, sizeof(buf));

        /* Get Report Descriptor Size */
        res = ioctl(fd, HIDIOCGRDESCSIZE, &desc_size);
        if (res < 0)
                perror("HIDIOCGRDESCSIZE");
        else
                printf("Report Descriptor Size: %d\n", desc_size);

        /* Get Report Descriptor */
        rpt_desc.size = desc_size;
        res = ioctl(fd, HIDIOCGRDESC, &rpt_desc);
        if (res < 0) {
                perror("HIDIOCGRDESC");
        } else {
```

```
        printf("Report Descriptor:\n");
        for (i = 0; i < rpt_desc.size; i++)
                printf("%hhx ", rpt_desc.value[i]);
        puts("\n");
}

/* Get Raw Name */
res = ioctl(fd, HIDIOCGRAWNAME(256), buf);
if (res < 0)
        perror("HIDIOCGRAWNAME");
else
        printf("Raw Name: %s\n", buf);

/* Get Physical Location */
res = ioctl(fd, HIDIOCGRAWPHYS(256), buf);
if (res < 0)
        perror("HIDIOCGRAWPHYS");
else
        printf("Raw Phys: %s\n", buf);

/* Get Raw Info */
res = ioctl(fd, HIDIOCGRAWINFO, &info);
if (res < 0) {
        perror("HIDIOCGRAWINFO");
} else {
        printf("Raw Info:\n");
        printf("\tbustype: %d (%s)\n",
                info.bustype, bus_str(info.bustype));
        printf("\tvendor: 0x%04hx\n", info.vendor);
        printf("\tproduct: 0x%04hx\n", info.product);
}
```

```
buf[0] = 0x9; /* Report Number */
buf[1] = 0xff;
buf[2] = 0xff;
buf[3] = 0xff;
res = ioctl(fd, HIDIOCSFEATURE(4), buf);
if (res < 0)
        perror("HIDIOCSFEATURE");
else
        printf("ioctl HIDIOCGFEATURE returned: %d\n", res);

/* Get Feature */
buf[0] = 0x9; /* Report Number */
res = ioctl(fd, HIDIOCGFEATURE(256), buf);
if (res < 0) {
        perror("HIDIOCGFEATURE");
} else {
        printf("ioctl HIDIOCGFEATURE returned: %d\n", res);
        printf("Report data (not containing the report number):\n\t");
        for (i = 0; i < res; i++)
                printf("%hhx ", buf[i]);
        puts("\n");
}

/* Send a Report to the Device */
buf[0] = 0x1; /* Report Number */
buf[1] = 0x77;
res = write(fd, buf, 2);
if (res < 0) {
        printf("Error: %d\n", errno);
        perror("write");
} else {
        printf("write() wrote %d bytes\n", res);
}
```

Após essa tentativa, foi usado então essas linhas de comando diretamente do terminal para poder gravar um arquivo de áudio e para ouvir esse arquivo, *"arecord -r 44100 -f cd -t wav -D plughw:1,0 nome_arquivo.wav"*, *"aplay -D plughw:1,0 nome_arquivo.wav"*

```
        /* Get a report from the device */
        res = read(fd, buf, 16);
        if (res < 0) {
                perror("read");
        } else {
                printf("read() read %d bytes:\n\t", res);
                for (i = 0; i < res; i++)
                        printf("%hhx ", buf[i]);
                puts("\n");
        }
        close(fd);
        return 0;
}

const char *
bus_str(int bus)
{
        switch (bus) {
        case BUS_USB:
                return "USB";
                break;
        case BUS_HIL:
                return "HIL";
                break;
        case BUS_BLUETOOTH:
                return "Bluetooth";
                break;
        case BUS_VIRTUAL:
                return "Virtual";
                break;
        default:
                return "Other";
                break;
        }
```

Como é perceptível, foram várias tentativas para fazer com que o projeto funcionasse como o esperado. Mais outra tentaiva foi através de uma biblioteca em python chamada *audiolzay* [1], ela é fruto do trabalho na área de processamento de sinais de audio em tempo real. Conforme a figura 12 é um dos vários códigos comportados feitos pelo autor dessa biblioteca, o mesmo pode ser encontrado no github *https://github.com/danilobellini/audiolazy*.

