Relatório Final
EEL7813 - Projeto Nível I em Eletrônica I

Aluno: Daniel Gomes de Pinho Zanco - 13100574
Professor: Sidnei Noceti Filho

1 de Dezembro de 2017

1 Descrição do Projeto

O objetivo deste projeto é desenvolver um pedal de efeitos de áudio do tipo *Auto-Wah*. Esse efeito é baseado no *Wah-Wah*, que consiste de um filtro passa-banda com frequência de corte variável, controlada pelo usuário através de um pedal. No *Auto-Wah*, a variação da frequência de corte é dada por um *LFO* ou pelo nível de amplitude do sinal de entrada. Neste projeto optou-se pelo controle através da amplitude do sinal. O código fonte do trabalho realizado está publicado no Github [1].

O efeito foi implementado de forma digital em um sistema composto de um *Arduino Due* e um *PedalSHIELD*. Na Figura 1, é mostrado um diagrama de blocos do sistema a ser implementado.

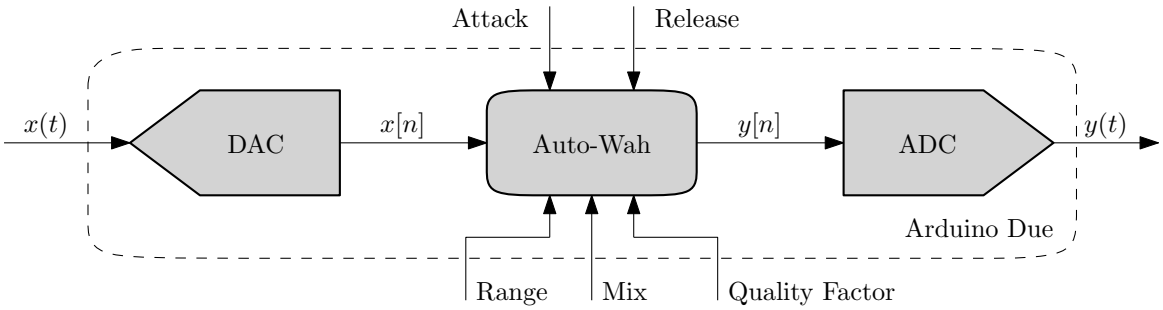


Figura 1: Diagrama de blocos do sistema.

É possível controlar 5 parâmetros através da chave e dos potenciômetros da *PedalSHIELD*: *Attack*, *Release*, *Range*, *Resonance* e *Mixing*. Na Tabela 1 são mostradas as posições de chave e as atribuições de cada potenciômetro para realizar o controles desses parêmetros.

Parâmetro	Estado da chave	Potênciômetro
Attack	Ligada	Esquerdo
Release	Ligada	Direito
Range	Desligada	Esquerdo
Quality Factor	Desligada	Direito
Mix	-	Central

Tabela 1: Acesso aos parâmetros na *PedalSHIELD*.

[1] Disponível em https://github.com/dangpzanco/autowah

2 Estrutura geral

Na Figura 2, é mostrada a estrutura básica do efeito a ser realizado no *Arduino*. Os diferentes blocos dos efeito foram implementados em MATLAB. A implementação de cada bloco e do efeito completo é mostrado nas seguintes subseções.

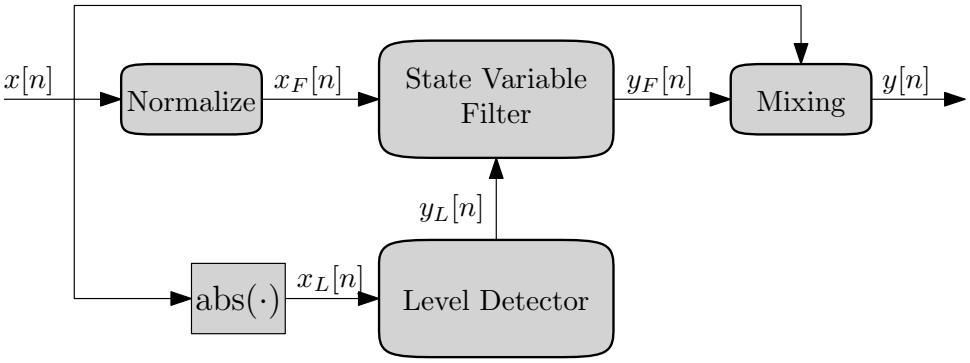


Figura 2: Diagrama de blocos do efeito *Auto-Wah*.

