UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE GRADUAÇÃO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

HENRIQUE BARTOSKI FERREIRA

PROCESSAMENTO DE SINAIS DIGITAIS DE ÁUDIO NO RASPBERRY PI E BEAGLEBONE BLACK

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO - PR

2014

HENRIQUE BARTOSKI FERREIRA

PROCESSAMENTO DE SINAIS DIGITAIS DE ÁUDIO NO RASPBERRY PI E BEAGLEBONE BLACK

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Tecnólogo em Tecnologia em Sistemas para Internet.

Orientador: Prof. Me. Ivanilton Polato

Co-orientador: Prof. Me. Paulo C. Gonçalves

CAMPO MOURÃO - PR

2014

ATA DA DEFESA DO TABALHO DE CONCLUSÃO DE CURSO

Às **vinte e duas e dez horas** do dia **vinte e sete de agosto de dois mil e quatorze** foi realizada no Mini-auditório do EAD da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso Superior de Tecnologia em Sistemas para Internet do acadêmico **Henrique Bartoski Ferreira** com o título **Processamento de sinais digitais usando dispositivos de baixo custo**. Estavam presentes, além do acadêmico, os membros da banca examinadora composta pelo professor **Me. Ivanilton Polato** (Orientador-Presidente), pelo professor **Me. Paulo Gonçalves,** pelo professor **Me. Paulo Sabo** e pelo professor **Me. Filipe Roseiro Côgo**. Inicialmente, o aluno fez a apresentação do seu trabalho, sendo, em seguida, arguido pela banca examinadora. Após as arguições, sem a presença do acadêmico, a banca examinadora o considerou **APROVADO** na disciplina de Trabalho de Conclusão de Curso e atribuiu, em consenso, a nota ____ (_____). Este resultado foi comunicado ao acadêmico e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações:

_____

_____

_____

_____

_____

Campo Mourão, 27 de agosto de 2014.

Prof. Me. Paulo César Gonçalves Membro	Prof. Me. Paulo Sabo Membro	Prof. Me. Filipe Roseiro Côgo Membro	Prof. Me. Ivanilton Polato Orientador

A Folha de Aprovação assinada encontra-se na Coordenação do Curso

RESUMO

FERREIRA, Henrique Bartoski. PROCESSAMENTO DE SINAIS DIGITAIS DE ÁUDIO NO RASPBERRY PI E BEAGLEBONE BLACK. 55 f. Trabalho de Conclusão de Curso – Curso de Graduação de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2014.

O processamento de sinais digitais traz diversas vantagens e oportunidades de expansão e criação de ferramentas úteis na produção e manipulação de áudio. Nesse contexto, os dispositivos computacionais tornaram-se peças fundamentais em estúdios de produção de áudio ou em apresentações artísticas ao vivo. Porém, equipamentos e softwares específicos para esse tipo de processamento possuem um alto custo e limitações de adaptações. Este trabalho explora e analisa alternativas para o processamento de sinais digitais utilizando dois dispositivos computacionais de baixo custo: o Raspberry Pi e o BeagleBone Black. Aliados ao software Pure Data, implementamos um ambiente de processamento de sinais digitais nesses dispositivos para o processamento de áudio em tempo real, ou pelo menos, próximo de tempo real. Por fim, realizamos uma análise das capacidades e limitações de cada dispositivo, comparando suas características com outras plataformas comuns de processamento de sinais digitais, sendo possível chegar a conclusão que os dispositivos estudados podem ter um bom desempenho no processamento de sinais digitais de áudio em tempo real.

Palavras-chave: sinal digital, processamento, RaspberryPi, BeagleBone, audio, filtros, Pure Data

ABSTRACT

FERREIRA, Henrique Bartoski. DIGITAL PROCESSING OF AUDIO SIGNALS IN RASPBERRY PI AND BEAGLEBONE BLACK. 55 f. Trabalho de Conclusão de Curso – Curso de Graduação de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2014.

Digital signal processing has many advantages and opportunities for expansion and creation of useful tools in the audio production and manipulation. Thus, computing devices have become essential in audio production studios or live artistic performances. However, specific equipment and software for this processing are quite expensive. Our work aims to explore and analyze alternatives for digital signal processing using two low-cost computing devices: the Raspberry Pi and the BeagleBone Black. Using the Pure Data software, we implemented algorithms and techniques to process digital signals in (near) real-time. Finally, we performed an analysis and reviewed the capabilities and limitations of each device, comparing their characteristics with other common platforms used in digital signals processing.

Keywords: digital signal, processing, RaspberryPi, BeagleBone, audio, digital filters, Pure Data

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O uso de novas tecnologias é cada vez mais comum durante o processo de produção de áudio. A cada dia surgem novos equipamentos e dispositivos que auxiliam no processo produtivo musical. Com a popularização dos estúdios domésticos de gravação de áudio, o computador se tornou um dos principais elementos nesses ambientes, onde muitas vezes substitui equipamentos de alto custo. Da mesma forma, é comum o uso de computadores para processamento de sinais digitais em eventos e apresentações ao vivo. Em decorrência desses fatos, estudos na área de processamento de sinais digitais (DSP - Digital Signal Processing), especialmente na área de processamento de áudio, são de grande interesse para técnicos e para profissionais do meio musical.

Até a década 1960, a tecnologia para o processamento de sinais era quase exclusivamente analógica. A rápida evolução dos computadores e microprocessadores digitais juntamente com alguns desenvolvimentos teóricos importantes, tais como o algoritmo de transformada rápida de Fourier (FFT - Fast Fourier Transform), causaram uma série de mudanças para a tecnologia digital, dando origem à área de processamento de sinais digitais (OPPENHEIM et al., 1999). Estudos nesse segmento possibilitam criar novas formas de interação entre o artista e sua apresentação, aumentar a gama de recursos sonoros durante a produção e, principalmente, encontrar alternativas para diminuir o custo de equipamentos.

São muitas as possibilidades de uso de tecnologia na produção sonora, como por exemplo, a expansão e o desenvolvimento de novos instrumentos, a simulação de instrumentos e a alteração de características e/ou criação de novos sons(OPPENHEIM et al., 1999). Dentro da área de processamento de áudio, uma das áreas de ampla exploração é a aplicação de filtros sobre o sinal original. Filtros digitais são algoritmos que possuem a capacidade de alterar características, como frequências e amplitudes, de um determinado sinal previamente convertido ao meio digital (EMBREE, 1995). O conceito de filtros digitais pode ser exemplificado estudando as características dos filtros conhecidos como Passa-baixa (Low Pass) e Passa-alta (Hi Pass). Esses filtros possuem a capacidade de atenuar a amplitude das frequências acima ou abaixo de uma frequência de corte determinada. Filtros digitais podem ser manipulados uti-

lizando softwares específicos para processamento de sinais digitais, como o Pure Data[1]. Essa plataforma de código aberto possibilita que pessoas sem um grande conhecimento na área computacional consigam desenvolver algoritmos para o tratamento de sinais digitais, usando uma interface gráfica. Nas próximas sessões serão apresentadas possibilidades de uso de softwares como o Pure Data aliado a dispositivos computacionais de baixo custo, como o Raspberry Pi[2] e BeagleBone[3], para o processamento de sinais digitais.

1.1 MOTIVAÇÃO

Para ampliar a interação de artistas com novas tecnologias, bem como aumentar a gama de recursos para a produção e manipulação de elementos sonoros musicais, é de grande interesse encontrar dispositivos e tecnologias que possuam usabilidade, isto é, que não exijam grande esforço para o aprendizado de uso, que sejam de fácil aquisição e, principalmente, acessíveis em relação ao custo. Visto que muitos dispositivos voltados para o processamento de sinais digitais no segmento de áudio possuem um custo elevado para para grande parte dos interessados (ANHAIA, 2013), a exploração de possibilidades em tecnologias populares e/ou de tecnologias de baixo custo pode ser uma opção viável.

Nessa abordagem, dispositivos de plataforma e código aberto são grandes atrativos a pesquisas relacionadas ao processamento de sinais digitais. Dispositivos como Raspberry Pi, dispositivos da familia BeagleBoard[4], Arduino[5], Unidades de Processamento Gráfico (GPU - Graphics Processing Unit), plataformas com o sistema operacional Android[6], entre outros equipamentos, aliados a softwares voltados para a manipulação de sinais digitais como o Pure Data e CSound[7] vêm sendo estudados para o processamento de sinais digitais. Nesse estudo serão levantadas e estudadas características de dispositivos criados com o intuito de serem computadores de baixo custo, como o Raspberry Pi e o BeagleBone.

Para a escolha dessas plataformas para o estudo, foram levados em conta alguns fatores como por exemplo, o baixo custo de aquisição do equipamento, a sua capacidade computacional e facilidade de uso. Neste trabalho, a principal justificativa para o uso dos dispositivos escolhidos como alternativas ao uso de dispositivos específicos para DSP é a relação custo/beneficio que os dispositivos oferecem. Levando em conta as licenças de utilização e a versatilidade de

[1] http://PureData.info
[2] http://www.raspberrypi.org
[3] http://beagleboard.org/Products/BeagleBone
[4] http://beagleboard.org/
[5] http://www.arduino.cc
[6] http://www.android.com
[7] http://www.csounds.com

aplicações de uso dos dispositivos, é possível que os mesmos sejam adaptados para processamento de áudio, ao contrário dos equipamentos específicos para tal ação, que foram projetados especificamente para processamento de sinais digitais. Levando-se em consideração a necessidade de poder computacional para manipular dados contínuos de áudio, é esperado um tempo de latência razoavelmente pequeno para considerarmos que esse processamento está ocorrendo próximo do tempo real. Embora o hardware dos dispositivos escolhidos seja limitado em relação a capacidade computacional, as características das plataformas, como preços de aquisição e portabilidade, podem, dentro de seus limites e em casos específicos, popularizar para o uso dos mesmos em processamento de sinais digitais de áudio.

1.2 OBJETIVOS

O presente trabalho tem o foco em encontrar um equilíbrio entre o fator latência e qualidade sonora nos dispositivos computacionais Raspberry Pi e BeagleBone, analisando e comparando seus limites e possibilidades.

Nossa hipótese é que essas plataformas possuem características desejáveis para nossa pesquisa e que podem apresentar resultados positivos, sendo alternativas relativamente baratas à equipamentos profissionais disponíveis no mercado.

1.3 OBJETIVOS ESPECÍFICOS

Visando atingir o objetivo principal, são nossos objetivos específicos:

- Preparar o conjunto software/hardware nos dispositivos estudados, instalando sistemas operacionais indicados pelos fabricantes dos dispositivos, servidores de áudio específicos para baixa latência, como o Jack (Jack Áudio Connection Kit)[8], e aplicativos específicos para a manipulação de sinais digitais, como o Pure Data.
- Implementar filtros e componentes de tratamento de áudio de diferentes níveis de custo computacional com o Pure Data.
- Executar os algoritmos implementados nos dispositivos estudados, utilizando algumas formas de entrada de sinais digitais e modificando parâmetros de configurações da entrada do sinal para encontrar situações onde seja possível obter uma menor latência.

[8] `http://jackaudio.org/`

- Analisar a latência gerada no processamento dos algoritmos implementados, comparando os resultados dos dispositivos testados.
- Obter um comparativo entre a capacidade de processamento de sinais digitais de áudio entre o Raspberry Pi e o BeableBone Black, bem como, compara-las com plataformas computacionais comuns de uso geral, ou seja, computadores pessoais usados em atividades comuns, com uma capacidade computacional mediana ao encontrada no mercado.

1.4 PRINCIPAIS RESULTADOS

Como citadas nas motivações, diversas são as possibilidades de exploração de novas tecnologias para o processamento de sinais digitais. Este trabalho tem como objetivo analisar através de medições de latência e análise de espectrogramas, o processamento de áudio digital nos dispositivos Raspberry Pi e BeagleBone Black que, como detalhado nas próximas seções, apresentam um desempenho satisfatório para o processamento de sinais de áudio em tempo real.

2 REVISÃO BIBLIOGRÁFICA

Nas próximas sessões serão revisados assuntos relevantes à fundamentação teórica do estudo apresentado nesse trabalho.

2.1 SINAIS DIGITAIS

Sinais pode ser definidos como funções que carregam informações (MELLO, 2013), e estão amplamente presentes em nosso cotidiano. Exemplos de sinais são música, imagens, vídeos e a fala (SILVA, 2007). O som é um sinal, gerado por uma perturbação da matéria que leva energia de um lugar para outro em forma de onda. Quando as cordas vocais vibram, elas criam uma compressão que empurra as moléculas de ar, e durante o intervalo de tempo em que as cordas vocais voltam, elas criam rarefações no ar. Esses eventos ocorrendo repetidamente caracterizam uma onda longitudinal, que se caracteriza por seus pontos se moverem em direção paralela à própria onda, possibilitando a propagação do som (PIERCE, 1989).

Após ser gerado, o sinal é interpretado por um receptor. No caso do som, o receptor pode ser o ouvido humano, que é sensível as vibrações sonoras que se propagam pelo ar, e capaz de converter tais sinais em uma forma interpretável ao cérebro humano. Da mesma forma, ao se tratar um sinal no meio digital, devemos usar um meio de transformar o sinal em uma forma interpretável ao meio digital, como por exemplo um microfone, que converte as vibrações sonoras em pulsos elétricos que posteriormente, são convertidos ao meio digital.

Um sinal analógico pode ser convertido para o meio digital através da combinação de três operações: amostragem, quantização e codificação (HAYKIN, 1988). A representação dos sinais analógicos de áudio no meio digital é obtida através de um conversor analógico-digital (ADC - *Analog-to-Digital Converter*), dispositivo eletrônico capaz de gerar uma representação digital a partir de uma grandeza analógica, analisando amostras das amplitudes do sinal analógico ao longo de um tempo predeterminado. Essas amplitudes são distribuídas sobre uma linha que representa o tempo da amostra ao longo de um eixo horizontal, e quantifica as amplitudes das amostras fixas representando-as por números ao longo do eixo de vertical (ZÖLZER,

2011), transformando um sinal contínuo no tempo em um sinal discreto no tempo, esse processo é chamado de amostragem.

Segundo o teorema de Nyquist, um sinal limitado em uma faixa em B Hz (amostras por segundo) pode ser reconstruído exatamente a partir de suas amostras tomadas a uma taxa maior ou igual à 2 vezes B Hz. Caso a frequência de amostragem seja menor que a frequência obtida a partir do teorema de Nyquist, podem correr perdas na qualidade do sinal, visto que o sinal poderá não ser recuperado completamente, causando distorções nas frequências mais altas.

Na fase de quantização, cada valor de amostra é aproximado pelo nível mais próximo dentre um conjunto finito de níveis discretos. Já na etapa de codificação, cada nível de quantização é representado por um código que consiste de uma sequencia de dígitos binários (MADEIRO, 2011).

Da mesma forma, os sinais representados no meio digital podem ser novamente convertidos ao meio analógico através de um conversor digital-analógico (DAC – Digital-to-Analog Converter) (ZÖLZER, 2011), este processo está descrito na Figura 1.

Figura 1: Representação da conversão analógico/digital e digital/analógoco de um sinal de áudio

2.2 PROCESSAMENTO DE SINAIS DIGITAIS

Processadores de sinais digitais realizam a manipulação matemática de informações digitais, com o intuito de modificar o sinal. A evolução dos circuitos projetados para o processamento de sinais digitais se deu pela inclusão de características que facilitam a computação de algoritmos de processamento de sinais digitais, como por exemplo, a inclusão do produto interno como operacional básica, permitindo a computação de filtros de resposta impulsiva infinita de forma mais rápida (BIANCHI, 2011), tais tecnologias ampliaram a área de atuação dos processadores de sinais digitais.

Apesar do processamento de sinais no meio analógico e digital terem características distintas e próprias, sendo muito discutido por profissionais do áudio as vantagens de cada meio,

o uso de processadores de sinais digitais se destaca em alguns pontos sobre os processadores de sinais analógicos. Os sistemas digitais de processamento de sinais sofrem menos alterações por condições externas, como temperatura e desgaste dos componentes eletrônicos do equipamento. Além disso, os sistemas DSP são bastante versáteis, pois os algoritmos de processamento não estão vinculados aos componentes eletrônicos, podendo ser modificados com maior facilidade.

Processamento de sinais digitais são usados em diversas aplicações, como processamento de áudio, processamento digital de imagens, redes de comunicação digital, radares, sonares, sismologia, controle de processos industriais, biomedicina (RABINER; GOLD, 1975; OPPENHEIM et al., 1999).

Além dos algoritmos DSP de propósitos específicos, são encontrados programas computacionais voltados a processamento de sinais digitas de propósito geral, que possuem capacidade de processar sinais digitais em tempo real, como o Pure Data (PUCKETTE, 1996).

