

FIA 2020/22

XII CONGRESSO/CONGRESO IBEROAMERICANO DE ACÚSTICA
XXIX ENCONTRO DA SOCIEDADE BRASILEIRA DE ACÚSTICA - SOBRAC

Florianópolis, SC, Brasil

Implementação do Efeito Wah Wah utilizando Arduino e PedalShield

Vieira, J. P.¹; Gontijo, W. A.²; Noceti Filho, S.³; Westphal, C. B.⁴

¹⁻³ LINSE - Laboratório de Circuitos e Processamento de Sinais, Departamento de Engenharia Elétrica e Eletrônica, Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil, {jpvieira, walter, sidnei} @linse.ufsc.br

⁴ Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil, carolinawestphal@gmail.com

Resumo

Um efeito de áudio é definido como a manipulação de sinais de áudio, ferramenta muito utilizada nas áreas de música e cinema. Os parâmetros controlados podem ser amplitude, frequência, fase e conteúdo espectral, que variam de acordo com o tipo de efeito. A finalidade é tornar o som reproduzido mais interessante e diferenciado. O efeito Wah Wah imita o som feito pela boca quando alguém tenta pronunciar seu nome, sendo principalmente usado por guitarristas. É obtido ao variar o conteúdo espectral do sinal de uma guitarra utilizando um filtro passa-baixa com alto fator de qualidade, um filtro seletor passa-faixa ou um equalizador bump (peaking filter). A implementação analógica apresenta um custo relativamente alto devido à necessidade do uso de um indutor com alto valor de indutância, de um potenciômetro especial e de um pedal que deve ser acionado pelo guitarrista. Na implementação digital proposta, o controle do espectro é realizado por um oscilador de baixa frequência (LFO) (caracterizando um Auto Wah Wah), evitando o uso de um potenciômetro acionado por um pedal. Os parâmetros controlados são o fator de qualidade Q ; as frequências mínima e máxima onde ocorrem os picos de amplitude do filtro e os níveis dos sinais original e processado em um Mixer. Este trabalho apresenta a implementação digital do efeito Auto Wah Wah. Inicialmente, o efeito é simulado no Matlab para avaliação e testes. Na sequência, o efeito é implementado no conjunto de hardware formado pelo Arduino Due e PedalShield para avaliação em tempo real. A escolha de tal conjunto se justifica pela disseminação do Arduino no desenvolvimento de diversas aplicações, bem como pelos recursos computacionais disponíveis no Due.

Palavras-chave: Arduino Due, efeito de áudio, implementação em tempo real, PedalShield, Wah Wah.

PACS: 43.05.-k, 43.75.-z, 43.75.Tv.

Implementation of the Wah Wah effect utilizing Arduino and PedalShield

Abstract

An audio effect is defined as the manipulation of audio signals, a tool widely used in the field of music and cinema. The controlled parameters can be amplitude, frequency, phase and spectral content, which vary according to the type of effect. The purpose is to make the reproduced sound more interesting and differentiated. The Wah Wah effect imitates the sound made by the mouth when someone tries to pronounce its name, being mainly used by guitarists. It is obtained by varying the spectral content of a guitar signal using a low-pass filter with a high quality factor, a band-pass selector filter or a bump equalizer (peaking filter). The analogic implementation is relatively costly because of the need to use an inductor with high inductance value, a special potentiometer and a pedal that must be operated by the guitarist. In the proposed digital implementation, the spectral control is performed by a low frequency oscillator (LFO) (characterizing an Auto Wah Wah) avoiding the use of a potentiometer driven by a pedal. The controlled parameters are the quality factor Q ; the minimum and maximum frequencies where the filter peaks of amplitude occur and the levels of the original and processed signals in a mixer. This paper presents a digital implementation of the Auto Wah Wah effect. Initially, the effect is simulated in Matlab for evaluation and testing. Next, the effect is implemented in the hardware set formed by Arduino Due and PedalShield for real-time evaluation. The choice of this hardware set is justified by the dissemination of Arduino in the development of various applications, as well as the computational resources available in the Due.