2.1 *Level Detector*

O bloco *Level Detector* foi implementado seguindo [1]. Sejam $x_L[n]$ e $y_L[n]$, respectivamente, a entrada e a saída do *Level Detector*, as equações a seguir definem um detector de pico desacoplado

$$x_L[n] = |x[n]| \tag{1}$$
$$y_1[n] = \max(x_L[n], \alpha_R y_1[n-1] + (1-\alpha_R)x_L[n]) \tag{2}$$
$$y_L[n] = \alpha_A y_L[n-1] + (1-\alpha_A)y_1[n] \tag{3}$$

onde α_A e α_R são os parâmetros de *Attack* e *Release*, respectivamente.

Também é importante notar a relação entre α_A e α_R e suas constantes de tempo respectivas τ_A e τ_R, definida pela seguinte equação

$$\alpha_A = e^{-1/(\tau_A f_s)},\ \alpha_R = e^{-1/(\tau_R f_s)}. \tag{4}$$

2.2 *State Variable Filter*

Seguindo [2], a estrutura de filtragem implementada combina filtros passa-baixa ($y_{lp}[n]$), passa-banda ($y_{bp}[n]$) e passa-alta ($y_{hp}[n]$) para uma mesma frequência central (ou de corte) f_c e fator de qualidade Q. As equações de diferenças são dadas por

$$y_{lp}[n] = f[n]y_{bp}[n] + y_{lp}[n-1] \tag{5}$$
$$y_{bp}[n] = f[n]y_{hp}[n] + y_{bp}[n-1] \tag{6}$$
$$y_{hp}[n] = x_F[n] - y_{lp}[n-1] - qy_{bp}[n-1] \tag{7}$$

onde $q = 1/Q$ é o parâmetro de *Quality Factor* e

$$f[n] = 2\sin\left(\pi\frac{f_c[n]}{f_s}\right). \tag{8}$$

A frequência central $f_c[n]$ é calculada a partir da saída do *Level Detector* pela simples relação linear

$$f_c[n] = y_L[n]B_f + f_{min} \tag{9}$$

onde B_f e f_{min} são o parâmetro *Range* e a frequências central mínima, respectivamente. Tipicamente, $f_{min} = 20$ Hz.

Além disso, $x_F[n]$ e $y_F[n] = y_{bp}[n]$ são, respectivamente, a entrada e a saída do *State Variable Filter*.

2.2.1 Normalização do filtro

Em aplicações práticas, existe a necessidade de compensar variações no ganho desse filtro. Pode ocorrer saturação caso o ganho seja muito elevado. Desta forma, foi utilizado um esquema de normalização L_2 sugerido por [2].

A normalização consiste em inserir um filtro passa-baixa de primeira ordem com frequência de corte de $f_c[n]$ e um ganho de $\sqrt{\zeta} = \sqrt{1/(2Q)} = \sqrt{q/2}$ na entrada do *State Variable Filter*. Na Figura 2, este é o bloco *Normalize*.