2.3 INTERAÇÃO ENTRE COMPUTADOR E DISPOSITIVOS DE ENTRADA E SAÍDA

Os sistemas operacionais de uso geral utilizam *buffers* de dados para a comunicação com dispositivos de E/S (Entrada e Saída). Apesar de ser vantajoso em termos de utilização de recursos da máquina, o uso de *buffers* não é adequado para aplicações onde a baixa latência é desejada, pois o uso dos mesmo acrescenta tempo ao processamento. A necessidade de uso de *buffers* é decorrente do mecanismo projetado de E/S, pois ao receber uma requisição de interrupção o processador abandona o processo concorrente para realizar as operações necessárias para a transferência de dados entre o computador e o dispositivo de entrada e saída de dados. Após a conclusão, o processador necessita dos dados do processo interrompido para continuar, recorrendo aos dados nos *buffers*. Dessa forma, alternativas, que serão citadas a diante neste trabalho, devem ser buscadas para reduzir o tempo consumido pelo uso de *buffers* durante o processamento de sinais em tempo real (LAGO, 2004).

As placas de som normalmente também possuem *buffers* de dados internos, usados para o armazenamento do áudio capturado e para o armazenamento do áudio a ser reproduzido. Em uma representação ideal, a menor latência possível pode ser obtida quando os dados são copiados diretamente da entrada para a saída, com isso o tempo corresponderia à soma das durações correspondentes ao tamanho do *buffer* de entrada e saída. Levando-se em conta que normalmente o tamanho dos *buffers* usados pelas placas de áudio podem ser configurados via software, é possível definir (dentro dos limites e levando-se em conta os prejuízos ao sinal) a latência de um sistema .

Por outro lado, softwares como ASIO (*Audio Stream Input/ Output*) e JACK, procuram garantir que aplicações que se comuniquem com o hardware de áudio obtenham melhores resultados em sistemas operacionais de uso geral. Para isso, esses softwares imitam mecanismos de funções *callback* responsáveis pelo tratamento de interrupções, abstraindo o mecanismo para o nível das aplicações do usuário, permitindo o tratamento de interrupções por qualquer aplicativo compatível com esses sistemas, sem a necessidade de alterações no código no núcleo (LAGO, 2004).

O sistema de gerenciamento de E/S e processamento de áudio baseado em funções *callback* mais utilizado na industria de sistemas para áudio profissional é o ASIO, compatível com os sistemas Windows e MacOS. Em sistemas Linux, o sistema mais usado é o JACK, que utiliza um conjunto de controladores de baixo nível, denominado ALSA (*Advanced Linux Sound Architecture*) (TORRINI; SHKOLNYY, 2014), para intermediar as comunicações entre aplicações e dispositivos de entrada e saída de áudio. O JACK, oferece uma interface de mais alto nível que os padrões ALSA, possibilitando a edição e configuração de parâmetros de entrada e saída dos dispositivos de E/S e estabelece mecanismos transparentes para a comunicação entre aplicações de áudio, possibilitando interligar diversas aplicações entre si e com a placa de som.

O JACK possui algumas características de maior importância, entre elas são:

- Escalonamento de tempo real: Permite a uma aplicação ser agendada para execução assim que esteja pronta, e é suspensa apenas se for bloqueada ou se outra aplicação de maior prioridade estiver pronta para a execução.
- *Patches* para baixa latência: Diversos *patches* de baixa latência para o núcleo do Linux são capazes de trazer o tempo de resposta do sistema operacional para valores entre um e dois milissegundos, reduzindo o tempo entre as operações necessárias para o sistema executar uma aplicação agendada.

Com as características e funções disponibilizadas pelo JACK, é possível desenvolver aplicações de áudio em tempo real com taxas de latência em torno de 3 a 6ms, executadas em um sistema utilizando *patches* de baixa latência e com a placa de áudio configurada para usar *buffers* de dados pequenos (MACMILLAN et al., 2001), o que pode auxiliar no melhor desempenho dos dispositivos no processamento de sinais digitais de áudio em tempo real.

2.4 FILTROS DIGITAIS

Pode se comparar o conceito de filtros digitais a uma peneira, que consegue selecionar determinados elementos com propriedades desejadas, a partir de um conjunto maior (ZÖLZER, 2011). No caso de sinais digitais de áudio, filtros podem permitir que componentes desejados da entrada de frequência cheguem inalterados à saída, enquanto outros componentes sejam bloqueados ou alterados (EMBREE, 1995).

Os filtros chamados de Passa-baixa e Passa-alta, descritos a seguir, são exemplos comuns de filtros de sinais, pois exemplificam de forma clara e simples seu funcionamento.

O filtro Passa-baixa remove ou atenua todas as frequências acima de uma frequência determinada em uma amostra, sem prejudicar ou atenuar as frequências abaixo ao valor de corte (FURUTANI et al., 1997). O Passa-baixa pode ser comparado a uma parede sólida: quando uma pessoa fala em um quarto fechado, sua voz perde as frequências mais altas ao ser ouvida de fora do quarto, pois a parede as atenua. Os filtros Passa-baixa são usados comumente na produção musical, em equipamentos de amplificação de áudio para diminuir a distorção, em transmissores FM para remover frequências que podem atrapalhar outras transmissões, e em sensores industriais (SOUSA, 2008).

O filtro Passa-alta atua de maneira bastante semelhante ao Passa-baixa, porém com um princípio invertido de funcionamento. O Passa-alta atenua ou remove as frequências abaixo da frequência de corte, deixando passar apenas as frequências mais altas que uma frequência de corte determinada (BEAMISH et al., 2007).

Para seu uso, segundo Bianchi (2011), os filtros digitais são implementados em algoritmos que podem ser classificados de acordo com o seu custo computacional.

Algoritmos de baixo custo computacional:

- Filtros básicos de suavização (realce de graves) ou de diferenças (realce de agudos) (BIANCHI, 2011);
- Efeitos simples como overdrive, phaser, wah-wah (ZÖLZER, 2011);
- Equalização em contexto geral usando pólos e zeros como descritores de regiões de ressonância e anti-ressonância em filtros (BIANCHI, 2011);
- Detecção de características sonoras de baixo nível, como frequência fundamental, energia rms, centróide espectral ou MFCC (*Mel-frequency cepstral coefficients*), ou descritores

psicoacústicos, como brilho, harmonicidade ou ruidosidade (PEETERS, 2004; BIANCHI, 2011);

Algoritmos de médio custo computacional:

- Efeitos como pitch shifting, flanger, companders, vibrato e chorus (ZÖLZER, 2011);
- Reverberação/espacialização genérica (ZÖLZER, 2011);

Algoritmos de alto custo computacional:

- Morphing em tempo real (aplicação da envoltória espectral de um sinal a outro sinal) (ZÖLZER, 2011);
- Phase Vocoder (com análise/ressíntese em tempo real) (ZÖLZER, 2011);
- Auralização usando respostas impulsivas medidas ou simulação através de modelos geométricos (MOORE, 1990).

Na Sessão 3 serão descritos os filtros utilizados nos testes, bem como suas características.

2.5 PURE DATA

O Pure Data (PD) é um ambiente de programação genérico, baseada no paradigma de programação gráfica MAX (PUCKETTE, 1998) e dedicado à síntese e processamento de áudio e vídeo em tempo real (PUCKETTE, 1996). O projeto foi originalmente desenvolvido por Miller Puckette e ao usar uma licença de software livre ganhou popularidade. Atualmente conta com uma grande quantidade de desenvolvedores trabalhando em extensões para o ambiente.

A ferramenta pode ser usada para processar e gerar som, vídeo, gráficos 2D/3D, manipular dados de sensores, de interfaces de entrada e MIDI (*Musical Instrument Digital Interface*), além de poder trabalhar facilmente em redes locais ou remotas, integrando tecnologias como sistemas de motores, sensores, equipamentos de iluminação, entre outros.

O Pd permite o desenvolvimento de algoritmos através da linguagem de programação C e/ou de uma interface visual. No desenvolvimento de algoritmos realizado a partir da interface visual, a plataforma oferece objetos com tarefas especificas em forma de caixas, que podem ser ligados através de linhas, abstraindo as manipulações matemáticas de baixo nível dos

métodos. Tal característica possibilita a utilização da ferramenta por usuários que não possuam amplo conhecimento em computação, permitindo seu uso por músicos, artistas e pesquisadores (PDCOMMUNITY, 2013).

Segundo o site oficial, o Pure Data pode ser encontrado em várias versões. As principais são:

- **Pd vanilla** (ou simplesmente Pd): é o núcleo do Pure Data, escrito principalmente por Miller Puckette, com foco em processamento de sinais de áudio e MIDI.
- **Pd extended**: possui muitas bibliotecas escritas pela comunidade. O Pd extended pode ser utilizado para renderização de gráficos, comunicações OSC (*Open Sound Control*), processamento de arquivos binários, *streaming* de áudio-visual, modelagem física, apresentações baseadas em sensores, entre outros.

Com uma ampla quantidade de usuários e com compatibilidade com múltiplos sistemas operacionais como Linux, Windows e MacOS, e também com dispositivos móveis. O sistema permite a construção e execução em diversos dispositivos, uma ampla variedade de algoritmos de síntese sonora, desde de MIDI simples, até a geração de som interativo, sendo ideal para ser usado como uma extensão de síntese sonora em geral (ALONSO et al., 2004), permitindo a expansão de dispositivos e formas de criação, interação e manipulação de sons.

As características apresentadas pelo Pure Data torna-o de grande utilidade como ambiente de processamento de sinais digitais, a união com os dispositivos computacionais aqui estudados tem grande potencial para criar novas formas de manipulação de áudio.

2.6 RASPBERRY PI

O Raspberry Pi é um computador que possui todo o hardware integrado em uma única placa do tamanho de um cartão de crédito. O dispositivo foi desenvolvido por uma organização inglesa sem fins lucrativos, chamada Raspberry Pi Foundation (Fundação Rasbperry Pi), com o objetivo de estimular o ensino de ciência da computação básica em escolas (ELINUX, 2013b) e ganhou o título de computador mais barato do mercado.

Existem três modelos do Raspberry Pi, o Modelo A, o Modelo B, e o B+ como especificado na tabela 1. A grande diferença entre os três modelos é que os Modelos B possui 512MB de memória Ram, um controlador Ethernet e duas (no modelo B) e 4 (no modelo B+) portas USB, enquanto o Modelo A possui apenas 256MB de memória Ram, uma porta USB e não possui porta de Ethernet (RASPBERRYPI.ORG, 2013).

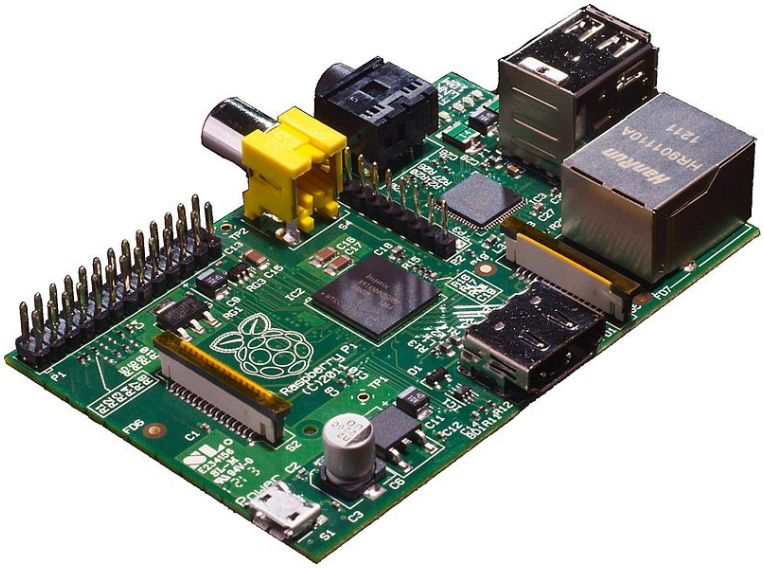


Figura 2: Raspberry Pi modelo B

	Model A	Model B/B+
Preço estimado	US$25	US$35
System-on-a-chip	Broadcom BCM2835	
CPU	700MHz ARM11 ARM1176JZF-Single core	
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, OpenVG 1080p30 H.264 high-profile encode/decode	
Memoria (SDRAM)	256 MiB (planejado com 128 MiB, atualizado para 256 MiB em 29 Fev 2012	512 MiB (desde out 15, 2012)
Portas USB 2.0	1 (BCM2835)	2 ou 4 (hub USB integrado)
Saídas de Vídeo	RCA, HDMI	
Saída de Áudio	TRS conector, jack 3,5 mm, HDM	
Entradas de áudio	nenhuma; microfone USB ou placa de som pode ser adicionada;	
Armazenamento	SD / MMC / SDIO	
Periféricos de baixo nível	General Purpose Input/Output (GPIO), Interface Periférica Serial Bus (SPI), I2C, UART)	
Sistema Operacionais recomentados	Raspbian, Pidora, RISC OS, OpenELEC, Arco, RaspBMC	
Fonte de alimentação	5 V (DC) via Micro USB tipo B ou GPIO	
Dimensões	85 x 56 x 15mm	85 x 56 x 17mm
Peso	31g	40g

Tabela 1: Especificações Técnicas do Raspberry Pi
Fonte: (ELINUX, 2013a)

2.7 BEAGLEBONE

O BeagleBone é um computador de baixo custo, com valores de US$89 em sua primeira versão e US$45 na versão Black (BEAGLEBOARD, 2013). O BeagleBone faz parte da linha BeagleBoard que possuem hardware 100% open-source, desenvolvidos pela Texas Instruments. O BeagleBone é pequena mesmo para os padrões BeagleBoard, mas proporciona muito do desempenho e capacidades dos BeagleBoards de maior porte. O dispositivo conta com um

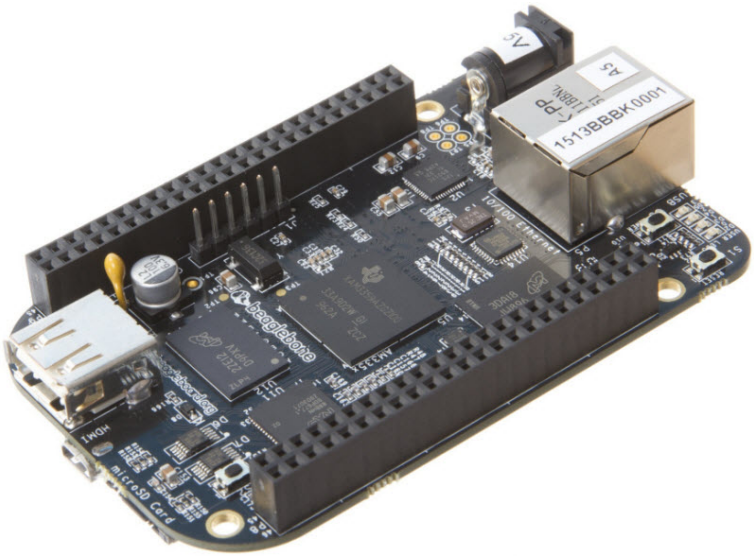


Figura 3: BeagleBone Black

cartão micro-SD de 4 GB pré-carregado com a distribuição Linux Angstrom ARM, um acelerador gráfico 3D POWERVR SGX e uma unidade de processamento programável de tempo real (PRU-ICSS), possibilitando desenvolver separadamente do core principal, aplicações e protocolos de tempo real.

O BeagleBone possui duas versões: a original e a versão Black. Ambas são baseadas no TI Sitara AM335x, um processador SoC que contém um ARM Cortex-A8 como núcleo. Apesar do preço menor, a versão Black possui algumas melhorias no hardware em relação da primeira versão (ELINUX, 2013a), como descrito na tabela 2.7:

	BeagleBone Black	**BeagleBone**
Processador	AM3359AZCZ100, 1GHz	AM3358ZCZ72, 720MHz
Saída de Video	HDMI	HDMI
Saída de áudio	HDMI	HDMI
Memória DRAM	512MD DDR3L	256MB DDR2
Armazenamento Flash	2GB eMMC, uSD	uSD
Onboard JTAG	Opcional	Via USB
Portas Serial	Header	Via USB
Alimentação	210-460 mA@5v	300-500 mA@5V
Preço sugerido	$45	$89

Tabela 2: Diferenças entre BegleBone e BeagleBone Black
Fonte: (DIGIKEY, 2013)

2.8 DSP EM DISPOSITIVOS DE BAIXO CUSTO

Com a difusão das tecnologias citadas e da oferta equipamentos computacionais de uso geral com grande capacidade de processamento o uso desses equipamentos vem se popularizando para o processamento de sinais digitais (LAGO, 2004). Com isso, o custo de criação e manutenção de estúdios domésticos sofreu uma grande redução, mantendo uma qualidade competitiva com os estúdios convencionais (OPPENHEIMER, 1999). Esses fatos levaram ao desenvolvimento de algumas pesquisas relacionadas a equipamentos de uso geral para a finalidade de processar sinais digitais, inclusive pesquisas com equipamentos de características similares as do Raspberry Pi e do BeagleBone.

2.8.1 ARDUINO

Um estudo apresentado na International Conference on New Interfaces for Musical Expression em 2011, trata da utilização do Arduino como placa de som, através da implementação de um driver utilizando a infraestrutura do projeto ALSA (DIMITROV; SERAFIN, 2011), porém a resolução dos sinais captados e emitidos é baixa, visto que o Arduino trabalha com apenas 8 bits contra 32 das placas convencionais (BIANCHI, 2011).