Keywords: Arduino Due, audio effect, real time implementation, PedalShield, Wah Wah.



1. INTRODUÇÃO

Os efeitos de áudio são amplamente usados por músicos nas suas formas analógica e digital. Na forma analógica, alguns efeitos são produzidos por circuitos analógicos contínuos e outros por circuitos analógicos amostrados [1]. Até recentemente, as implementações de efeitos de áudio na forma digital faziam uso de DSP [2]. Atualmente, devido à facilidade, tais implementações podem ser realizadas com o ambiente Arduino [3]. A obtenção dos efeitos consiste em variar, ao longo do tempo, características como amplitude, fase e/ou frequência, conteúdo espectral ou ainda uma combinação de várias dessas características. O efeito Wah Wah imita o som feito pela boca quando alguém tenta pronunciar seu nome, sendo principalmente usado por guitarristas. É obtido ao variar o conteúdo espectral de um sinal de entrada utilizando um filtro passa-baixa com alto fator de qualidade, um filtro seletor passa-faixa ou um equalizador bump (peaking filter) [4]. Neste artigo, é proposta a implementação digital do Wah Wah, com a utilização de um filtro passa-faixa e a variação do espectro sendo realizada por um oscilador de baixa frequência (LFO) (caracterizando um Auto Wah Wah). Inicialmente, o efeito é implementado e simulado em Matlab para avaliação de desempenho e testes com diferentes sinais de entrada. Na sequência, o algoritmo do Wah Wah é convertido para a linguagem “C” para a execução em tempo real no ambiente Arduino Due e PedalShield.

2. EFEITO DE ÁUDIO: WAH WAH

A Figura 1 apresenta o diagrama de blocos da implementação digital do Wah Wah considerada neste artigo.

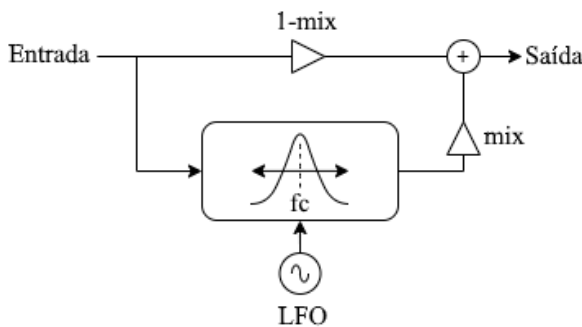


Figura 1: Diagrama de blocos para um Auto Wah Wah.

Observa-se em tal figura a presença de um filtro passa-faixa cuja frequência central é variada por um LFO. Observa-se ainda a presença de um fator “mix”, que permite a ponderação entre o sinal de entrada e a saída do efeito.

A função de transferência no domínio “S” de um filtro passa-faixa de segunda ordem é dada pela Equação (1)[5]

$$T(s) = \frac{Bs}{s^2 + Bs + W}, \quad (1)$$

em que os coeficientes B e W, dados por $B = \frac{\omega_0}{Q}$, $W = \omega_0^2$, dependem da frequência angular central ω_0 e do fator de qualidade Q do filtro, que expressa a razão entre a frequência central f_c e a banda B.

Aplicando a transformação bilinear, a correspondente função de transferência no domínio “Z” é dada pela Equação (2)

$$T(z) = \frac{BF(1 - z^{-2})}{[(F^2 + BF + W) + (2W - 2F^2)z^{-1} + (F^2 - BF + W)z^{-2}]}, \quad (2)$$

em que $F = 2f_s$ (f_s representa a frequência de amostragem).