As equações de diferenças desse filtro são dadas por

$$x_N[n] = x[n] - a_1 x_N[n-1] \tag{10}$$
$$x_F[n] = b_0 x_N[n] + b_1 x_N[n-1] \tag{11}$$

onde

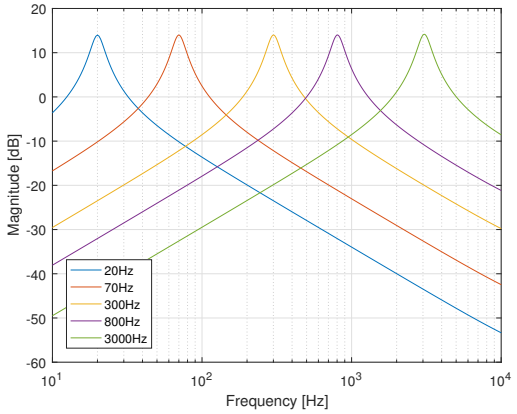
$$b_0 = b_1 = \frac{\sqrt{0.5q}K}{K+1}, \tag{12}$$
$$a_0 = \frac{K-1}{K+1}, \tag{13}$$
$$K = \tan\left(\pi \frac{f_c[n]}{f_s}\right). \tag{14}$$

2.3 *Mixing*

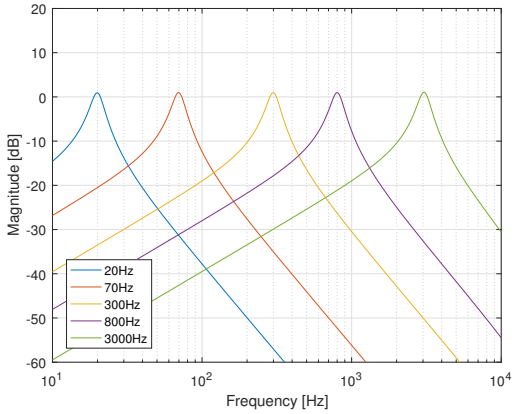
Por fim, é feita uma média ponderada entre a entrada e o sinal processado pelo efeito para realizar um *Mixing*

$$y[n] = \alpha_{mix} y_F[n] + (1 - \alpha_{mix}) x[n] \tag{15}$$

onde α_{mix} é o parâmetro *Mix*.

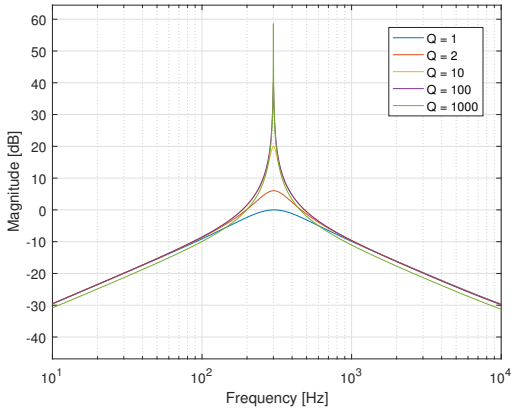


(a) Resposta em frequência do *State Variable Filter* com $Q = 5$.

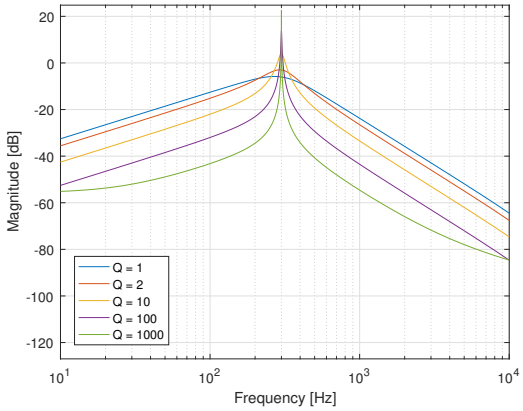


(b) Resposta em frequência do conjunto *Normalize* + *State Variable Filter* com $Q = 5$.

Figura 3: Efeito da variação do parâmetro f_c.



(a) Resposta em frequência do *State Variable Filter* com $f_c = 300$ Hz.



(b) Resposta em frequência do conjunto *Normalize* + *State Variable Filter* com $f_c = 300$ Hz.

Figura 4: Efeito da variação do parâmetro Q.