Projetos envolvendo o Arduino para a produção musical são populares na Web, entre eles está o projeto de uma Harpa a Laser usando o Arduino como controlador para produzir sinais MIDI (WOODRUFF; GÖRMEZ, 2012). O projeto é composto de uma série de lasers

e foto-sensores. O sistema detecta a posição da mão do utilizador quando ela cruza um ou mais feixes de laser. Os feixes de laser se tornam visíveis com uma pequena quantidade de fumaça em um ambiente escuro. Quando o feixe é quebrado o sistema detecta e gera um sinal MIDI que é, posteriormente, interpretado por outro dispositivo para a geração do áudio. Apesar de ter objetivos e características distintas do trabalho aqui proposto, tal projeto mostra como equipamento de custo relativamente baixo podem trazer novas formas de interação no processo de produção de áudio.

Outras implementações mostram a capacidade do Arduino para DSP. Um exemplo é um pedal de efeitos Lo-Fi [1] para guitarra [2], feito com a plataforma Arduino, unido a um conjunto de componentes eletrônicos externos. Apesar da baixa resolução do áudio, o equipamento consegue aplicar filtros simples sobre os sinais inseridos, mostrando que mesmo um dispositivo com baixa capacidade de processamento pode ser usado para processamento de sinais digitais.

2.8.2 RASPBERRY PI

Apesar da similaridade em relação à facilidade de uso com os computadores de uso geral e de suas capacidades computacionais, o Raspberry Pi ainda não possui grandes estudos publicados em relação à sua capacidade de processamento de sinais digitais de áudio. Apesar disso, o Raspberry Pi vem sendo usado em alguns projetos envolvendo processamento de sinais de áudio, já sendo usado até mesmo em apresentações ao vivo e em estúdios musicais.

Um dos projetos de maior relevância para trabalho trata do uso do Raspberry Pi como um processador de efeitos para guitarras. O projeto, desenvolvido por Pierre Massat (MASSAT, 2013), utiliza uma interface de áudio externa ligada via USB ao Raspberry Pi, e trata os sinais inseridos com o Pure Data. Pierre relata que foram realizados diversos testes para que fosse possível obter uma latência aceitável, foram realizadas modificações e configurações no sistema operacional e nas aplicações utilizadas para a manipulação do áudio (como Pd e JACK), permitindo que o tempo de processamento não fosse percebido ao se tocar a guitarra. Em relação à aplicação dos efeitos no áudio, foram desenvolvidos diversos filtros em Pd. Pierre também desenvolveu um pedal controlador utilizando um Arduino, possibilitando a alteração dos efeitos e a manipulação dos parâmetros de configuração dos filtros usados na aplicação de efeitos no áudio.

[1] Lo-Fi é um estilo de produção musical que usa técnicas de gravação de baixa fidelidade

[2] `http://www.instructables.com/id/Lo-fi-Arduino-Guitar-Pedal/`

2.8.3 BEAGLEBOARDS

Apesar de possuírem algumas características distintas, os dispositivos da família BeagleBoard se destacam pelo baixo preço e por possuírem alguns projetos e estudos relacionados ao processamento de sinais digitais.

Um estudo publicado na EDERC[3] (Education and Research Conference), trata da utilização do BeagleBoard C4 para processar em tempo real sinais de áudio com o algoritmo Diarization para obter um sinal mais nítido. Esse algoritmo tem a função de indexação de áudio, como por exemplo, quando se deseja separar a fala de um locutor do restante dos ruídos do ambiente. Apesar das limitações computacionais, alterações no algoritmo para obter uma otimização levaram à obtenção de resultados experimentais que comprovam a eficácia da abordagem (COLAGIACOMO et al., 2010).

Um projeto, desenvolvido pela Universidade Stanford[4] e denominado "Satellite CCRMA" (BERDAHL, 2013), tem o objetivo de ser uma plataforma para a construção de instrumentos musicais embutidos para ser usada por artistas e engenheiros, possibilitando a criação de novas formas de interação com a tecnologia durante o processo de produção ou execução de músicas. O projeto é baseado em hardware e software de código aberto, e entre os dispositivos estudados estão o Raspberry Pi e o BeagleBoard xM, aliados a softwares como Pure Data extended, SuperCollider, Audacity, JackTrip, Emacs e g++/gcc.

No projeto envolvendo o BeagleBoard, realizado pelo "Satellite CCRMA" (BERDAHL, 2011), foram implementados processadores de efeitos de áudio em tempo real (pedais de efeitos para guitarras), instrumentos musicais baseados em sensores (uma pista de dança para vídeo games usada para inserir dados no Pure Data para produzir sons) e sensores que capturam o movimento da água em um recipiente e gerando sons.

Nos próximos capítulos serão detalhadas as atividades e resultados obtidos com o presente trabalho. As informações contidas nas seções anteriores serão de grande importância para a compreensão e aproveitamento do conteúdo a frente descrito.

[3] `http://www.ti.com/ww/eu/ederc/`
[4] `http://www.stanford.edu/`

3 ANÁLISE DOS CENÁRIOS DE PROCESSAMENTO DE SINAIS DIGITAIS DE ÁUDIO EM TEMPO REAL

Este capítulo tem como objetivo a exibição dos resultado obtidos durante os testes, ações realizadas para a obtenção dos dados, bem como as dificuldades encontradas.

3.1 METODOLOGIA

Para um sistema de processamento de sinais digitais trabalhar em tempo real, o mesmo deve possuir uma latência baixa o suficiente para que o tempo de processamento não seja percebido pelo usuário. Assim, a latência se torna uma questão central para a implementação de sistemas em tempo real.

Os testes realizados tiveram foco em encontrar um equilíbrio entre o fator latência e qualidade sonora. Tal fator está diretamente ligado à taxa de amostragem aplicada ao sinal: quanto maior a taxa de amostragem, mais amostras do sinal analógico são representadas no meio digital em uma faixa de tempo, e consequentemente, maior a fidelidade do sinal. Entretanto, quanto melhor a qualidade do áudio, maior a exigência de recursos computacionais para o processamento do sinal. Da mesma forma, o tamanho do *buffer* está diretamente ligado aos fatores latência e qualidade do áudio: quanto menor o tamanho do *buffer*, menor o tempo de latência, porém, maior será o processamento necessário e maior o é o risco de perda da qualidade do áudio.

Para encontrar uma boa combinação entre qualidade e latência, foram modificados a taxa de amostragem e tamanho de *buffer* durante a execução dos filtros de diferentes níveis de exigência computacional, analisando a latência e a faixa de frequência abrangente no áudio processado.

3.1.1 AMBIENTE DE TESTES

Para se obter o tempo de latência dos sistemas, foi utilizada uma ferramenta chamada *jack_delay*[1], disponível no pacote de utilitários do *Jack*, capaz de medir com precisão a latência de ida e volta de uma cadeia de áudio circuito fechado.

O *jack_delay* permite medir a latência total do sistema, emitindo sons e capturando-os novamente depois passar através de toda a cadeia. Para isso, existe a necessidade de que o sistema seja fechado em um circuito de ida e volta do sinal. Isso pode ser feito ligando a saída de sua interface de áudio na entrada do dispositivo usando um cabo. Esta abordagem pode ser usada em circuitos analógicos ou digitais, porem a latência dos dispositivos envolvidos também será considerada.

Os testes foram realizados através da ligação dos dispositivos da seguinte forma:

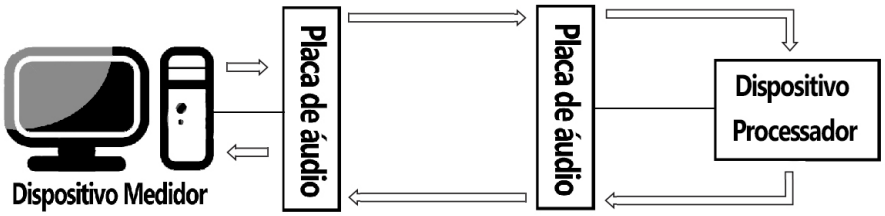


Figura 4: Topologia do ambiente de realização dos testes

No modelo descrito na Figura 4, o dispositivo medidor possui o *Jack_delay* funcionando e emitindo sinais de áudio para a saída da placa de áudio do dispositivo medidor. A saída da placa de áudio do dispositivo medidor está ligada a entrada da placa de áudio do dispositivo que irá processar os sinais de áudio, esta placa converte o áudio novamente para o meio digital e envia para o dispositivo processador para ser manipulado pelo Pure Data.

Após ser processado, o sinal de áudio realiza o percurso contrário, indo da saída da placa de áudio do dispositivo processador para a entrada da placa de áudio do dispositivo medidor. Apos isso o áudio é registrado pelo *Jack_delay*, que realiza o calculo do tempo gasto na realização do circuíto pelo áudio.

Apesar da medição considerar a latência de todos dispositivos envolvidos, testes preeliminares foram realizados a fim de se medir a latência gerada por cada elemento. Com isso, foi possível chegar a uma latência de 9,10 ms para cada placa de áudio envolvida na cadeia.

[1] http://wiki.linuxaudio.org/apps/all/jack_delay

3.2 ATIVIDADES DESENVOLVIDAS

Para atingir o objetivo proposto, foram realizadas as seguintes ações:

1. Preparação do cenário DSP com um sistema operacional compatível com cada dispositivo e a ferramenta de manipulação de sinais digitais nos dispositivos computacionais escolhidos:

   - A primeira etapa foi a realização de levantamentos para a verificação do desempenho de algumas distribuições Linux como o RaspBian[2], Debian[3], Ubuntu[4] e Angstrom[5], para processamento de sinais, com o intuito de encontrar a melhor opção para cada plataforma de estudo. Foram escolhidas as distribuições Raspbian para o cenário com o Raspberry Pi e Ubuntu para o BeagleBone e Notebook, visto que estas distribuições são as mais usadas nesses dispositivos para o processamento de sinais digitais pois apresentam poucos problemas de incompatibilidade com os programas envolvidos;
   - Os sistemas operacionais escolhidos foram preparados para um melhor desempenho no processamento de áudio. Foram realizadas configurações para o correto funcionamento da interface de áudio USB, desinstalação de serviços e programas desnecessários, além da instalação de pacotes e aplicativos específicos para o processamento de dados em tempo real;
   - O software Pure Data foi instalado e configurado para trabalhar com os dispositivos de entrada e saída de sinais.

2. Implementação de filtros com diferentes níveis de exigência computacional com

   - Seguindo a literatura, algoritmos de manipulação de sinais podem exigir diferentes níveis de recursos computacionais para serem executados. Nesse contexto é necessário avaliar o comportamento do sistema ao processar filtros de diferentes categorias e em diferentes condições. Para isso foram implementados ou adaptados os filtros Fuzz, Phaser e Ring Modulator em Pure Data, para processar sinais digitais de áudio.

3. Manipulação de sinais de áudio convertidos por uma interface de áudio externa ligada via USB:

[2] `http://www.raspbian.org/`
[3] `http://www.debian.org/`
[4] `http://www.ubuntu.com/`
[5] `http://www.angstrom-distribution.org/`

- Configuração dos dispositivos para receberem sinais de áudio de uma interface de áudio externa;
- Execução dos algoritmos de filtragem em diferentes níveis de exigência computacional nos dispositivos escolhidos, processando os sinais capturados pela placa de áudio externa;
- Realização de testes de desempenho, coletando dados sobre o processamento nas plataformas escolhidas para o estudo.

4. Manipulação de sinais de áudio convertidos por interfaces de áudio conectadas por portas I2S nas plataformas estudadas:
    - Foram realizadas modificações e configurações no sistema Raspbian para que o mesmo suporte a comunicação com o dispositivo de entrada e saída de áudio por meio das portas I2S. Apesar do sucesso nas modificações, o o dispositivo externo não funcionou, provavelmente por defeito do dispositivo ou incompatibilidade, o que causou a inviabilidade das atividades com I2S e o abandono das mesmas.

3.3 COLETA DE DADOS

Para verificar a latência utilizamos softwares que conseguem analisar a diferença de tempo entre a entrada e a saída do sinal. No caso de faixa de frequência, um sinal de áudio que contenha todas as frequências interpretadas pelo ouvido humano (entre 20Hz e 20KHz), como um ruído rosa[6], pode ser processado no cenário de processamento, e medido através de um espectrograma na saída do cenário, a fim de analisar a curva de resposta de frequência do sinal processando, levando-se em conta as características do algoritmo aplicado ao sinal.

Deve-se levar em conta a difícil comparação entre os dispositivos propostos ao estudo e os dispositivos exclusivamente produzidos para DSP, visto que, torna-se difícil obter os dados comparativos de dispositivos produzidos para DSP dos quais não possuíamos acesso para a realização dos testes. Dessa forma, para obter dados comparativos, os testes também foram realizados em um computador pessoal de capacidade mediana aos encontrados no mercado atualmente, contando com um processador Intel i5 segunda geração e 4GB de memória RAM, uma vez que os mesmos são populares e amplamente utilizados como alternativa a equipamentos exclusivos a processamento de sinais digitais e conseguem obter resultados satisfatórios.

[6]O ruído rosa é um tipo de ruído produzido pela combinação simultânea de sons de todas as frequências (FILHO,)

3.3.1 FILTROS UTILIZADOS

Foram selecionados quatro programas de processamento de sinais digitais, escritos em Pure Data, para a realização dos testes:

3.3.1.1 FILTROS DE BAIXO CUSTO COMPUTACIONAL

- Sem Filtro: um programa, que cria uma ligação da entrada para a saída de áudio, fazendo com que o sinal passe pelo Pure Data, sofrendo todo o processo de conversões necessárias para o processamento de um sinal de áudio pelo Pure Data, porém, sem modificar intencionalmente as características do sinal. Sua implementação em Pure Data está disponível no Anexo 7.1;

- Fuzz: filtro amplamente utilizado no processamento de instrumentos, principalmente guitarras. Sua característica é saturar o som através de um multiplicador de frequências, que altera um sinal de áudio até quase torná-lo uma onda quadrada e adiciona sobretons complexos (ZÖLZER, 2011). Sua implementação em Pure Data está disponível no Anexo 7.2;

- Phaser: É um efeito de áudio que cria um efeito de varredura no áudio através da modulação de um sinal. O filtro causa cancelamentos de componentes de frequência no sinal original, e mistura a saída do filtro com o sinal original. Para obter o efeito de oscilação no áudio, uma baixa frequência osciladora (LFO - Low Frequency Oscillation), ou seja , um sinal de áudio geralmente abaixo dos 20 Hz que cria um ritmo de pulsação ao invés de um som audível, e é usado para mover-se lentamente o centro da frequência (ZÖLZER, 2011). Sua implementação em Pure Data está disponível no Anexo 7.3;

3.3.1.2 FILTROS DE MÉDIO CUSTO COMPUTACIONAL

- Ring modulator: implementa uma modulação de amplitude ou frequência de mistura, realizada através da multiplicação de dois sinais, onde um é tipicamente uma onda senoidal ou de outra forma de onda simples (MASSAT, 2013). Este filtro possibilita a criação e efeitos que geram sons de vozes de robôs, muito usados em filmes de ficção científica a partir dos anos 50. Sua implementação em Pure Data está disponível no Anexo 7.4;

Para a escolha dos filtros foram levados em conta do custo computacional, as características sonoras e a pureza dos filtros, uma vez que existe uma grande quantidade de simuladores

de equipamentos analógicos que possuem características específicas imitadas em implementações digitais, podendo ocorrer acréscimos ao custo computacional durante suas execução. Assim, foram utilizados filtros puros oferecidos pelo Pure Data. Esses filtros são como bibliotecas para o desenvolvimento de filtros compostos e oferecem um melhor controle sobre a implementação dos filtros.

3.3.2 CONFIGURAÇÕES UTILIZADAS

Foram realizados testes com diferentes configurações no Pure Data para analisar as condições de latência que cada uma implica no processamento. As configurações de Taxa de amostragem e tamanho de buffer em blocos de amostras estão descritas na Tabela 3.

Taxa de Amostragem	**Blocos de amostra de Buffer**
44,1 KHz	64
44,1 KHz	128
64 KHz	64
64 KHz	128
88,2 KHz	64
88,2 KHz	128

Tabela 3: Configurações do PureData utilizadas nos testes

4 RESULTADOS

Com os testes realizados foi possível chegar às conclusões sobre a capacidade de processamento de sinais digitais de áudio em tempo real dos dispositivos. Se tratando de processamento de áudio em tempo real os pontos que mais chamam atenção para um bom resultado são o atraso resultante do processamento, e possíveis perdas de qualidades do áudio durante o processamento. Dessa forma, foram realizados testes para medir a latência do áudio processado e as características sonoras do sinal processado em cada dispositivo e em cada filtro proposto.

4.1 LATÊNCIA

Ao analisar a latência imposta pelo cenário de processamento, composto pelo dispositivo computacional e os dispositivos de entrada e saída, foi possível obter o tempo de toda a cadeia de dispositivos envolvida no experimento. Não foi considerado o tempo de processamento individual dos dispositivos estudados, uma vez que os outros dispositivos envolvidos, como as placas de áudio e o computador usado para medição também impõem latência no sinal. Entretanto, os valores obtidos representam um baixo tempo, mesmo com os acréscimos de tempo impostos pelos outros dispositivos envolvidos na cadeia de áudio.