Aplicando as propriedades da transformada Z, obtém-se a equação de diferença para a saída do filtro passa-faixa, que é apresentado na Equação (3)

$$y[n] = \frac{\frac{2}{Q}\omega'_0[n](x[n] - x[n-2]) - (2\omega'_0[n]^2 - 8)y[n-1] - (4 - \frac{2}{Q}\omega'_0[n] + \omega'_0[n]^2)y[n-2]}{4 + \frac{2}{Q}\omega'_0[n] + \omega'_0[n]^2}}, \quad (3)$$

em que $\omega'_0[n] = \frac{\omega_0[n]}{f_s}$ é a frequência (angular) central variável do filtro passa-faixa, normalizada por f_s .

A frequência angular $\omega_0[n]$ é obtida a partir da Equação (4)

$$\omega_0[n] = 2\pi \frac{f_c[n]}{f_s}, \quad (4)$$

que depende dos valores da frequência central $f_c[n]$, definidos pela Equação (5)

$$f_c[n] = \frac{f_{min} + f_{max}}{2} + \frac{f_{min} - f_{max}}{2} \sin(2\pi f_{LFO} \frac{n}{f_s}), \quad (5)$$

que relaciona os parâmetros f_{min} , f_{max} e f_{LFO} . Tais parâmetros definem a faixa correspondente da frequência central, de um valor mínimo f_{min} a um máximo f_{max} , em uma frequência de oscilação f_{LFO} .

3. HARDWARE UTILIZADO

Para a implementação do efeito em tempo real, foi utilizada a plataforma de prototipagem Arduino, em conjunto com a placa PedalShield.

3.1 Arduino

Arduino é uma plataforma “open-source” de hardware e software utilizada em projetos eletrônicos [6]. Normalmente, o hardware utiliza microcontroladores ATMELE e são disponibilizadas diferentes placas de desenvolvimento, por exemplo, Uno, Mini, Mega, Due e etc. [7]. Para o desenvolvimento de software, pode-se usar o IDE do Arduino [8] ou um compilador C, C++, por exemplo, Atmel Studio. Além das placas principais, há também um extenso conjunto de placas filhas (denominadas *Shields*) que permite a utilização da plataforma Arduino em diferentes áreas de aplicação. Para a implementação dos efeitos de áudio considerados nesse artigo, são utilizados o Arduino Due [9] (apresentado na Figura 2), a *shield* PedalShield [10] e o IDE do Arduino [8].

3.2 Arduino Due

O kit Arduino Due é o primeiro a utilizar um microcontrolador ARM de 32 bits e suas principais características são apresentadas na Tabela 1.

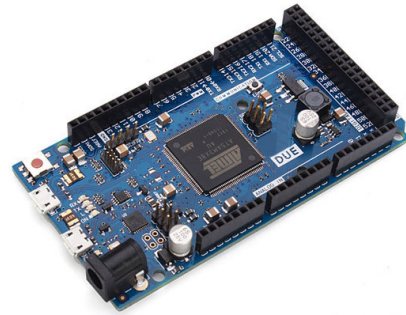


Figura 2: Kit Arduino Due (retirado de [9]).

Microcontrolador	ATMEL AT91SAM3X8E (ARM Cortex M3);
Frequência de clock	84 MHz
Pinos de E/S digital	54 (12 desses podem ser configurados como saídas PWM)
Pinos de Entrada Analógica	12
Pinos de Saída Analógica	2
Resolução do AD	12 bits
Frequência de Amostragem	8 kHz, 16 kHz, 32 kHz, 44,1 kHz e etc.
UARTs	4
Micro-usb	2
SPI	1
JTAG	1
Tensão de Alimentação e E/S	3,3 V
Memória Flash	512 kB disponíveis para o usuário
SRAM	96 kB (dois bancos: 64 kB e 32 kB)

Tabela 1: Principais Características do Arduino Due (retirada de [4]).

Devido à maior capacidade de processamento e recursos de memória do ARM Cortex M3, o Arduino Due suporta a implementação de algoritmos de processamento digital de sinais, como filtros digitais e efeitos de áudio.

Os principais sinais usados na interligação entre o Arduino Due e a PedalShield são mostrados na Figura 3.