3 Validação em MATLAB

Para validar o efeito, foram realizadas avaliações dos blocos *Normalize*, *State Variable Filter* e *Level Detector*.

O resposta em frequência do *State Variable Filter* foi comparada com a do conjunto *Normalize* + *State Variable Filter*, de forma a avaliar os efeitos da variação dos parâmetros f_c e Q. A entrada $x[n]$ utilizada foi um impulso discreto e está na taxa de amostragem de $f_s = 48$ kHz.

Na Figura 3 são mostrados os efeitos da variação do parâmetro f_c na resposta em frequência do (a) filtro e do (b) filtro com normalização. Pode-se notar que a resposta do filtro se modifica adequadamente com a variação da frequência central f_c. Entretanto, no caso do filtro sem normalização, há ganhos elevados próximo à frequência central, como visto na Figura 3a. Isso não acontece no filtro com normalização, visto que o ganho máximo é próximo de 0 dB, como visto na Figura 3b. No filtro normalizado, a queda mais acentuada da resposta em frequências mais altas (acima de f_c) não é um problema nessa aplicação, pois depende de uma avaliação subjetiva do usuário do efeito.

Na Figura 4 são mostrados os efeitos da variação do parâmetro Q na resposta em frequência do (a) filtro e do (b) filtro com normalização. Na resposta do filtro sem normalização, grandes valores de Q causam ganhos elevados, como visto na Figura 4a. Esse efeito é mitigado satisfatoriamente no filtro normalizado, cujo ganho só começa a ficar elevado a partir de $Q = 10$, como visto na Figura 4b.

Para avaliar o bloco *Level Detector*, a entrada $x[n]$ utilizada foi um ruído branco com distribuição normal com a amplitude modulada por uma onda triangular de baixa frequência. O sinal $x[n]$ foi normalizado para o intervalo $[-1, 1]$ e está na taxa de amostragem de $f_s = 48$ kHz. Foram utilizados os parâmetros $\tau_A = 20$ ms e $\tau_R = 100$ ms. A análise realizada pode ser vista na Figura 5, que mostra a envoltória do sinal de entrada $x[n]$ e a envoltória estimada pelo *Level Detector* $y_L[n]$. Nota-se que o *Level Detector* estima adequadamente o nível de amplitude do sinal.

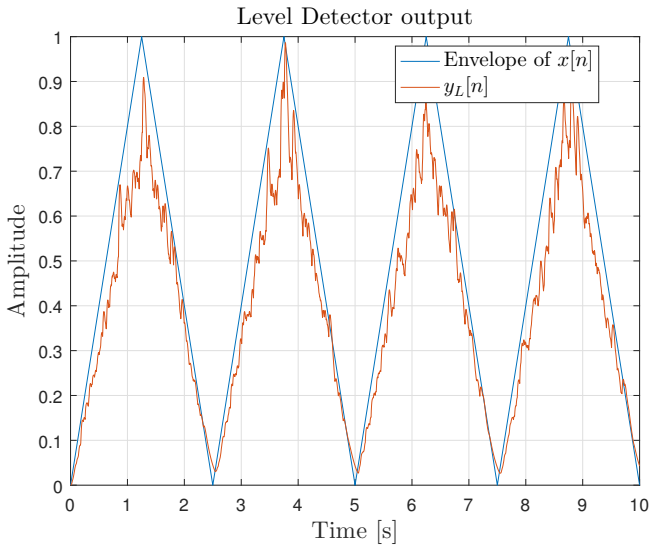


Figura 5: Envoltória do sinal de entrada e saída do *Level Detector*.

4 Implementação em C++

Com a estrutura do efeito validada em MATLAB, foi realizada a sua implementação em C++. O código desenvolvido deve ser compilado para um sistema embarcado de poucos recursos computacionais, logo a implementação foi dividida em duas etapas, caracterizadas pela representação numérica utilizada nos cálculos: ponto flutuante (*floating-point*, em inglês) e ponto fixo (*fixed-point*, em inglês). Essa abordagem é necessária, visto que o microcontrolador disponível no *Arduino Due* (*SAM3X8E*, baseado em *ARM Cortex-M3* de 32 bits) não possui uma unidade ponto flutuante (*FPU*, em inglês) dedicada, aumentando em muito o tempo de processamento de cálculos ao utilizar variáveis do tipo *float*.