Durante os testes foi possível perceber que as características sonoras geradas pelos filtros podiam interferir no resultado. A latência foi medida usando-se um sinal de áudio enviado para o processamento e recebido na saída da cadeia de dispositivos, onde o dispositivo medidor calcula o tempo gasto entre o envio e o retorno do sinal. As alterações geradas no áudio processado por alguns filtros e por configurações dos dispositivos que geraram prejuízos ao áudio, fizeram, em alguns casos, com que o sinal de retorno não fosse corretamente reconhecido.

Dessa forma, as características impostas pelo filtro no áudio processado como, por exemplo, atrasos intencionais gerados por filtros para causar um determinando efeito, causavam interferência no dispositivo medidor, afetando assim a medição do tempo de latência. Assim, o filtro Phaser apresentou uma latência bastante elevada em relação aos outros filtros nos testes realizados, causado pela sua constituição e implementação, uma vez que esse aumento foi

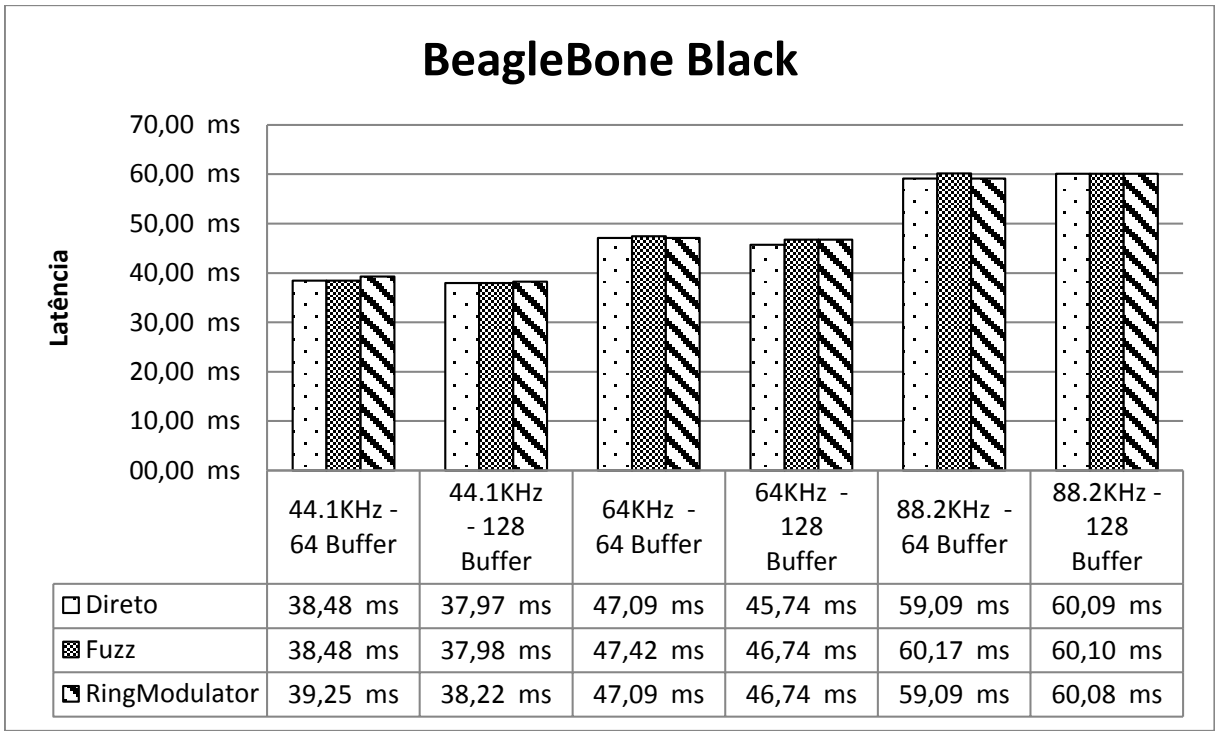
padronizado nos diversos dispositivos testados.

Além dos testes de medição, foi possível realizar testes de audição. Um dos testes realizados por audição foi o processamento de áudio originado por um microfone, com a reprodução do áudio em tempo real. Com esse teste foi possível perceber que todos dispositivos testados, em todas configurações realizadas, um sutil atraso no áudio ocorria, o que não causava um desconforto ao usuário ao ouvir a sua voz, captada por um microfone, e processada em tempo real, com um pouco de atraso em relação à voz original.

Nas próximas sessões, serão apresentados os dados obtidos nos experimentos, categorizados de acordo com os dispositivos, possibilitando um panorama das capacidades de processamento de sinais digitais de áudio em cada dispositivo de acordo com cada configuração realizada. Posteriormente, os dados obtidos serão comparados entre os dispositivos, possibilitando obter o dispositivo com o melhor desempenho dentro de suas características e limitações.

4.1.1 BEAGLEBONE BLACK

Os testes de latência realizados no BeagleBone Black obtiveram os seguintes resultados:



	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64 Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
Direto	38,48 ms	37,97 ms	47,09 ms	45,74 ms	59,09 ms	60,09 ms
Fuzz	38,48 ms	37,98 ms	47,42 ms	46,74 ms	60,17 ms	60,10 ms
RingModulator	39,25 ms	38,22 ms	47,09 ms	46,74 ms	59,09 ms	60,08 ms

Figura 5: Latência imposta pelo processamento no BeagleBone Black
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

Todos os filtros testados no BeagleBone Black obtiveram um resultado satisfatório nos testes auditivos quando configurados com uma taxa de amostragem menor, sendo bem perceptível o atraso do som quando configurado com a taxa de amostragem em 88,2KHz. Esse atraso é ainda mais visível ao analisar os dados sobre as latências impostas nos testes realizados, onde é possível perceber um crescimento da latência junto ao crescimento da taxa de amostragem, levando as configurações com 88,2KHz de taxa de amostragem a ter as maiores latências.

O comportamento do crescimento da latência em relação ao aumento da taxa de amostragem se mostrou uniforme nos filtros Fuzz, Ring Modulator e no sinal sem aplicação de filtros, mantendo desempenho parecido, não havendo grande variação nos tempos de processamento. O filtro Ring Modulator, que segundo a literatura é o de maior custo computacional entre os aqui estudados, também não apresentou significativas diferenças em termos de desempenho.

É possível perceber que o uso da taxa de amostragem em 44.1KHz, permite obter o melhor tempo de resposta. Apesar de ser a menor taxa de amostragem testada, 44.1KHz, segundo o Teorema de Nyquist, é o suficiente para registrar todas as frequências perceptíveis ao ouvido humano no meio digital. Em configurações usando taxas de amostragem maior, o Pure Data passa a ter uma taxa maior erros com perda de amostras, o que causa um ruído característico, perceptível no áudio final.

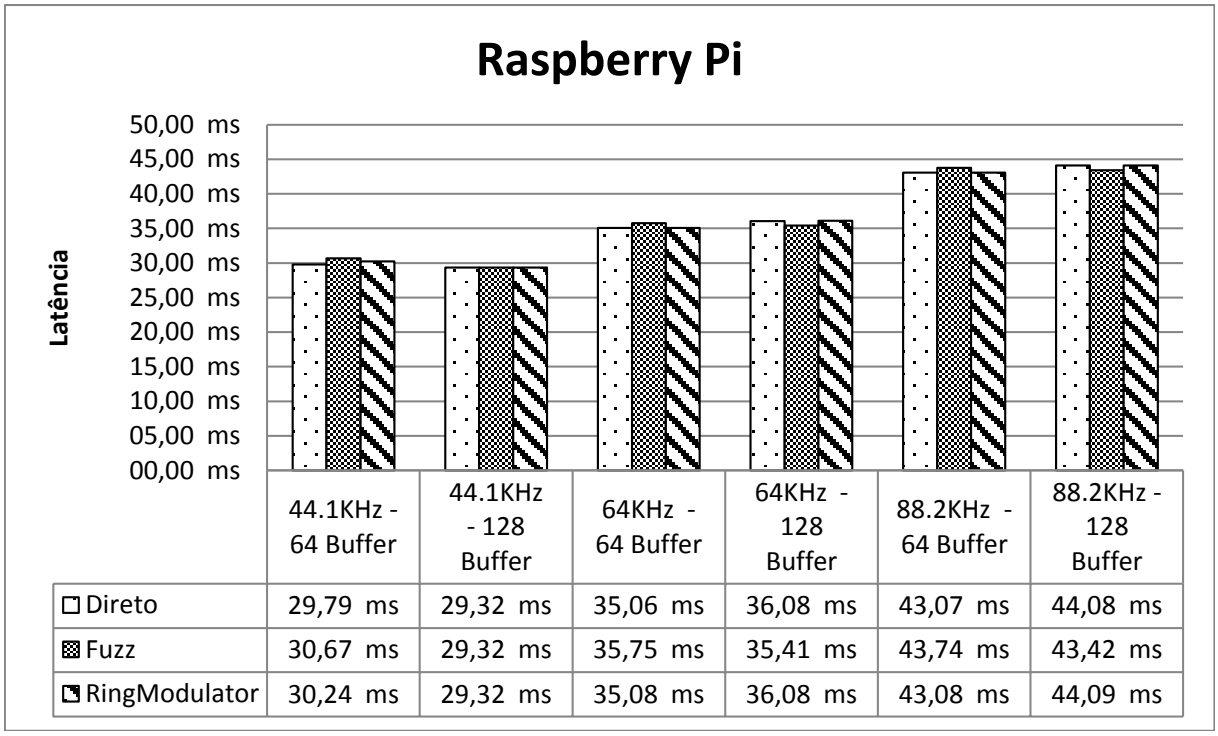
Em relação ao tamanho do buffer nas configurações do ambiente, o uso de 128 amostras como tamanho de buffer resultou em um melhor desempenho, gerando uma latência levemente menor que nos testes usando 64 amostras de buffer. Porém, em todos os testes, as configurações de buffers não geraram grande variação de latência.

Cabe ressaltar que o filtro Phaser gerou dados bastante controversos, bem acima da média dos outros filtros testados. Porém, por suas características interferirem na medição da latência, sua comparação em relação aos outros filtros testados tornou-se difícil, o que não impede uma comparação entre os resultados gerado pelo Phaser em cada dispositivo.

4.1.2 RASPBERRY PI

Para o Raspeberry Pi, a realização dos testes obteve os seguintes resultados:

O Raspberry Pi obteve atrasos de processamento que traz boas condições para o processamento de áudio próximo de tempo real, conforme apresentado na Figura 6. O atraso entre o áudio não processado e o processado, na maior parte dos testes, foi imperceptível ou pouco perceptível em termos auditivos. Os menores tempos de latência foram obtidos com a taxa de amostragem em 44.1 Khz com 128 amostras de buffer. Entretanto, houve pouca variação



	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64 Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
Direto	29,79 ms	29,32 ms	35,06 ms	36,08 ms	43,07 ms	44,08 ms
Fuzz	30,67 ms	29,32 ms	35,75 ms	35,41 ms	43,74 ms	43,42 ms
RingModulator	30,24 ms	29,32 ms	35,08 ms	36,08 ms	43,08 ms	44,09 ms

Figura 6: Latência imposta pelo processamento no Raspberry PI
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

de latência entre as configurações de tamanho de buffer. O aumento da latência imposta pelo ambiente de teste em cada configuração estuda pode ser comparado no gráfico da Figura 6.

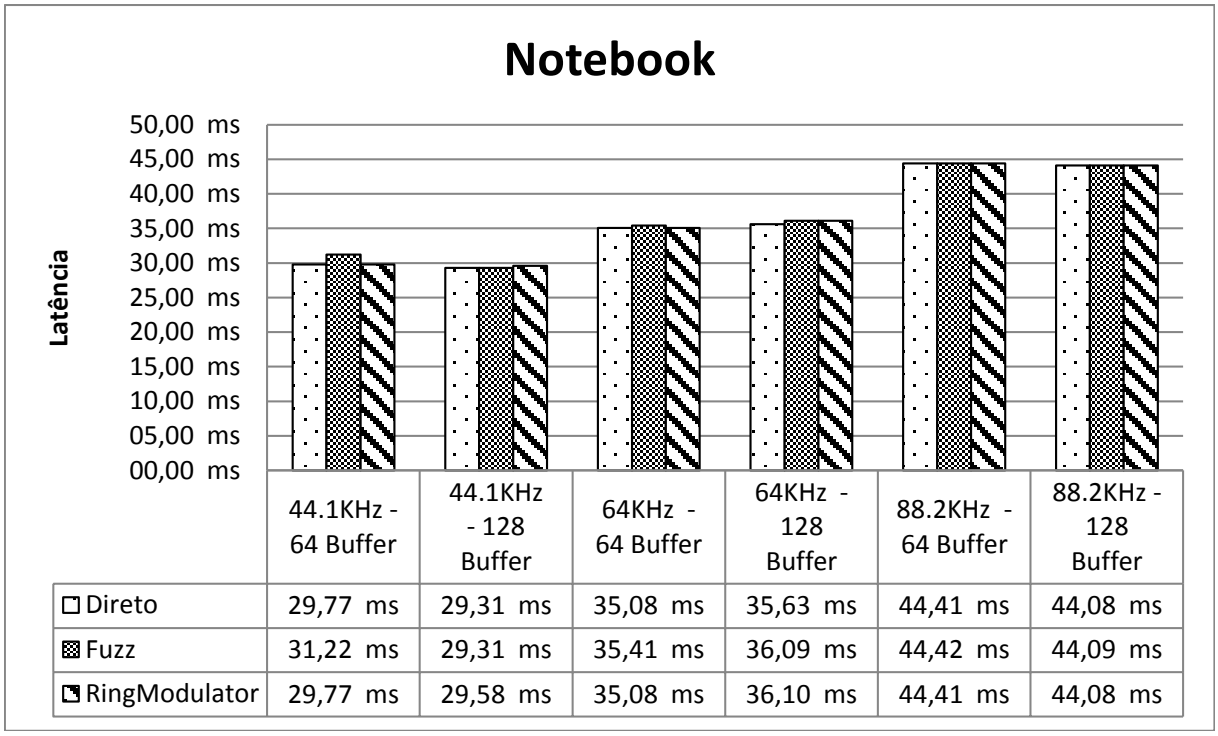
Mais uma vez o filtro Phaser obteve valores muito além dos outros filtros testados, deixando sua comparação com os mesmos difíceis por conta de suas características.

Em relação a diferença de tempo de processamento entre os filtros, os mesmos tiverem valores parecidos em cada configuração. O crescimento do atraso com o aumento da taxa de amostragem também se manteve uniforme entre os filtros.

4.1.3 NOTEBOOK

O notebook apresentou resultados semelhantes ao outros dispositivos, e como esperado, possui configurações que permitem o uso de processamento de áudio digital com um tempo de resposta que permite o uso em tempo real, mantendo o mesmo padrão de aumento de latência em relação ao aumento da taxa de amostragem como pode ser visto no gráfico da Figura 7.

Como nos outros dispositivos, os valores dos tempos de latências se mantiveram próximos entre os filtros de processamento e o sinal sem filtro, mesmo com as variações de confi-



	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64 Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
Direto	29,77 ms	29,31 ms	35,08 ms	35,63 ms	44,41 ms	44,08 ms
Fuzz	31,22 ms	29,31 ms	35,41 ms	36,09 ms	44,42 ms	44,09 ms
RingModulator	29,77 ms	29,58 ms	35,08 ms	36,10 ms	44,41 ms	44,08 ms

Figura 7: Latência imposta pelo processamento no Notebook
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

gurações.

Assim como nos outros dispositivos, a latência é pouco perceptível com taxa de amostragem em 44.1Khz, e aumenta com o incremento da taxa de amostragem. Porém, com taxas de amostragens altas ocorrem poucos erros de perda de amostras, não resultando em prejuízo aparente ao áudio processado.

4.2 ANÁLISE E COMPARAÇÃO ENTRE OS DISPOSITIVOS ESTUDADOS

A princípio, todos dispositivos testados apresentaram uma latência baixa o suficiente para não ser percebida, ou ser pouco percebida pelo ouvido, o que permite o processamento de áudio em situações que exigem uma resposta rápida de processamento, como a aplicação de efeitos no som de um instrumento musical, sem atrapalhar o desempenho do músico. Após as medições de latência, foi possível afirmar que os dispositivos podem executar o processamento de sinais digitais com características de processamento em tempo real, uma vez que as latências, mesmo com o acréscimos de tempo pelos outros elementos envolvidos no experimento, se mantiveram próximo de 30 milissegundos nas configurações menos exigentes.

Ao se considerar que cada placa de áudio utilizada do ambiente gera uma latência de

9,10 milissegundos. A utilização das duas placas gerou um atraso de aproximadamente 18,2 milissegundos, valor este que, pode ser desconsiderado na analise da latência dos dispositivos processadores. Com isso, a latencia individual dos dispositivos processadores estiveram entre aproximadamente 11,56 e 21,05 milissegundos. Latências da ordem de 30ms podem ser consideradas normais e aceitáveis na maioria dos casos, não interferem no desempenho do instrumentista . A fim de comparar o desempenho entre os dispositivos, serão comparados os desempenhos dos mesmos em cada filtro.