Observa-se na Figura 3 que a PedalShield possui os estágios de entrada e saída para a conexão com

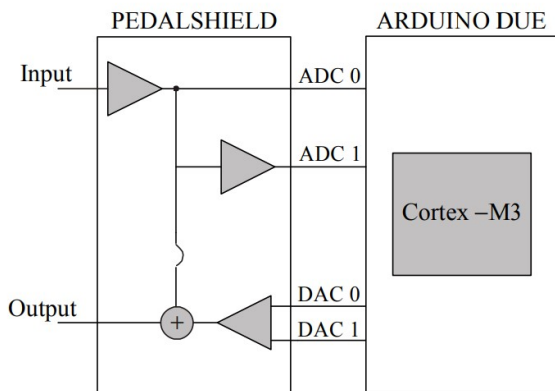


Figura 3: Conexão entre a PedalShield e o Arduino Due (retirado de [4]).

o Arduino Due. O estágio de entrada amplifica o sinal analógico recebido na entrada e gera saídas balanceadas que são enviadas para os conversores AD (ADC0 e ADC1) do Arduino Due. Já o estágio de saída permite que o Arduino Due externe as saídas analógicas balanceadas dos conversores DA (DAC0 e DAC1), sendo que essas saídas podem ser somadas à entrada original.

3.3 PedalShield Due

A Figura 4 mostra os principais componentes de *hardware* da PedalShield Due. Observa-se em tal figura a presença de potenciômetros que permitem ao usuário o controle de parâmetros do programa executado no Arduino. Observa-se ainda as conexões de entrada e saída de tipo P10, além de *switches* [10].

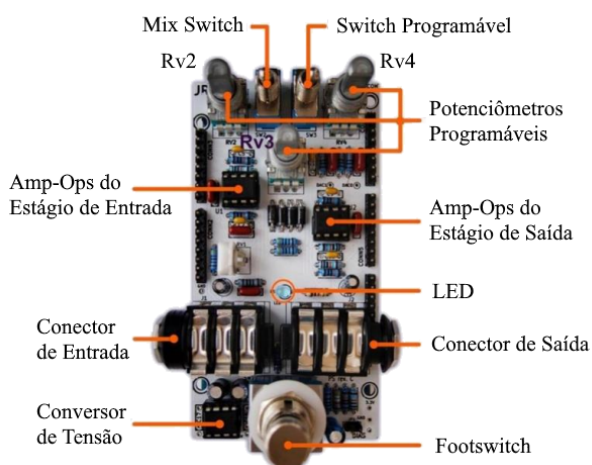


Figura 4: Placa PedalShield Due (adaptado de [4]).

4. IMPLEMENTAÇÃO

Inicialmente, o algoritmo do Wah Wah é implementado em Matlab para avaliação e testes de desempenho. O algoritmo é excitado com diferentes sinais de entrada com as respectivas saídas gravadas. A Figura 5 mostra o procedimento utilizado na implementação e testes realizados com o Matlab.