Num primeiro momento, o efeito foi implementado através de uma classe em C++ com os cálculos realizados em ponto-flutuante de 32 bits. Foram realizados os mesmos cálculos que a implementação em MATLAB e, como a precisão numérica foi semelhante (para esta aplicação), foram obtidos os mesmos resultados do MATLAB e análises mais aprofundadas não foram realizadas.

A partir da implementação em C++ ponto flutuante foi desenvolvida uma versão do efeito utilizando a representação numérica de ponto fixo *Q16.16*, ou seja, cada variável é de 32bits, com 16 bits fracionários, 15 bits inteiros e 1 bit de sinal. Para facilitar a portabilidade do código, parte da biblioteca *MFixedPoint* [2] foi modificada e utilizada para realizar a representação numérica e as operações numéricas de ponto fixo *Q16.16*.

Ambas implementações (ponto flutuante e fixo) foram testadas utilizando o *Visual Studio 2017* e foram comparadas com a implementação em *MATLAB*, obtendo resultados similares, com diferenças não perceptíveis.

5 Implementação no Arduino Due

Na implementação do efeito no *Arduino Due* foi configurado um dos *timers* do processador para fixar a taxa de amostragem em $44,1$ kHz, a cada interrupção do *timer* é feita a leitura dos DACs e, após a execução do efeito, a escrita nos ADCs.

Para realizar o controle dos parâmetros do efeito, foram utilizados os potenciômetros da *PedalSHIELD*. Os valores dos potenciômetros são atualizados a cada 10 ms, ou 441 amostras. Na Figura 2 são mostrados os intervalos utilizados para os parâmetros lidos dos potenciômetros em representação numérica de ponto flutuante (para facilitar o entendimento). É interessante notar que os intervalos de α_A e α_R correspondem a constante de tempo de *attack* $\tau_A \in [5.3730, 92.8685]$ ms e a constante de tempo de *release* $\tau_R \in [1.0878, 743.0272]$ ms, respectivamente.

Ao ligar a chave para a seleção dos parâmetros pode ocorrer um problema de *"bouncing"*, em que a posição da chave varia eletricamente de forma indesejável. Esse problema foi contornado satisfatoriamente ao realizar a troca de estado da chave somente a cada 100 ms, ou 10 atualizações de parâmetros. Entretanto, em alguns casos esse problema ainda ocorre, o que faz com que os valores de *Range* e *Quality Factor* se alterem ocasionalmente caso a chave esteja ligada, o contrário não ocorre (com a chave desligada), logo sugere-se que a chave seja mantida desligada após regular os valores de *Attack* e *Release*.

[2] Disponível em https://github.com/mbedded-ninja/MFixedPoint

Parâmetro	Variável	Intervalo
Attack	α_A	$[0.9958, 0.9998]$
Release	α_R	$[0.9794, 1)$
Range	B_f	$[0.01, 0.7]$
Quality Factor	q	$[0.02, 0.707]$
Mix	α_{mix}	$[0, 1)$

Tabela 2: Variáveis afetadas pelos parâmetros da *PedalSHIELD* e o seu intervalo de valores.

6 Resultados e Considerações Finais

Com o efeito concluído, foram realizados testes com sinais de áudio reais para avaliar o seu desempenho. Foram utilizados trechos de amostra de um *Hohner Clavinet D6* que faz parte da *Nord Piano Library* [3], uma biblioteca de pianos digitais para os teclados da *Nord*. Junto com os trechos *"clean"*, há também os mesmos sons sob o efeito de um efeito *Auto-Wah* presente nos teclados da *Nord*, desta forma, é possível comparar o efeito desenvolvido com um efeito de uso profissional.