```
18  Realtime STFT effect to robotize a voice (or anything else)
19
20  This is done by removing (zeroing) the phases, which means a single spectrum
21  block processing function that keeps the magnitudes and removes the phase, a
22  function a.k.a. "abs", the absolute value. The initial zero-phasing isn't
23  needed at all since the phases are going to be removed, so the "before" step
24  can be safely removed.
25  """
26
27  from audiolazy import window, stft, chunks, AudioIO
28  import sys
29
30  wnd = window.hann
31  robotize = stft(abs, size=1024, hop=441, before=None, wnd=wnd, ola_wnd=wnd)
32
33  api = sys.argv[1] if sys.argv[1:] else None
34  chunks.size = 1 if api == "jack" else 16
35  with AudioIO(True, api=api) as pr:
36    pr.play(robotize(pr.record()))
```

Figura 12. Efeito de robotização

Fonte: https://github.com/danilobellini/audiolazy/blob/master/examples/robotize.py

Esse é um simples exemplo de efeito em que foi encontrado no git, é um efeito de robotização. É um efeito interessante

para guitarra e até mesmo usando para modular um sinal de voz. A aplicação de que esses efeitos funcionam foi acompanhada em uma palestra de [1], onde o autor com uma guitarra toca e implementa os efeitos usando a biblioteca *audiolazy* em tempo real durante a palestra.

A ideia era de pegar esses codigos em python(comportados pela biblioteca *audiolazy*), chamá-los em um código em "C"usando a função *sytem()*, processar os dados de audio e apresentar o sinal filtrado na saída da interface de audio.

IV. Resultados

Os resultados não foram alcaçados como os requisitados no início do projeto. Durante o trabalho tentou-se filtrar um sinal utilizando média móvel, porém este não ficou bem audível aos ouvidos humano. Foi encontrado uma biblioteca muito poderosa em python, pode ser vista em [1]. O autor dela fez vários filtros como "fazzer", "wah wah", "distrção", "robotização"para guitarra. Tentou-se implementar a biblioteca com vários códigos escrito em python, mas não tivemos exito, uma vez que ao tentar compilar os programas, alguns módulos não estavam instalados e devido ao pouco conhecimento de python dos autores deste trabalho e ao tempo para tal aprendizado, não foi possível a implementação dos filtros nem em tempo real (por causa do drive da interface de audio) e também sem ser em tempo real.

Com o uso do programa para tentar ler o arquivo hidraw, não foi possível fazer a leitura e escrita nesse arquivo. Isso pode ter ocorrido em virtude de uma não compatibilidade com a interface de áudio. Com as linhas de comando para apenas gravar e executar uma arquivo de áudio, com a quantidade certa de bits, 16 bits, e com uma taxa de amostragem de 44100Hz, garantindo uma boa qualidade no som do arquivo.

V. Conclusão

Ao longo do desenvolvimento do trabalho, foi visto que mesmo com todas as dificuldades que se encontrou ao longo do caminho, certamente não faltava muito para conseguir implementar o nosso equalizador de áudio ou conseguir fazer a leitura do arquivo pertencente a interface. Essa dificuldade pode ter vindo justamente do modelo da interface usada, que poderia não ter uma plena compatibilidade com os sistema Linux. É notório a complexidade deste projeto, é uma ideia a ser abordada em nível de um trabalho de conclusão de curso ou mestrado. Mesmo com toda essa complexidade e não concretização do objetivo em foco ter funcionado, foi de grande valia o estudo sobre este tema e o aprendizado ao longo da disciplina.

Referências

[1] Danilo J. S. Bellini. Real-time expressive digital signal processing (dsp) package for python, 2017. https://pypi.python.org/pypi/audiolazy/0.6.

[2] Henrique Bartoski Ferreira. Processamento de sinais digitais de áudio no raspberry pi e beaglebone black. B.S. thesis, Universidade Tecnológica Federal do Paraná, 2014.

[3] Christian Gonçalves Herrera. *Projeto de Sistemas de Processamento Digital de Sinais de Áudio Utilizando Metodologia Científica*. PhD thesis, Universidade Federal de Minas Gerais, 2004.

[4] Carlos Alexandre Mello. Processamento digital de sinais. *Centro de Informática UFPE. Disponível em< http://www. cin. ufpe. br/˜ cabm/pds/PDS. pdf>. Acesso em*, 30(09), 2013.