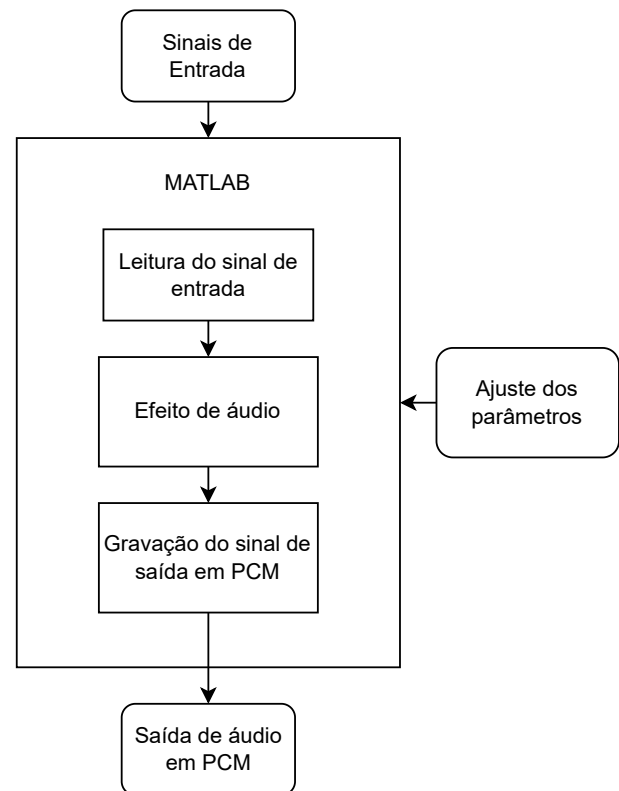


Figura 5: Processo de avaliação do algoritmo em MATLAB (adaptado de [4]).

Na sequência, o algoritmo é convertido para ser executado em tempo real no conjunto Arduino Due e PedalShield. Basicamente, tal conversão consiste na redefinição dos tipos de variáveis e na utilização de uma biblioteca de ponto fixo para as operações matemáticas, a FixedPointsArduino [11]. A partir da inclusão de tal biblioteca, as operações de soma, multiplicação e divisão são redefinidas e passam a operar com variáveis em ponto fixo, acelerando os cálculos. O tipo de variável escolhida para este trabalho é da forma "SQ15x16", em que "S" representa 1 bit de sinal e "Q" define o número com 15 bits para a parte inteira e 16 bits para a parte fracionária.

O algoritmo é excitado com os mesmos sinais uti-

lizados nos testes em Matlab. As respectivas saídas do Arduino Due são gravadas para posterior comparação com os resultados obtidos em Matlab. A Figura 6 mostra o procedimento utilizado nos testes realizados com o conjunto Arduino Due e PedalShield.

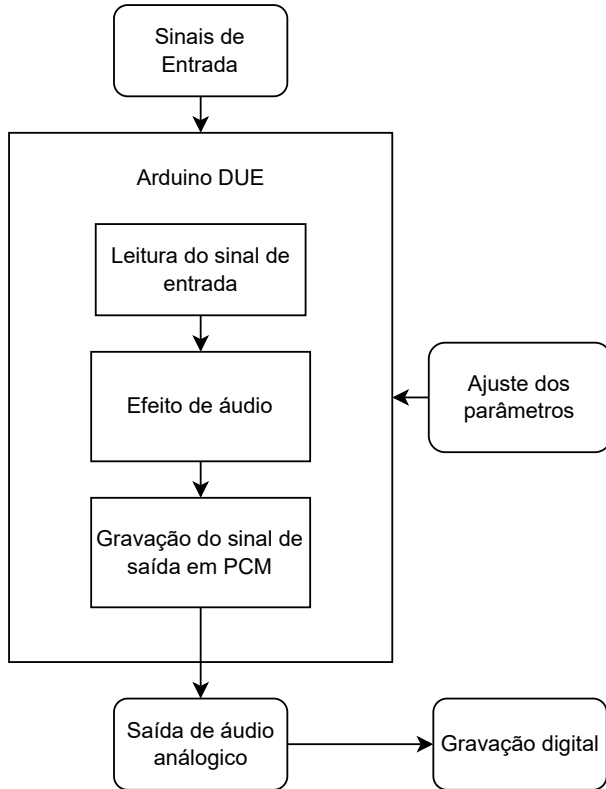


Figura 6: Processo de avaliação do algoritmo no conjunto Arduino Due e PedalShield (adaptado de [4]).

4.1 Implementação em MATLAB

A implementação em Matlab segue os procedimentos mostrados na Figura 5. Inicialmente, são definidos os valores das variáveis utilizadas na Equação (5), como f_{min} , f_{max} , f_{LFO} e f_s . A faixa escolhida para operação do efeito varia entre 60 Hz (f_{min}) e 3500 Hz (f_{max}), e a f_s é 16 kHz. Na simulação, o valor escolhido para a frequência do LFO é 1 Hz.