4.2.1 SINAL DE ÁUDIO PURO SEM FILTROS

Para obtermos um parâmetro de qual é o tempo de processamento no Pure Data mais próximo do mínimo possível, foram realizados testes onde foi medida a latência de um programa em Pure Data usando apenas os métodos de entrada e saída da ferramenta. Esses métodos são usados nos outros filtros com a criação dos algoritmos de processamento. Nesse caso, o sinal de áudio, originário da entrada uma placa de som, passa pelo Pure Data e é novamente enviado para saída da placa de áudio.

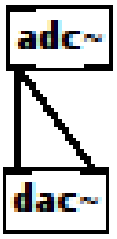


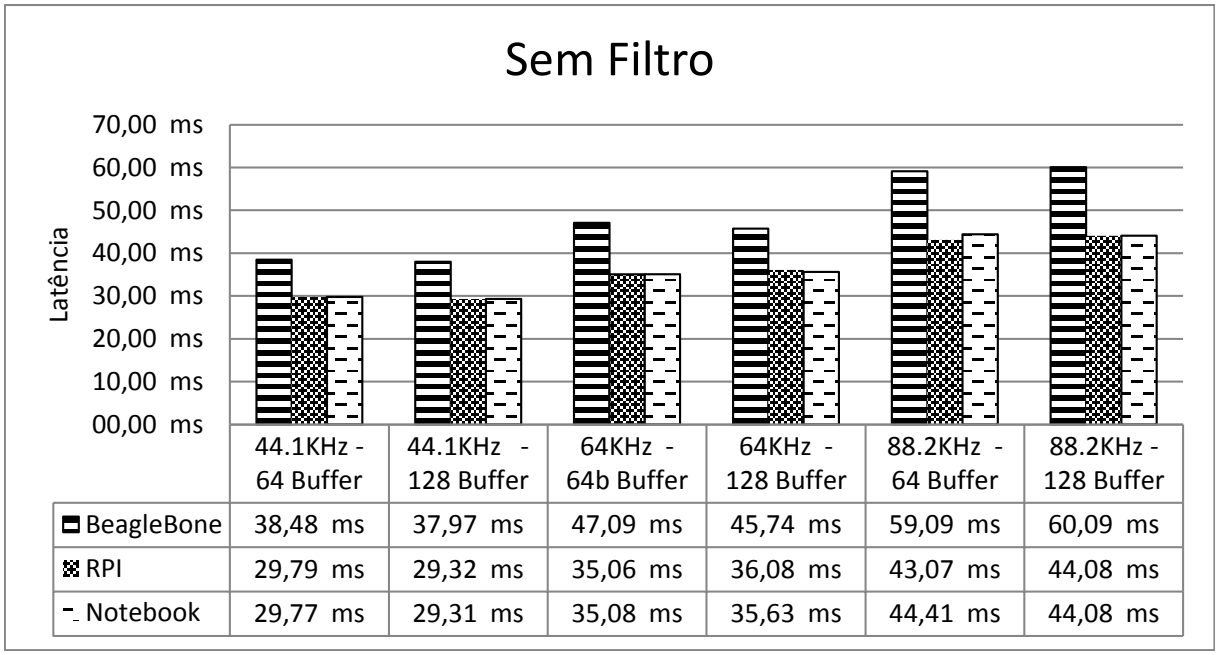
Figura 8: Ligação entre entrada e saída de áudio no Pure Data

Os resultados de latências obtidos em cada plataforma são apresentados na Figura 9.

Os testes mostraram que o tempo de processamento do áudio no Raspberry Pi se manteve, em todas configurações, muito próximo do tempo de processamento do Notebook. O BeagleBone Black obteve latências mais altas em todas configurações, porém ainda assim, o tempo de atraso registrado não causa grandes prejuízos.

4.2.2 FUZZ

O Fuzz é um efeito amplamente utilizado em instrumentos como guitarras, é de baixo custo computacional e de implementação simples em Pure Data. No algoritmo Pure Data testado acrescentando apenas uma amplificação no áudio, foi utilizado o modulo “clip”, nativo do

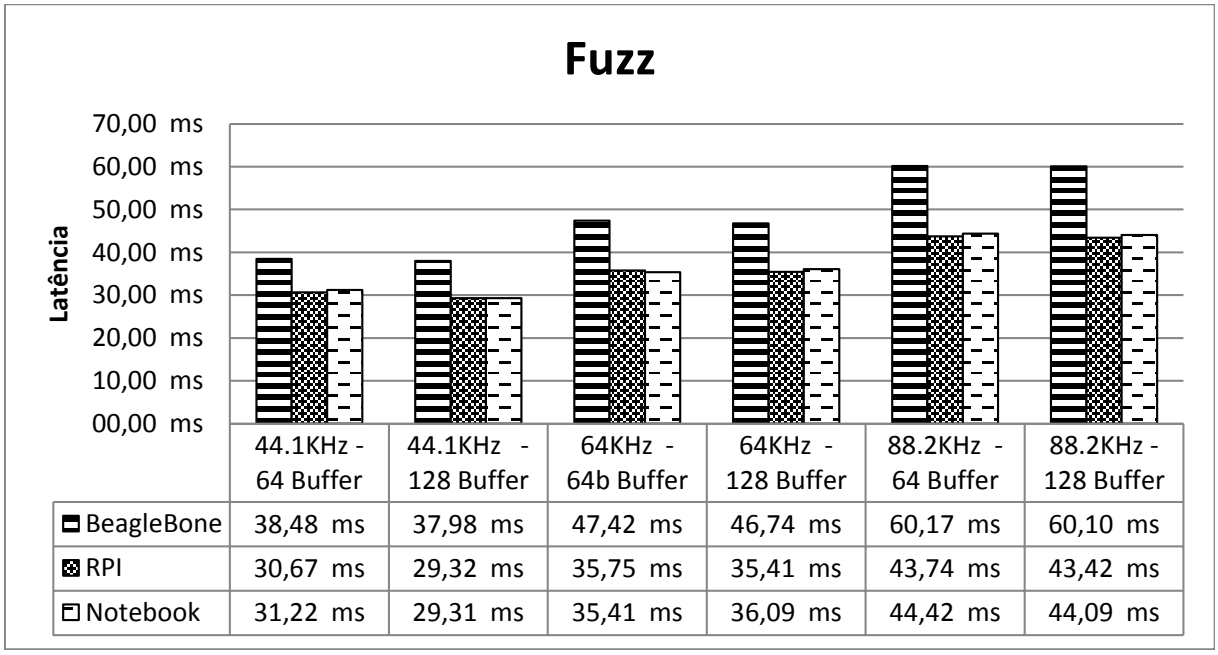


	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64b Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
BeagleBone	38,48 ms	37,97 ms	47,09 ms	45,74 ms	59,09 ms	60,09 ms
RPI	29,79 ms	29,32 ms	35,06 ms	36,08 ms	43,07 ms	44,08 ms
Notebook	29,77 ms	29,31 ms	35,08 ms	35,63 ms	44,41 ms	44,08 ms

Figura 9: Latência imposta pelo processamento com Pure Data sem filtros
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

Pure Data, para se produzir uma distorção audível no áudio.

Os resultados obtidos após os testes estão disponíveis na Figura 10.



	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64b Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
BeagleBone	38,48 ms	37,98 ms	47,42 ms	46,74 ms	60,17 ms	60,10 ms
RPI	30,67 ms	29,32 ms	35,75 ms	35,41 ms	43,74 ms	43,42 ms
Notebook	31,22 ms	29,31 ms	35,41 ms	36,09 ms	44,42 ms	44,09 ms

Figura 10: Latência imposta pelo processamento de Fuzz no Pure Data
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

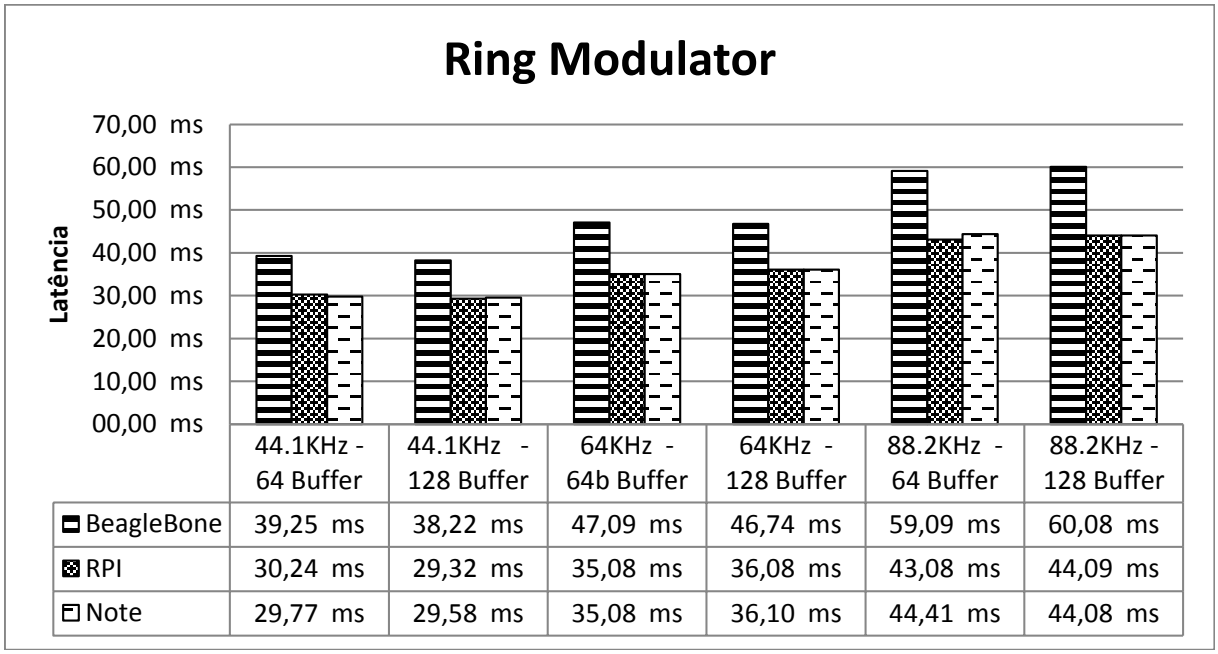
Assim como no sinal direto, o processamento com Fuzz, gerou latências parecidas

entre o Rasperry Pi e o Notebook. No geral, o Raspberry Pi gerou resultados ligeiramente menores que o Notebook. O Beaglebone Back se mostrou novamente, mais lento que os demais dispositivos, apresentando uma latência mais alta, porem muito próxima dos resultados obtidos nos testes sem uso de filtros.

4.2.3 RING MODULATOR

O Ringo Modulator é considerado um filtro de médio custo computacional. Sua implementação é relativamente complexa, unindo diversos módulos nativos do Pure Data, gerando uma cadeia de efeitos e possibilidades de parâmetros de configuração até chegar ao efeito final.

Os testes resultaram nos seguintes dados:



	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64b Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
BeagleBone	39,25 ms	38,22 ms	47,09 ms	46,74 ms	59,09 ms	60,08 ms
RPI	30,24 ms	29,32 ms	35,08 ms	36,08 ms	43,08 ms	44,09 ms
Note	29,77 ms	29,58 ms	35,08 ms	36,10 ms	44,41 ms	44,08 ms

Figura 11: Latência imposta pelo processamento de Ring Modulator no Pure Data
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

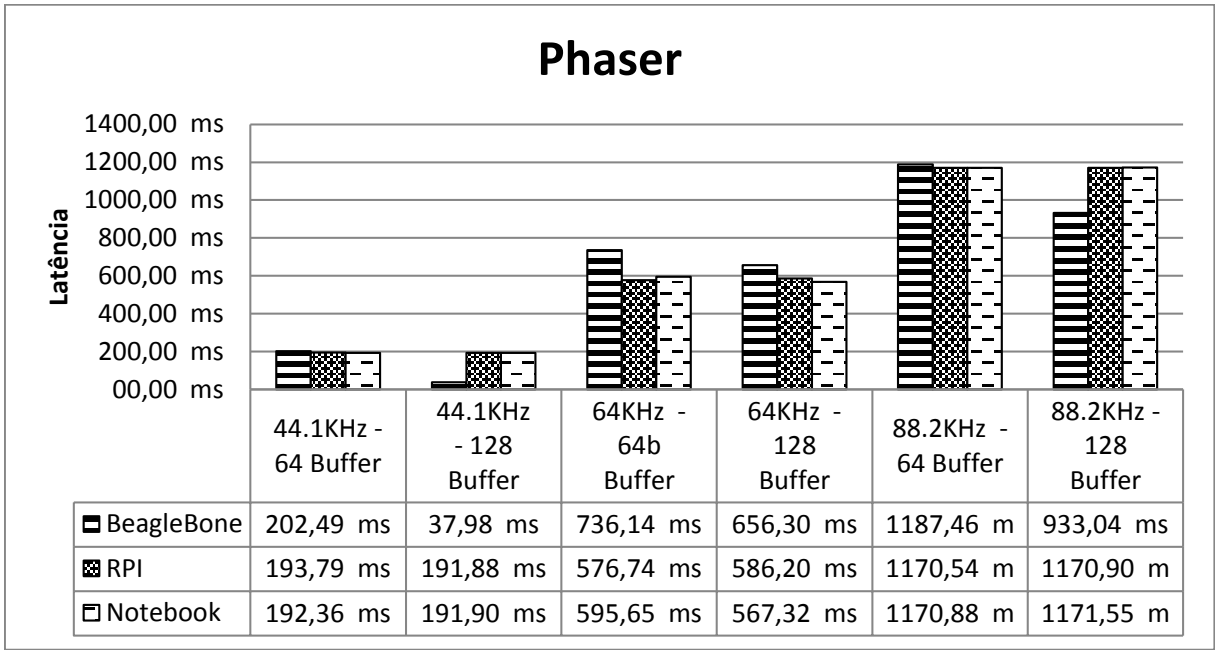
Os dados resultantes dos restes realizados com o filtro Ring Modulator mostram o que os dispositivos se comportaram de maneira semelhante aos testes realizados com os demais filtros, com o Raspberry Pi e o Notebook com valores próximos e com o Beaglebone Black apresentando valores maiores.

Vale destacar que, em todos dispositivos, não houve aumento maior a 1 ms entre os testes com filtros de menor custo computacional e o Ring Modulator, que segundo a literatura é o filtro de maior custo computacional entre os testados no presente estudo, e os testes realizados sem o processamento de filtros.

4.2.4 PHASER

O filtro Phaser é um efeito amplamente utilizado na produção de áudio, é popular pelo seu uso em instrumentos como guitarra, porém também é muito usado em efeitos, sintetizadores e até mesmo em voz. É considerado um filtro de baixo custo computacional, sua implementação em Pure Data é relativamente simples.

Nos testes realizados a ferramenta de medição apresentou um valor de latência muito além do esperado, visto que nos testes de audição o Phaser não gerou grande atraso perceptível, ficando em alguns casos até imperceptível. Porém, como citado anteriormente, a ferramenta de medição usa um sinal de áudio para realizar as medições, e o mesmo está sujeito às transformações impostas pelo filtro. O Phaser trabalha com o cancelamento de fase para gerar o efeito, essa modulação gerou os valores não correspondentes a latência real, como apresentados na Figura 12



	44.1KHz - 64 Buffer	44.1KHz - 128 Buffer	64KHz - 64b Buffer	64KHz - 128 Buffer	88.2KHz - 64 Buffer	88.2KHz - 128 Buffer
BeagleBone	202,49 ms	37,98 ms	736,14 ms	656,30 ms	1187,46 m	933,04 ms
RPI	193,79 ms	191,88 ms	576,74 ms	586,20 ms	1170,54 m	1170,90 m
Notebook	192,36 ms	191,90 ms	595,65 ms	567,32 ms	1170,88 m	1171,55 m

Figura 12: Latência imposta pelo processamento de Phaser no Pure Data
KHZ = Taxa de amostragem
b = Quantidade de blocos de amostra de buffer

Apesar de não ser possível garantir que tais resultados sejam condizentes e que possam trazer algum grau de conhecimento em relação o tempo de processamento em cada dispositivo, os testes mostraram que mesmo em condições adversas os resultados mantiveram um padrão de crescimento parecido ao encontrado nos outros filtros, e da mesma forma, os resultados do Raspberry Pi e do Notebook foram semelhantes.

4.3 ANÁLISE DE ESPECTRO

Analisar características de um áudio pode ser uma tarefa complexa. Nos testes foram gerados espectrogramas do áudio processado para melhor comparar o resultado em cada um dos dispositivos e poder analisar as perdas na qualidade do áudio geradas pelo processamento.