Na Figura 6, o sinal do canal esquerdo é a saída do efeito *Auto-Wah* em um teclado *Nord*, enquanto que no canal direito está a saída na *PedalSHIELD*.

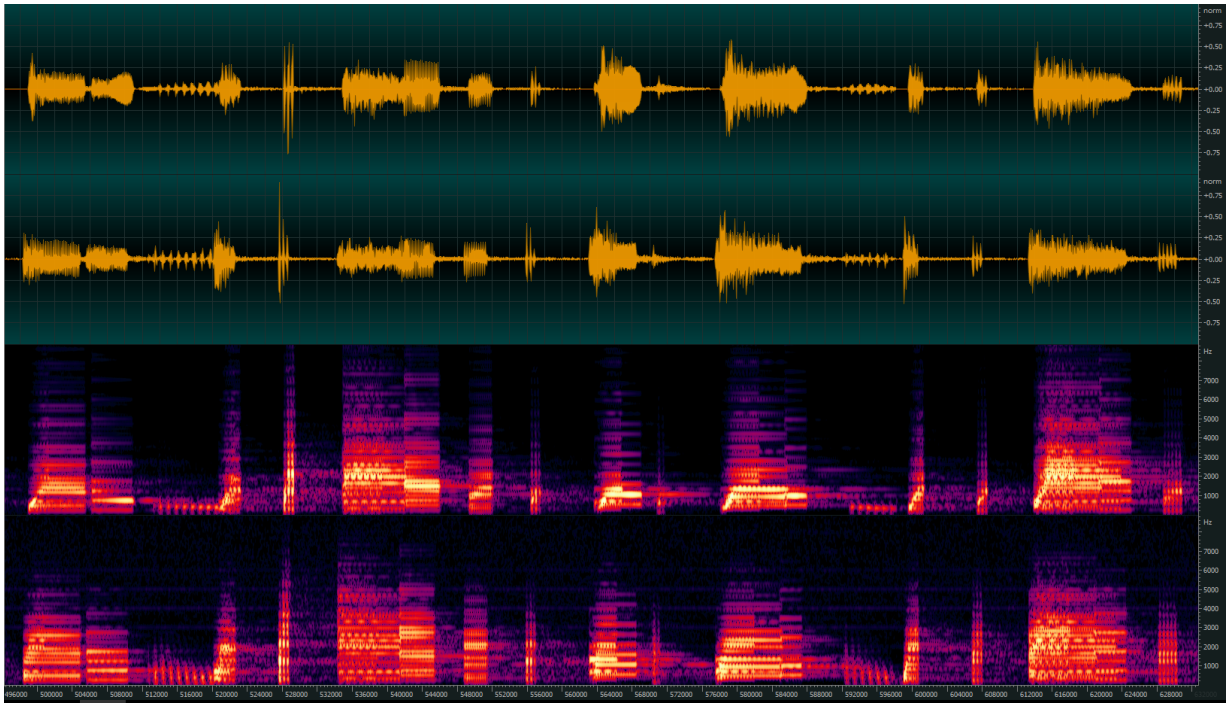


Figura 6: Saída do efeito *Auto-Wah* em um teclado *Nord* (canal esquerdo) e na *PedalSHIELD* (canal direito).

Contudo, pôde-se notar empiricamente que o efeito desenvolvido obteve resultados comparáveis a um efeito de uso profissional.

Referências

[1] D. Giannoulis, M. Massberg, and J. D. Reiss, "Digital dynamic range compressor design—a tutorial and analysis," *Journal of the Audio Engineering Society*, vol. 60, no. 6, pp. 399–408, 2012.

[2] U. Zölzer, *DAFX: Digital Audio Effects*. Wiley, 2011.

[3] Disponível em http://www.nordkeyboards.com/sound-libraries/nord-piano-library/hohner-clavinet-d6.