Na sequência, é realizada a leitura de um sinal de entrada e o processamento amostra a amostra via Equação (3). A cada amostra, o valor de $\omega_0[n]$ (dado na Equação (4)) é atualizado, conferindo a variação da frequência de corte. Finalmente, a saída é ponderada com a entrada através do fator mix e externalizada em PCM para avaliação.

4.2 Implementação em C

Inicialmente, para a conversão do código em Matlab para C, são redefinidos os tipos de variáveis (de *float* para ponto fixo) e incluída a biblioteca FixedPointsArduino [11]. A Tabela 2 mostra as principais modificações realizadas.

<i>float</i>	Ponto fixo
#include math.h	#include <FixedPoints.h>
float A	SQ15x16 A
float B	SQ15x16 B
float Y	SQ15x16 Y
Y = A+B	Y = A+B
	SQ15x16 Y + (SQ15x16 A, SQ15x16 B)
Y = A*B	Y = A*B
	SQ15x16 Y * (SQ15x16 A, SQ15x16 B)

Tabela 2: Principais modificações na conversão de *float* para ponto fixo.

Deve-se comentar ainda que foram realizadas configurações no *timer* do Arduino Due para que a frequência de amostragem opere em 16 kHz. Os códigos em Matlab e C estão disponíveis na plataforma Github [12].

4.3 Controle do Usuário

Na implementação considerada, o usuário pode atuar sobre o Wah Wah via um *footswitch* e potenciômetros, mostrados na Figura 4. O *footswitch* permite ligar ou desligar o efeito, enquanto os potenciômetros possibilitam variar a frequência do LFO, o fator de qualidade Q, o fator *mix*, e as frequências f_{min} e f_{max} . A Tabela 3 mostra a faixa de valores disponíveis para cada parâmetro citado.

Parâmetro	Faixa
mix	0 : 1
Q	1 : 4
f_{LFO} (Hz)	1 : 2,5
f_{min} (Hz)	100 : 300
f_{max} (Hz)	2500 : 3500

Tabela 3: Faixa de valores para os parâmetros do efeito.

5. RESULTADOS

Inicialmente, são apresentados os resultados da implementação em Matlab que mostram o fun-