Para os testes, foi usado um programa em Pure Data, que não causa modificações no áudio. Sinais de áudio contendo um ruído rosa foram processados gerando espectrogramas contendo a média de ganho em decibéis de cada frequência no sinal processado. Após isso foi possível comparar de maneira visual o áudio gerado em cada dispositivo para identificar diferenças entre os sons processados, causadas apenas pela passagem do áudio pelo ambiente de processamento em diversas configurações.

Cabe ressaltar que, as características sonoras impostas pelos outros elementos envolvidos no processamento, como placas de áudio e cabos, e até mesmo ruídos originários da rede elétrica, são registrados nesse teste. Entretanto, o método demonstra ser eficiente para visualizar modificações no áudio. Os resultados obtidos são apresentados a seguir.

4.3.1 NOTEBOOK

Através dos espectrogramas gerados a partir dos áudios processados pelo Pure Data no Notebook, foi possível chegar a conclusão que o processamento gera uma pequena modificação no áudio em relação ao áudio original, porém todas as configurações apresentam uma modificação semelhante, como descrito na Figura 13.

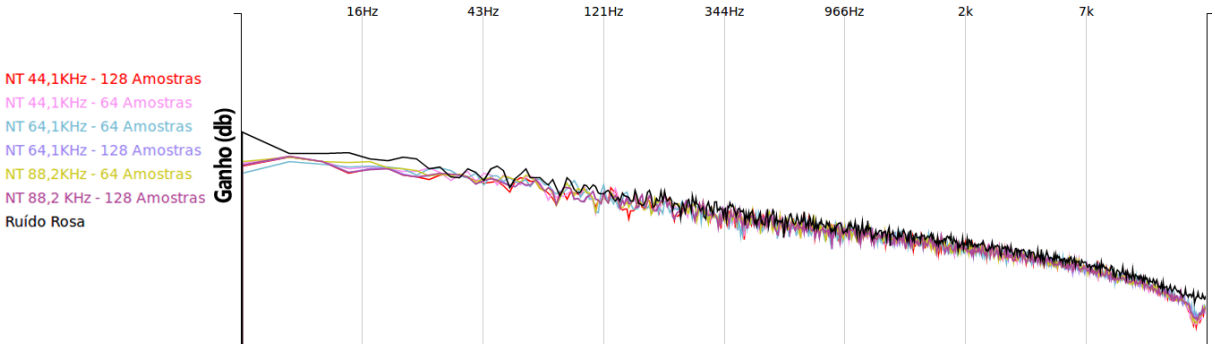


Figura 13: Espectogramas gerados pelo processamento no Notebook

4.3.2 BEAGLEBONE BLACK

Os áudios processados no BeagleBone Black apresentaram características parecidas em todas as configurações utilizadas, havendo pouca variação entre os espectros gerados, como

visível na Figura 14.

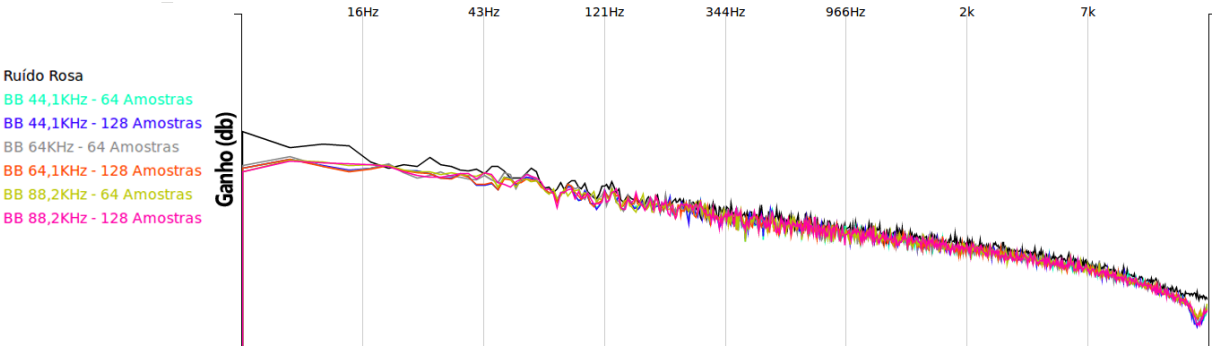


Figura 14: Espectrogramas gerados pelo processamento no BeagleBone Black

O sinal processado teve uma pequena redução de ganho em relação ao ruído original, porém não houve grande variação entre os sinais processados. Mesmo nas configurações com maior contraste entre os valores, e que nos testes de latência apresentaram valores distintos. Como apresentado na Figura 15 os espectros gerados foram parecidos. É possível comparar os espectros gerados pelo ruído rosa original sem processamento, ruído processado em configuração usando 44,1KHz e de taxa de amostragem.

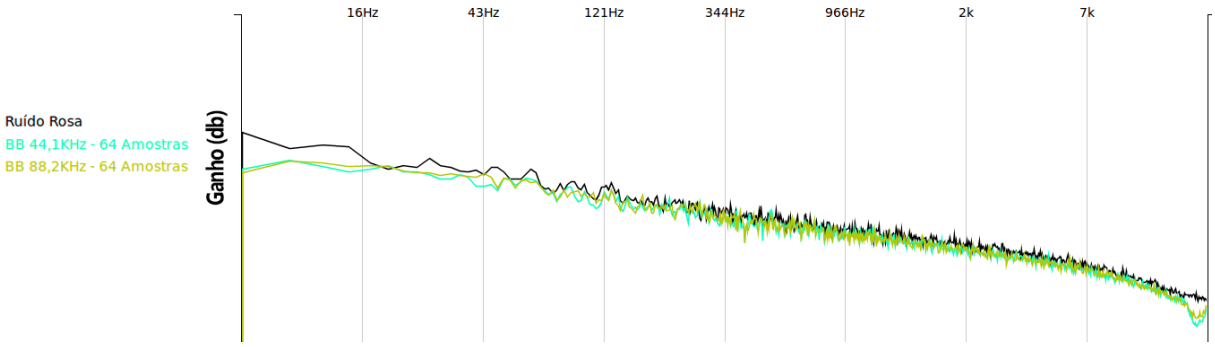


Figura 15: Espectrogramas gerados pelo ruído rosa original sem processamento, ruído processado em configuração usando taxa de amostragem de 44,1KHz e ruído processado em configuração usando taxa de amostragem de 88,2KHz no BeagleBone Black.

4.3.3 RASPBERRY PI

Como nos outros dispositivos, o Raspberry Pi não gerou grandes variações entre os sinais resultantes do processamento do Pure Data em diferentes configurações, como é possível verificar na Figura 16.

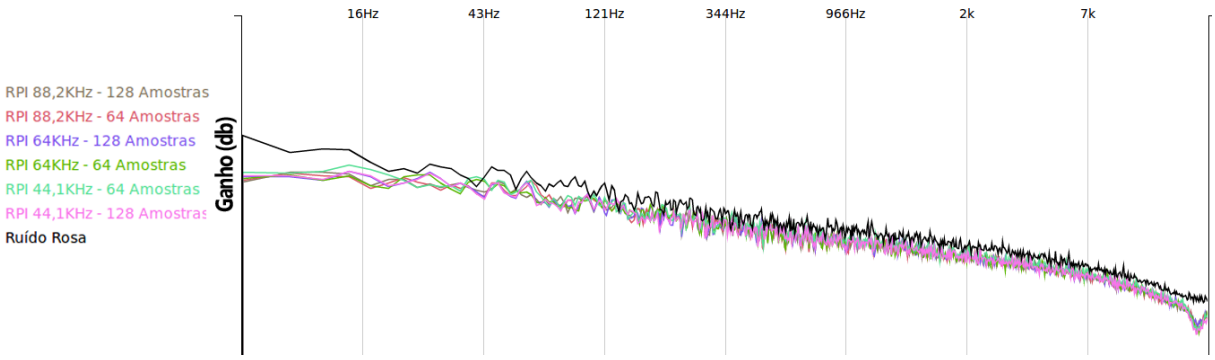


Figura 16: Espectrogramas gerados pelo processamento no Raspberry Pi

4.3.4 RASPBERRY PI EM MODO GRÁFICO

Como percebido nos testes de audição, o Raspberry Pi com o modo gráfico ativo, apresentou modificações na qualidade do áudio, apresentando perceptível perda de resolução no áudio. Ao se comparar o espectrograma gerado pelo áudio processado no Raspberry Pi em modo gráfico (com o Pure Data configurado com taxa de amostragem de 44,1KHz e buffer de 64 blocos de amostras) com espectrogramas gerados na mesma configuração sem o modo gráfico, é possível notar que houve diferenças entre cada sinal, como descrito na Figura 17.

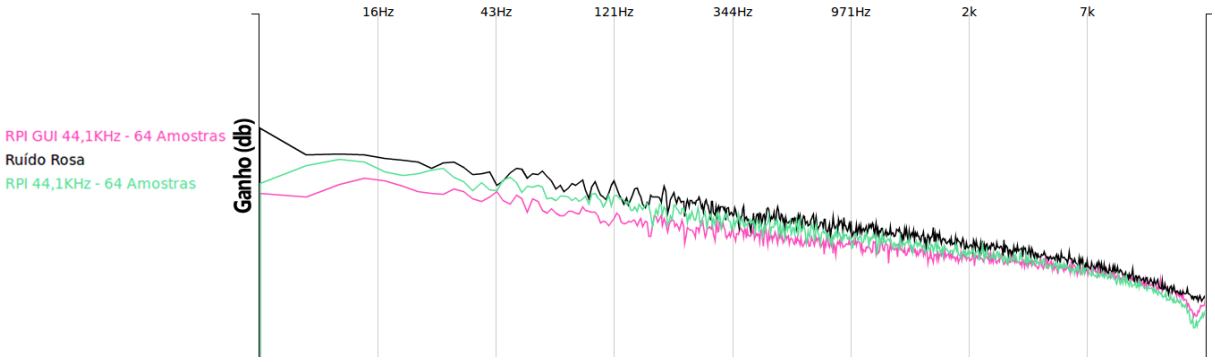


Figura 17: Espectrogramas gerados pelo processamento no Raspberry Pi

O sinal processado com o ambiente gráfico ativado teve uma redução do ganho em frequências a baixo de 3KHz em relação ao sinal processado sem o modo gráfico, e um pequeno ganho nas frequências maiores quando comparado ao sinal processado sem modo gráfico.

Essa modificação do espectrograma sintetiza de forma visual o que foi notado nos testes de audição, onde era claro a modificação do áudio e aparente perda de qualidade. Analisando as evidentes modificações do áudio processado e a quantidade alertas de erros exibidos pelo Pure Data, é notado que ouve muitas perdas de amostras, causadas pelas limitações do dispositivo. As perdas de amostras em grandes quantidades causam a redução da capacidade do sinal ser representado novamente no meio analógico com fidelidade ao original, caracterizando

um fenômeno conhecido como *Aliasing*.

O *Aliasing* ocorre quando por algum motivo a quantidade de amostras de um sinal é insuficiente para a sua reconstrução com fidelidade, ou seja, a quantidade de amostras por segundo é menor que a metade do valor da frequência máxima que se deseja registrar, gerando uma distorção e perda de qualidade do sinal (TANAKA et al., 2006).

5 CONCLUSÕES

As possibilidades de utilização de diferentes tecnologias para o processamento de sinais digitais em tempo real geram possibilidades de expansão dos recursos para a manipulação de sinais digitais. Este estudo teve como objetivo explorar as possibilidades de alguns dispositivos não voltados para o processamento de sinais digitais de áudio.

5.1 DIFICULDADES ENCONTRADAS

Esta seção apresenta as conclusões obtidas a partir da análise dos dados, bem como dificuldades encontradas e possibilidades de trabalhos futuros em continuidade dessa proposta. Das diversas atividades propostas inicialmente, algumas resultaram em problemas ou limitações durante seu desenvolvimento. Dentre elas:

- Atrasos não previstos durante a aquisição do dispositivo BeagleBone Black geraram um período de atraso nos trabalhos realizados com o mesmo.
- Houve dificuldade na implementação de filtros de alto custo computacional pela complexidade dos mesmos e pela falta de informação sobre a implementação dos mesmos em Pure Data. Este fato resultou na realização de poucos experimentos com filtros de custo computacional elevado.
- Os dispositivos Raspberry Pi e BeagleBone Black não tiveram sucesso no processamento dos filtros de alto custo computacional. A impossibilidade de geração de dados sobre estes filtros levou ao abandono dos mesmos.
- Dificuldades para o correto funcionamento de versões estendidas do Pure Data no Raspbian no Raspberry Pi e no Ubuntu no BeagleBone Black, inviabilizaram a implementação de uma maior quantidade de filtros que necessitam de módulos estendidos do Pure Data.
- Apesar do sucesso nas modificações e compilação do sistema operacional Raspbian para funcionar com placas de áudio I2S, o dispositivo externo não funcionou, provavelmente

por defeito do dispositivo ou incompatibilidade. Com a dificuldade de aquisição de dispositivos do gênero, e pela indisponibilidade de outra opção, os testes com dispositivos I2S, foram inviabilizados.

- A dificuldade na medição do atraso no processamento de sinais gerados por um controlador MIDI levou a inviabilidade da realização dos testes.

5.2 CONCLUSÕES SOBRE OS TESTES

Apesar das diferenças entre os dispositivos estudados, todos obtiveram um desempenho satisfatórios no processamento de sinais de áudio digital em tempo real. Os dispositivos obtiveram um comportamento parecido no crescimento da latência em relação crescimento da exigência nos parâmetros de configuração do Pure Data. Para melhor compreensão dos resultados, os mesmos serão divididos em tópicos:

5.2.1 DESEMPENHO NO PROCESSAMENTO DOS FILTROS DE DIFERENTES CUSTOS COMPUTACIONAIS

Durante os testes iniciais dos filtros, foi possível notar que filtros de maior exigência computacional, como os filtros de alto custo computacional, não conseguiram funcionar de forma correta no Raspberry Pi e no BeagleBone Black, obtendo apenas um ruído característico como saída e diversos erros de perda de pacotes relatados no log de saída do Pure Data, não sendo possível dados sobre os testes com a metodologia proposta.

Por outro lado, os filtros que obtiveram sucesso no processamento geraram um tempo de latência aceitável, sendo possível perceber que o Pure Data possui mecanismos de gerenciamento dos algoritmos nele implementados. Esses mecanismos garantem que os filtros tenham um bom desempenho, não havendo grandes variações do tempo de processamento como reflexo à complexidade e custo computacional maior.

Concluímos que os filtros que foram executados com sucesso não geraram uma grande variação de latência e perda de qualidade do áudio, mesmo com distintas classificações de custo computacional dos algoritmos usados para a implementação dos filtros.

5.2.2 DESEMPENHO EM DIFERENTES CONFIGURAÇÕES

Um dos quesitos mais relevantes para a obtenção de um tempo de processamento eficiente é a correta configuração dos dispositivos e softwares envolvidos nesta ação. Após a

realização dos testes, foi possível perceber a curva de crescimento da latência em relação a diferentes configurações realizadas, e definir a configuração do Pure Data que obtém a menor latência entre as configurações estudadas.

Entre as configurações, as taxas de amostragem menores geraram menos latência em todos dispositivos. Com o Pure Data trabalhando com taxa de amostragem de 44,1KHz as latências registradas em todos dispositivos possibilitam que o áudio seja convertido e processado no ambiente em tempos entre 29,76 e 39,25 milissegundos, o que não gera grandes prejuízos para o processamento de áudio em condições que necessitam de resposta em tempo real, possibilitando que o atraso seja quase imperceptível. Os dispositivos mostraram nos testes que, ao contrário do que se imaginava anteriormente, trabalham melhor com o maior tamanho de buffer do Pure Data entre os testados. Por se tratar de um buffer interno do Pure Data, o mesmo aparenta funcionar de forma diferente a outros softwares e dispositivos voltados para o processamento de áudio digital, que costumam forçar que os dados a passarem mais rapidamente pelo armazenamento de espera dos buffers em direção ao processamento, mesmo que tal ação leve a perda de informações. Nos testes realizados o Pure Data teve melhor desempenho com uma taxa de 128 amostras de buffers em relação ás configurações com 64 blocos de amostra como tamanho de buffer.

Além das questões de latência, foi perceptível nos testes de audição que no Raspberry Pi e BeagleBone, com configurações com taxa de amostragem de 88,2KHz, ocorria uma considerável quantidade de erros por perda de blocos de amostras, resultando em mensagens de erro e ruído característico na saída do áudio.

Desta forma, a configuração usando 44,1KHz de taxa de amostragem, e 128 amostras como tamanho de buffer, gera a menor latência nos três dispositivos estudados, permitindo assim, uma latência com proporções aceitáveis e uma resolução de áudio digital com ótima fidelidade em relação ao áudio original no meio analógico.

5.2.3 COMPARATIVO ENTRE OS DISPOSITIVOS ESTUDADOS

A escolha dos dispositivos estudados se deu a principio pelas suas semelhanças. A popularização de dispositivos computacionais como o Raspberry Pi e os dispositivos da família BeagleBoards, possibilita que diversas soluções em processamento de áudio sejam criadas e recriadas.