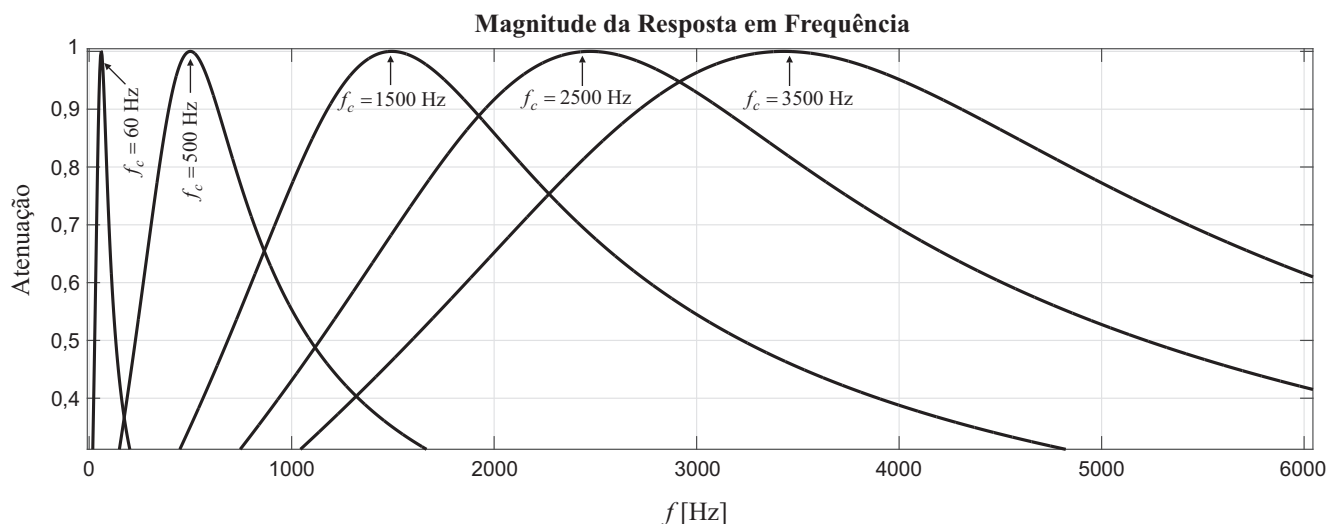


Figura 7: Resposta em frequência do passa-faixa para alguns valores de f_c .

cionamento do Wah Wah. Na sequência, é apresentada uma comparação entre os tempos de execução do algoritmo com a customização para o Arduino Due. Finalmente, é apresentada a comparação entre os resultados obtidos na simulação em Matlab e em tempo real no conjunto Arduino Due e PedalShield.

5.1 Resultados em Matlab

Na Figura 7, é apresentada a resposta em frequência do filtro passa-faixa para alguns valores de f_c (60 Hz, 500 Hz, 1500 Hz, 2500 Hz, 3500 Hz), entre f_{min} e f_{max} . Observa-se que a banda de atuação do filtro se torna cada vez maior à medida que f_c aumenta.

Para avaliar o desempenho do efeito é utilizado na entrada um ruído branco gaussiano. Nas Figuras 8 e 9, são mostradas, respectivamente, o espectrograma da entrada e saída da simulação. Observa-se na Figura (9) a variação senoidal esperada.

5.2 Análise de Tempo

Para avaliar o tempo de execução do algoritmo no Arduino Due, é utilizada a função *micros()* [13], presente no Arduino IDE. Na Tabela 4, são mostrados os tempos de execução do código em C utilizando notação em ponto flutuante e ponto fixo. Observa-se em tal tabela que o tempo de execução em ponto flutuante não atende ao requisito de tempo real ($f_s = 16$ kHz, $t_s = 62,5$ μ s). Entretanto, com as customizações realizadas, o código resultante em ponto fixo atende a tal restrição.

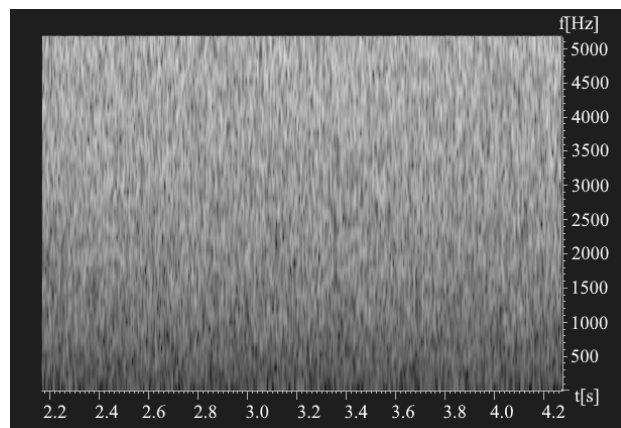


Figura 8: Espectrograma do ruído branco gaussiano.

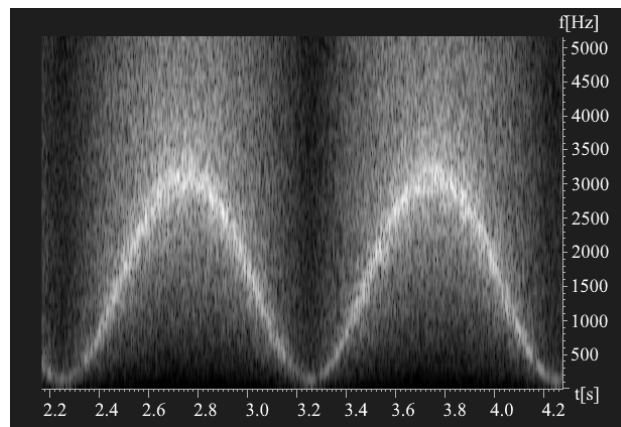


Figura 9: Espectrograma de saída do Wah Wah para uma entrada de ruído branco - simulação Matlab.

Notação	Tempo
Ponto Flutuante	185 μ s
Ponto Fixo	20 μ s

Tabela 4: Tempos de operação para as notações numéricas de ponto fixo e ponto flutuante.

5.3 Comparação entre os resultados

Na Figura 10, é apresentado o espectrograma da saída do conjunto Arduino Due e PedalShield quando excitado por um ruído branco. Comparando-se este resultado com o obtido na simulação em Matlab (Figura 9), observa-se equivalência entre as respostas em frequência.

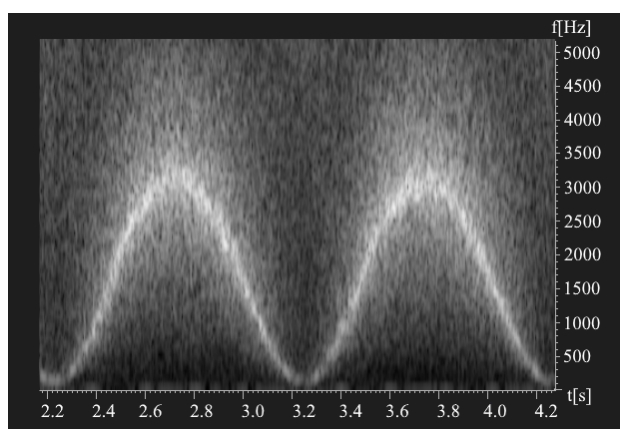


Figura 10: Espectrograma de saída do Wah Wah para uma entrada ruído branco - Arduino Due / PedalShield.

6. CONSIDERAÇÕES FINAIS

Neste artigo, é apresentada a implementação e execução em tempo real do efeito Wah Wah no conjunto Arduino Due e PedalShield. A implementação considera um filtro passa-faixa com frequência central variada por um LFO. Inicialmente, o efeito é implementado em Matlab, para validação e testes de desempenho. Na sequência, o algoritmo é customizado e convertido em ponto fixo para o Arduino Due. Os resultados obtidos mostram o correto funcionamento do algoritmo quando executado em tempo real no conjunto Arduino Due e PedalShield.

REFERÊNCIAS

- [1] Noceti Filho, Sidnei; Gontijo, Walter A. e Dalcagnê, André L. Filtros e efeitos de áudio analógicos e digitais. Em *Anais do 9o Congresso de Engenharia de Áudio da AES Brasil (AES Brasil 2011)*, págs. 12–37, Florianópolis, SC, 2011.
- [2] Hunter, Dave. *Guitar Effects Pedals the Practical*

Handbook. Backbeat Book, 2004.

- [3] <http://arduino.cc>.
- [4] Pavei, Andrew H.; Gontijo, Walter A. e Noceti Filho, Sidnei. Implementações de efeitos de Áudio utilizando arduino due e pedalshield. Em *Anais do 14o Congresso de Engenharia de Áudio da AES Brasil (AES Brasil 2016)*, págs. 81–89, Florianópolis, SC, 2016.
- [5] Westphal, Carolina B. *Efeitos de áudio digitais: Implementação do efeito Wah-wah utilizando Arduino e Pedalshiled*. Trabalho de conclusão de curso, Universidade Federal de Santa Catarina, Florianópolis, SC, 2021.
- [6] <https://www.arduino.cc/en/guide>.
- [7] <https://www.arduino.cc/en/hardware#boards>.
- [8] <https://www.arduino.cc/en/software>.
- [9] <https://store.arduino.cc/products/arduino-due>.
- [10] <https://www.electromash.com/pedalshield>.
- [11] <https://github.com/Pharap/FixedPointsArduino>.
- [12] <https://github.com/jpvieir/mfold-autowahwah>.
- [13] <https://www.arduino.cc/reference/pt/language/functions/time/micros/>.