A comparação entre os dispositivos fica além da comparação das capacidades de processamento. Cabe destacar o tamanho reduzido, portabilidade, possibilidade de adaptação do

hardware e preço acessível dos dispositivos Raspberry Pi e BeagleBone Black.

Como citado anteriormente, para se obter um parâmetro de comparação com um dispositivo que possibilite o processamento de sinais digitais de áudio em tempo real, um Notebook contendo um processador Intel i5 e 4GB de memória também foi escolhido para a realização dos testes. Esse por sua vez, além da sua disponibilidade, foi escolhido por ter capacidade computacional mediana em relação aos dispositivos encontrados no mercado.

Em se tratando de desempenho no processamento de áudio digital, o Raspberry Pi e o Notebook geraram latências semelhantes, e em alguns testes, o Raspberry Pi se mostrou mais rápido de o Notebook. Apesar de o BeagleBone Black contar com um processador mais robusto que o Raspberry Pi, seu desempenho foi menor que os outros dispositivos, gerando uma latência consideravelmente maior.

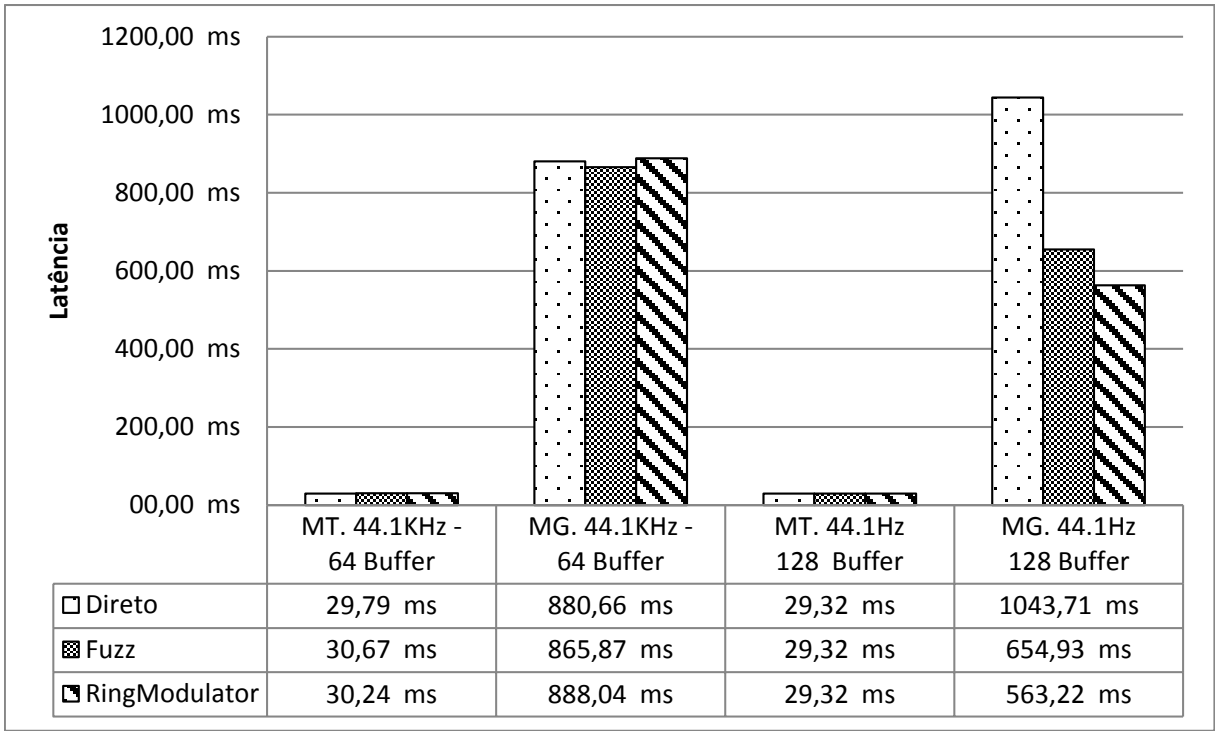
Cabe destacar novamente que todos os dispositivos foram preparados para os testes, e que a não realização de tais preparações geraria prejuízos ao tempo de processamento. Por isso, os testes foram repetidos no Raspberry Pi, porém dessa vez, com o modo gráfico ao invés do modo texto, como realizado anteriormente.

Os testes com o Raspberry Pi em modo gráfico geraram resultados bastante distintos dos testes realizados em modo texto. Os testes de audição iniciais já mostravam que o áudio era prejudicado pelo processamento, gerando um áudio com baixa resolução e muitos ruídos resultantes da falha de processamento, que por sua vez, eram relatadas no log do Pure data. Por conta da baixa qualidade do áudio processado, o programa de medição de latência não conseguiu obter valores exatos, assim como ocorreu com as modificações no áudio impostas pelo filtro Phaser. Os testes foram realizados com configurações usando 44,1KHZ de taxa de amostragem e 64 amostras de tamanho de buffer, e com 44,1KHZ de taxa de amostragem e 128 amostras de tamanho de buffer, conforme apresentado na Figura 18.

Foram também realizados testes com taxas de amostragem maiores (64KHz e 88,2KHz), porém, os filtros não funcionaram com essas configurações, impedindo a obtenção de resultados.

Com a realização dos testes e análise dos dados obtidos no Raspberry Pi em modo gráfico, é possível notar que a liberação do custo computacional imposto pela interface gráfica do Raspbian, gera a possibilidade de que o Pure Data obtenha um melhor desempenho.

O Raspberry Pi possui capacidades de execução de elementos gráficos melhores que o BeagleBone Black, uma das possibilidades, que necessitam de maiores estudos subsequentes, é a utilização da capacidade de processamento gráfico para o processamento de outras atividades,



	MT. 44.1KHz - 64 Buffer	MG. 44.1KHz - 64 Buffer	MT. 44.1Hz 128 Buffer	MG. 44.1Hz 128 Buffer
Direto	29,79 ms	880,66 ms	29,32 ms	1043,71 ms
Fuzz	30,67 ms	865,87 ms	29,32 ms	654,93 ms
RingModulator	30,24 ms	888,04 ms	29,32 ms	563,22 ms

Figura 18: Comparativo entre latências geradas com o Raspberry Pi em modo texto e gráfico
MT = Modo Texto
MG = Modo Gráfico
Buffer = Tamanho do buffer em blocos de amostras
KHz = Taxa de amostragem

como no processamento gerado na execução do Pure Data.

Apesar dos dispositivos Raspberry Pi e BeagleBone Black possibilitarem o processamento de áudio digital em tempo satisfatórios para condições que necessitem de resposta em tempo real, uma das questões de maior impacto negativo aos dispositivo é a usabilidade. Para obter um bom resultado, os dispositivos passaram por configurações com um nível de dificuldade considerável. Além disso, durante o uso, devem funcionar sem interface gráfica, sua utilização nessas condições podem causar dificuldades a pessoas sem conhecimento.

6 CONSIDERAÇÕES FINAIS

Sem dúvidas a utilização de novas tecnologias, bem como a adaptação de dispositivos de uso não especifico ao processamento de áudio, contribuem para o desenvolvimento de novas perspectivas de produção, manipulação e interação de áudio. O uso de dispositivos não específicos para o processamento de áudio, mas que sejam de baixo custo, com fácil aquisição, e uma boa portabilidade está se tornando comum principalmente no mercado de produção musical.

Durante o decorrer do presente trabalho, foi possível identificar diversas possibilidades e limitações dos dispositivos computacionais Raspberry Pi e BeagleBone Black para o processamento de sinais digitais de áudio em tempo real, comparando-os com um dispositivo computacional de maior capacidade.

Ao final do presente trabalho conclui-se que Raspberry Pi e o BeagleBone Black demonstraram, dentro de suas limitações, serem uma alternativa viável para o processamento de áudio digital. Os dispositivos, quando configurados corretamente, apresentam latências que não trazem grandes prejuízos ao processamento de áudio em tempo real, bem como, apresentam pouca interferência na fidelidade do áudio processado em relação ao áudio original.

Além do desempenho satisfatório, o conjunto hardware e software envolvido são de licenças Open Source, possibilitando uma série de vantagens como a possibilidade de melhoria, adaptação e distribuição dos elementos envolvidos no projeto. Cabe destacar também, as inúmeras possibilidades que o software Pure Data trás para o processamento de sinais digitais, possibilitando diversas formas de manipulação de áudio e vídeo através de módulos disponíveis em seu pacote.

Porém, o uso dos dispositivos aqui estudados para o processamento de sinais de áudio digital em tempo real geraram algumas dificuldades e desvantagens. O que mais chama atenção negativamente no uso dos dispositivos é a dificuldade de usabilidade e a falta de recursos para uma maior interação com os dispositivos.

O uso do Raspberry Pi e BeagleBone Black para o processamento de áudio pode ser difícil para usuários sem um conhecimento especifico. Se considerarmos que o maior publico

alvo para tais propósitos aqui apresentados são músicos e profissionais do áudio, as condições de uso pode ser um forte fator á atrapalhar a popularização do usos dos dispositivos para o processamento de áudio. Porém, é possível criar inúmeras formas de se melhorar a interação e usabilidade dos dispositivos para o processamento de áudio digital.

Além das dificuldades geradas pelos dispositivos, o desenvolvimento dos filtros em Pure Data também não são triviais. Apesar da aprendizagem de uso do Pure Data ser simples, o desenvolvimento de filtros requer um aprofundado conhecimento sobre o funcionamento do filtro ou efeito. Com isso, o nível de dificuldade para o desenvolvimento de um programa em Pure Data cresce, exigindo um conhecimento aprofundado na contrição dos filtros desejados.

Apesar das dificuldades encontradas, as experiências obtidas no uso do Raspberry Pi e do BeagleBone Black, unidos ao software Pure Data, para o processamento de sinais digitais de áudio, trazem boas perspectivas para o uso e tais dispositivos para essa função. Existem uma grande quantidade de programas Pure Data já implementados e disponíveis para uso em diversas aplicações na produção do áudio, e o uso dos dispositivos é eficiente e pode criar diversas formas de criação, manipulação e interação com áudio em tempo real.

6.1 TRABALHOS FUTUROS

Durante o decorrer do presente trabalho muitas possibilidades de estudos subsequentes foram percebidas. Dentre elas podemos destacar:

- Desenvolvimento de formas de melhorar a interação e usabilidade dos dispositivos Raspberry Pi e BleagleBone Black para o uso em processamento de sinais digitais em tempo real. Entre as possibilidades, sugerimos a implementação de pedais controladores, softwares para o controle dos dispositivos e softwares envolvidos através de dispositivos móveis, como *smartphones* e *tablets*.

- Estudos comparativo das capacidades da plataforma Pure Data em relação a softwares comerciais.

- Estudos sobre o uso de dispositivos de baixo custo para o desenvolvimento de instrumentos musicais alternativos.

- Estudos sobre o uso do chip gráfico do Raspberry Pi para otimização de processamento em outras tarefas.

REFERÊNCIAS

ALONSO, M.; GEIGER, G.; JORDÀ, S. An internet browser plug-in for real-time audio synthesis. In: . [s.n.], 2004. Disponível em: http://mtg.upf.edu/files/publications/9d0455-WEDEL2004-AlonsoGeigerJorda.pdf.

ANHAIA, P. **Mix in the box**. 2013.

BEAGLEBOARD. **BeagleBoard**. 2013. Acessado em 15 de Setembro de 2013. Disponível em: http://beagleboard.org/.

BEAMISH, N.; O'KEEFFE, C.; VERELLEN, R. **High pass filter**. nov. 27 2007. WO Patent App. PCT/IB2007/054,812.

BERDAHL, W. J. E. Satellite ccrma: A musical interaction and sound synthesis platform. **NIME (International Conference on New Interfaces for Musical Expression)**, v. 1, 2011. Disponível em: https://ccrma.stanford.edu/~eberdahl/Papers/NIME2011SatelliteCCRMA.pdf.

BERDAHL, W. J. E. **Satellite CCRMA**. 2013. Acessado em 16 de Setembro de 2013. Disponível em: https://ccrma.stanford.edu/~eberdahl/Satellite/.

BIANCHI, A. J. **Processamento de áudio em tempo real em dispositivos não convencionais**. Dissertação (Mestrado) — Universidade de São Paulo, 2011.

COLAGIACOMO, V.; PRINCIPI, E.; CIFANI, S.; SQUARTINI, S. Real-time speaker diarization on ti omap3530. In: **Education and Research Conference (EDERC), 2010 4th European**. [S.l.: s.n.], 2010. p. 215–219.

DIGIKEY. **BeagleBone Black**. Setembro 2013. Acessado em 17 Setembro de 2013. Disponível em: http://www.digikey.com/product-highlights/us/en/texas-instruments-beagleboard/685.

DIMITROV, S.; SERAFIN, S. Audio Arduino - an ALSA (Advanced Linux Sound Architecture) Audio Driver for FTDI-based Arduinos. In: **Proceedings of the International Conference on New Interfaces for Musical Expression**. [s.n.], 2011. p. 211–216. Disponível em: http://www.nime.org/proceedings/2011/nime2011_211.pdf.

ELINUX. **BeagleBone**. 2013. Acessado em 15 de Setembro de 2013. Disponível em: http://elinux.org/BeagleBone.

ELINUX. **RPI Hub**. Setembro 2013. Acessado em 8 de Setembro de 2013. Disponível em: http://elinux.org/RPi_Hub.

EMBREE, P. M. **C language algorithms for real-time DSP**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995. ISBN 0-13-337353-3.

FILHO, R. H. de O. Disciplina: Acústica.

FURUTANI, K.; KATO, M.; TSURU, T. **Low-pass filter**. [S.l.]: Google Patents, set. 16 1997. US Patent 5,668,511.

HAYKIN, S. **Digital Communications**. John Wiley And Sons Limited, 1988. ISBN 9780471637752. Disponível em: http://books.google.com.br/books?id=XAPgAQAACAAJ.

LAGO, N. P. **Processamento Distribuído de Audio em Tempo Real**. Dissertação (Mestrado) — Universidade de São Paulo, Abril 2004. Disponível em: http://www.teses.usp.br/teses/disponiveis/45/45134/tde-05102004-154239/pt-br.php.

MACMILLAN, K.; DROETTBOOM, M.; FUJINAGA, I. Audio latency measurements of desktop operating systems. In: SN. **Proceedings of the International Computer Music Conference**. [S.l.], 2001. p. 259–262.

MADEIRO, W. T. A. L. F. Introdução à compressão de sinais. **Revista de Tecnologia da Informação e Comunicação**, v. 1, p. 33–44, Outubro 2011.

MASSAT, P. **Guitar Extended**. 2013. Acessado em 13 de Setembro de 2013. Disponível em: http://guitarextended.wordpress.com/.

MELLO, C. A. Processamento digital de sinais. In: . [S.l.: s.n.], 2013.

MOORE, F. R. **Elements of computer music**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990. ISBN 0-13-252552-6.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. **Discrete-time signal processing (2nd ed.)**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999. ISBN 0-13-754920-2.

OPPENHEIMER, S. e. a. The complete desktop studio. **Electronic Musician**, v. 15, n. 6, p. 48–104, Junho 1999.

PDCOMMUNITY. **Pure Data**. Setembro 2013. Acessado em Setembro de 2013. Disponível em: http://puredata.info/.

PEETERS, G. {A large set of audio features for sound description (similarity and classification) in the CUIDADO project}. 2004.

PIERCE, A. D. **Acoustics: an introduction to its physical principles and applications**. [S.l.]: Acoustical Soc of America, 1989.

PUCKETTE, M. Pure data: another integrated computer music environment. In: **in Proceedings, International Computer Music Conference**. [S.l.: s.n.], 1996. p. 37–41.

PUCKETTE, M. International computer music conference. In: INTERNATIONAL COMPUTER MUSIC ASSOCIATION. **in Proceedings, International Computer Music Conference**. [S.l.], 1998. p. 420–429.

RABINER, L. R.; GOLD, B. Theory and application of digital signal processing. **Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.**, v. 1, 1975.

RASPBERRYPI.ORG. **FAQs. Raspberry Pi Foundation**. 2013. Acessado em 8 de Setembro de 2013. Disponível em: http://www.raspberrypi.org/faqs.

SILVA, M. A. G. da. Filtros digitais aplicados em sinais de audio. v. 1, p. 331, 2007.

SOUSA, S. Amplificadores digitais de audio: Estado da arte. 2008.

TANAKA, N.; SHIMADA, O.; TSUSHIMA, M.; NORIMATSU, T.; CHONG, K.; KUAH, K.; NEO, S.; NOMURA, T.; TAKAMIZAWA, Y.; SERIZAWA, M. **Audio decoding apparatus and method for band expansion with aliasing adjustment**. Google Patents, jun. 27 2006. US Patent 7,069,212. Disponível em: `http://www.google.com/patents/US7069212`.

TORRINI, A.; SHKOLNYY, K. **Audio source system and method**. [S.l.]: Google Patents, jan. 28 2014. US Patent 8,639,370.

WOODRUFF, A.; GÖRMEZ, B. **Laser Music System : Implemented using lasers, infrared sensors, photocells and a Arduino Microcontroller**. 2012. 61 p. Disponível em: `http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-19057`.

ZÖLZER, U. (Ed.). **DAFX: Digital Audio Effects**. second. [S.l.]: Wiley, 2011.

7 ANEXOS

7.1 IMPLEMENTAÇÃO DO PROGRAMA "SEM FILTRO"EM PD

Code 7.1: Implementação do programa Sem filtro em Pure Data

```
#N canvas 1 82 1351 714 10;
#X obj 156 39 adc~;
#X obj 157 121 dac~;
#X obj 19 15 loadbang;
#X msg 16 48 \; pd dsp 1 \;;
#X connect 0 0 1 0;
#X connect 0 0 1 1;
#X connect 2 0 3 0;
```

7.2 IMPLEMENTAÇÃO DO PROGRAMA "FUZZ"EM PD

Code 7.2: Implementação do programa Fuzz em Pure Data

```
#N canvas 0 54 1366 714 10;
#X obj 108 28 adc~;
#X obj 116 241 dac~;
#X obj 107 87 *~ 40;
#X obj 106 138 clip~ -0.5 0.5;
#X text 163 84 <-- Amplify the sound from the guitar;
#X text 225 137 <-- Hard-clip it to produce audible distortion;
#X text 163 29 <-- Read audio data from the soundcard (sound of the
guitar);
#X text 174 242 <-- Send the processed sound back to the soundcard
;
#X obj 701 10 loadbang;
#X msg 698 43 \; pd dsp 1 \;;
#X connect 0 0 2 0;
#X connect 2 0 3 0;
#X connect 3 0 1 0;
```

```
17 #X connect 3 0 1 1;
18 #X connect 8 0 9 0;
```

7.3 IMPLEMENTAÇÃO DO PROGRAMA "PHASER"EM PD

Code 7.3: Implementação do programa Phaser em Pure Data

```
1  #N canvas 376 217 892 516 10;
2  #X obj 80 -71 adc~;
3  #X obj 79 322 dac~;
4  #X obj 500 15 hsl 128 15 0 10 0 0 empty empty empty -2 -8 0 10 -262144
5  -1 -1 5800 1;
6  #X obj 338 122 *~ 2;
7  #X obj 338 142 +~ 2;
8  #X obj 377 164 *~ 2;
9  #X obj 376 187 +~ 2;
10 #X obj 124 96 delwrite~ phase2 100;
11 #X obj 376 213 vd~ phase2;
12 #X obj 377 120 wrap~;
13 #X obj 347 65 phasor~;
14 #X obj 376 99 +~ 0.5;
15 #X obj 338 99 cos~;
16 #X obj 377 140 cos~;
17 #X obj 503 112 hsl 128 15 0 3 0 0 empty empty empty -2 -8 0 10 -262144
18 -1 -1 0 1;
19 #X obj 376 236 *~ 0.8;
20 #X obj 338 297 *~ 0.8;
21 #X obj 447 324 s~ output;
22 #X obj 111 282 r~ output;
23 #X obj 532 231 hsl 128 15 -0.9 0.9 0 0 empty empty empty -2 -8 0 10
24 -262144 -1 -1 0 1;
25 #X text 252 30 <-- Write to delay lines;
26 #X text 448 210 <-- Read from delay line 2;
27 #X text 442 274 <-- Read from delay line 1;
28 #X obj 339 274 vd~ phase1;
29 #X obj 125 36 delwrite~ phase1 100;
30 #X text 410 65 <-- Low Frequency Oscillator (LFO);
31 #X floatatom 497 39 5 0 0 0 - - -;
32 #X floatatom 501 138 5 0 0 0 - - -;
33 #X floatatom 529 252 5 0 0 0 - - -;
34 #X text 640 11 <-- Speed (try 0.7);
35 #X text 645 110 <-- Depth (try 2.4);
```

```
#X text 672 231 <-- Feedback (try -0.3);
#X obj 813 -63 loadbang;
#X msg 810 -30 \; pd dsp 1 \;;
#X connect 0 0 1 0;
#X connect 0 0 1 1;
#X connect 0 0 7 0;
#X connect 0 0 24 0;
#X connect 2 0 10 0;
#X connect 2 0 26 0;
#X connect 3 0 4 0;
#X connect 4 0 23 0;
#X connect 5 0 6 0;
#X connect 6 0 8 0;
#X connect 8 0 15 0;
#X connect 8 0 17 0;
#X connect 9 0 13 0;
#X connect 10 0 11 0;
#X connect 10 0 12 0;
#X connect 11 0 9 0;
#X connect 12 0 3 0;
#X connect 13 0 5 0;
#X connect 14 0 3 1;
#X connect 14 0 5 1;
#X connect 14 0 4 1;
#X connect 14 0 6 1;
#X connect 14 0 27 0;
#X connect 15 0 7 0;
#X connect 16 0 24 0;
#X connect 18 0 1 1;
#X connect 18 0 1 0;
#X connect 19 0 16 1;
#X connect 19 0 15 1;
#X connect 19 0 28 0;
#X connect 23 0 16 0;
#X connect 23 0 17 0;
#X connect 32 0 33 0;
```

7.4 IMPLEMENTAÇÃO DO PROGRAMA "RING MODULATOR"EM PD

Code 7.4: Implementação do programa Ring Modulator em Pure Data

```
#N canvas 1 82 1024 692 10;
```

```
#X obj 30 26 adc~;
#X obj 12 654 dac~;
#X obj 58 561 *~;
#X obj 97 391 osc~;
#X obj 230 477 line~;
#X obj 231 457 pack 0 5;
#X msg 632 39 \; pd dsp 1 \;;
#X obj 632 17 loadbang;
#X obj 96 132 phasor~;
#X obj 93 214 tabread4~ waveform_square;
#X obj 92 237 +~ 0.5;
#X msg 632 149 \; waveform_sine sinesum 2051 1 \; waveform_sine normalize
0.5 \;;
#X obj 802 593 table waveform_square;
#X obj 802 616 table waveform_sine;
#X obj 85 495 *~;
#X obj 207 503 *~;
#X obj 105 470 line~;
#X obj 106 450 pack 0 5;
#X obj 117 428 expr 1-$f1;
#X obj 349 389 tgl 45 0 empty toggle_man_lfo Manual_/_LFO 0 -5 0 10
-4034 -1 -1 1 1;
#X text 673 653 Pierre Massat \, GuitarExtended \, 2012;
#X obj 99 320 *~;
#X obj 98 355 +~;
#X obj 343 240 cnv 15 200 100 empty empty MANUAL 20 12 0 14 -4034 -66577
0;
#X obj 351 287 mtof;
#X obj 354 269 hsl 128 15 0 127 0 0 empty manual_freq empty -2 -8 0
10 -262144 -1 -1 8000 1;
#X obj 351 310 osc~;
#X obj 343 16 cnv 15 200 200 empty empty LFO 20 12 0 14 -260097 -66577
0;
#X obj 354 52 hsl 128 15 0 127 0 0 empty lfo_upper Upper_limit -2 -8
0 10 -262144 -1 -1 5000 1;
#X obj 354 101 hsl 128 15 0 127 0 0 empty lfo_lower Lower_limit -2
-8 0 10 -262144 -1 -1 3000 1;
#X floatatom 351 119 6 0 0 0 - - -;
#X floatatom 351 70 6 0 0 0 - - -;
#X obj 260 156 -;
#X obj 260 134 bang;
#X obj 301 195 mtof;
#X obj 302 131 mtof;
```

```
#X obj 259 175 max 1;
#X obj 286 222 min;
#X obj 157 277 line~;
#X obj 158 257 pack 0 5;
#X obj 146 334 line~;
#X obj 147 314 pack 0 5;
#X obj 353 154 hsl 128 15 0 30 0 0 empty lfo_freq LFO_frequency -2
-8 0 10 -262144 -1 -1 1693 1;
#X msg 350 177 set waveform_square;
#X msg 350 197 set waveform_sine;
#N canvas 0 0 450 300 random_manual 0;
#X obj 163 115 random 120;
#X obj 164 143 + 10;
#X obj 162 236 s manual_freq;
#X obj 162 197 line;
#X obj 178 171 pack 0 0;
#X floatatom 216 216 5 0 0 0 - - -;
#X floatatom 268 194 5 0 0 0 - - -;
#X obj 233 141 + 50;
#X obj 163 91 metro 3000;
#X obj 235 115 random 2950;
#X obj 202 17 inlet;
#X connect 0 0 1 0;
#X connect 1 0 4 0;
#X connect 3 0 5 0;
#X connect 3 0 2 0;
#X connect 4 0 3 0;
#X connect 7 0 4 1;
#X connect 8 0 0 0;
#X connect 8 0 9 0;
#X connect 9 0 6 0;
#X connect 9 0 7 0;
#X connect 10 0 8 0;
#X restore 547 450 pd random_manual;
#N canvas 0 0 607 397 random_LFO 0;
#X obj 61 140 + 10;
#X obj 59 194 line;
#X obj 75 168 pack 0 0;
#X floatatom 113 213 5 0 0 0 - - -;
#X floatatom 165 191 5 0 0 0 - - -;
#X obj 130 138 + 50;
#X obj 132 112 random 2950;
#X obj 233 113 random 120;
```

```
88  #X obj 231 261 line;
89  #X obj 247 235 pack 0 0;
90  #X floatatom 285 280 5 0 0 0 - - -;
91  #X floatatom 338 192 5 0 0 0 - - -;
92  #X obj 303 139 + 50;
93  #X obj 305 113 random 2950;
94  #X obj 231 300 s lfo_lower;
95  #X obj 60 112 random 31;
96  #X obj 59 233 s lfo_freq;
97  #X obj 157 18 inlet;
98  #X obj 383 113 random 120;
99  #X obj 384 141 + 10;
100 #X obj 382 195 line;
101 #X obj 398 169 pack 0 0;
102 #X floatatom 436 214 5 0 0 0 - - -;
103 #X floatatom 488 192 5 0 0 0 - - -;
104 #X obj 453 139 + 50;
105 #X obj 383 89 metro 3000;
106 #X obj 455 113 random 2950;
107 #X obj 382 234 s lfo_upper;
108 #X obj 247 175 *;
109 #X obj 233 132 + 10;
110 #X obj 246 154 / 130;
111 #X obj 249 210 min;
112 #X obj 281 179 bang;
113 #X obj 60 88 metro 2500;
114 #X obj 233 89 metro 2400;
115 #X connect 1 0 3 0;
116 #X connect 1 0 16 0;
117 #X connect 2 0 1 0;
118 #X connect 5 0 2 1;
119 #X connect 6 0 4 0;
120 #X connect 6 0 5 0;
121 #X connect 7 0 29 0;
122 #X connect 8 0 10 0;
123 #X connect 8 0 14 0;
124 #X connect 9 0 8 0;
125 #X connect 12 0 9 1;
126 #X connect 13 0 11 0;
127 #X connect 13 0 12 0;
128 #X connect 15 0 0 0;
129 #X connect 15 0 2 0;
130 #X connect 17 0 25 0;
```

```
131 #X connect 17 0 33 0;
132 #X connect 17 0 34 0;
133 #X connect 18 0 19 0;
134 #X connect 18 0 28 1;
135 #X connect 18 0 31 1;
136 #X connect 18 0 32 0;
137 #X connect 19 0 21 0;
138 #X connect 20 0 22 0;
139 #X connect 20 0 27 0;
140 #X connect 21 0 20 0;
141 #X connect 24 0 21 1;
142 #X connect 25 0 18 0;
143 #X connect 25 0 26 0;
144 #X connect 26 0 23 0;
145 #X connect 26 0 24 0;
146 #X connect 28 0 31 0;
147 #X connect 29 0 30 0;
148 #X connect 30 0 28 0;
149 #X connect 31 0 9 0;
150 #X connect 32 0 31 0;
151 #X connect 33 0 6 0;
152 #X connect 33 0 15 0;
153 #X connect 34 0 7 0;
154 #X connect 34 0 13 0;
155 #X restore 445 450 pd random_LFO;
156 #X obj 446 394 tgl 40 0 empty empty Random_LFO 0 -5 0 10 -258113 -1
157 -1 0 1;
158 #X obj 547 396 tgl 40 0 empty empty Random_Manual 0 -5 0 10 -4160 -1
159 -1 0 1;
160 #N canvas 0 0 612 483 equal_power_crossfade 0;
161 #X obj 33 48 inlet~;
162 #X obj 244 58 inlet;
163 #X obj 33 332 outlet~;
164 #X obj 319 54 loadbang;
165 #X msg 319 74 0.5;
166 #X obj 49 194 expr cos($f1 * 3.14159 / 2);
167 #X obj 244 194 expr sin($f1 * 3.14159 / 2);
168 #X obj 244 134 clip 0 1;
169 #N canvas 0 0 450 300 pan_core 0;
170 #X obj 21 106 inlet~;
171 #X obj 226 126 inlet;
172 #X obj 80 453 *~;
173 #X obj 144 497 outlet~;
```

```
#X obj 210 452 *~;
#X obj 226 428 line~;
#X obj 96 428 line~;
#X obj 96 127 inlet;
#X obj 179 104 inlet~;
#X text 244 544 comment;
#X obj 96 389 pack f 5;
#X obj 226 389 pack f 5;
#X text 223 540 Hans-Christoph Steiner \, 2004;
#X text 223 526 Adapted from [pan_core~];
#X connect 0 0 2 0;
#X connect 1 0 11 0;
#X connect 2 0 3 0;
#X connect 4 0 3 0;
#X connect 5 0 4 1;
#X connect 6 0 2 1;
#X connect 7 0 10 0;
#X connect 8 0 4 0;
#X connect 10 0 6 0;
#X connect 11 0 5 0;
#X restore 39 266 pd pan_core;
#X text 256 437 Adapted from [equal_power_pan~] \, Hans-Christoph Steiner
\, 2004;
#X text 411 449 comment;
#X obj 128 46 inlet~;
#X connect 0 0 8 0;
#X connect 1 0 7 0;
#X connect 3 0 4 0;
#X connect 4 0 7 0;
#X connect 5 0 8 1;
#X connect 6 0 8 3;
#X connect 7 0 5 0;
#X connect 7 0 6 0;
#X connect 8 0 2 0;
#X connect 11 0 8 2;
#X restore 13 610 pd equal_power_crossfade;
#X obj 350 508 hsl 128 25 0 1 0 0 empty crossfade Dry_---_Wet_(crossfade)
-2 -8 0 10 -257985 -1 -1 6350 1;
#X obj 96 152 *~ 2048;
#X obj 96 169 +~ 1;
#X msg 633 75 \; waveform_square sinesum 2051 1 0 0.333333 0 0.2 0
0.142857 0 0.111111 0 0.090909 0 0.076923 0 0.066666 0 0.058823 0 0.052631
0 0.047619 0 0.043478 0 0.04 0 0.037037 \; waveform_square normalize
```

```
0.5 \;;
#X text 175 611 <-- Abstraction to crossfade smoothly between dry and
wet.;
#X text 720 444 <-- Control the patch using;
#X text 744 459 these toggles and sliders.;
#X text 761 17 <-- Create the LFO waveforms and;
#X text 785 32 initialize a bunch of parameters.;
#X msg 632 201 \; manual_freq 80 \; lfo_freq 4 \; lfo_upper 50 \; lfo_lower
30 \; toggle_man_lfo 1 \; crossfade 0.5 \;;
#X connect 0 0 2 0;
#X connect 0 0 49 0;
#X connect 2 0 49 1;
#X connect 3 0 14 0;
#X connect 4 0 15 1;
#X connect 5 0 4 0;
#X connect 7 0 6 0;
#X connect 7 0 53 0;
#X connect 7 0 11 0;
#X connect 7 0 59 0;
#X connect 8 0 51 0;
#X connect 9 0 10 0;
#X connect 10 0 21 0;
#X connect 14 0 2 1;
#X connect 15 0 2 1;
#X connect 16 0 14 1;
#X connect 17 0 16 0;
#X connect 18 0 17 0;
#X connect 19 0 5 0;
#X connect 19 0 18 0;
#X connect 21 0 22 0;
#X connect 22 0 3 0;
#X connect 24 0 26 0;
#X connect 25 0 24 0;
#X connect 26 0 15 0;
#X connect 28 0 35 0;
#X connect 29 0 34 0;
#X connect 32 0 36 0;
#X connect 33 0 32 0;
#X connect 34 0 32 1;
#X connect 34 0 33 0;
#X connect 34 0 37 0;
#X connect 35 0 32 0;
#X connect 35 0 37 1;
```

```
#X connect 36 0 31 0;
#X connect 36 0 39 0;
#X connect 37 0 30 0;
#X connect 37 0 41 0;
#X connect 38 0 21 1;
#X connect 39 0 38 0;
#X connect 40 0 22 1;
#X connect 41 0 40 0;
#X connect 42 0 8 0;
#X connect 43 0 9 0;
#X connect 44 0 9 0;
#X connect 47 0 46 0;
#X connect 48 0 45 0;
#X connect 49 0 1 0;
#X connect 49 0 1 1;
#X connect 50 0 49 2;
#X connect 51 0 52 0;
#X connect 52 0 9 0;
```