UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GUILHERME NUNES SERAFINA

Afinador automático digital para guitarras baseado no método do espectro do produto harmônico

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Tiago Balen

Porto Alegre
2015

“A utopia está lá no horizonte. Me aproximo dois passos, ela se afasta dois passos. Caminho dez passos, e o horizonte corre dez passos. Por mais que eu caminhe, jamais alcançarei. Para que serve a utopia? Serve para isso: para que eu não deixe de caminhar.”

(EDUARDO GALEANO)

AGRADECIMENTOS

Gostaria de agradecer à minha família, em especial minha mãe e minha vó, que sempre estiveram ao meu lado e apoiaram-me em todas as minhas empreitadas na vida.

Também gostaria de agradecer aos meus amigos, companheiros de estudo e de tantas diversões. Pessoas com as quais a existência é certamente mais agradável.

Agradeço aos meus professores que transmitiram seu conhecimento durante tantos anos, ajudando a moldar o engenheiro que hoje sou. Um agradecimento especial ao meu orientador neste trabalho.

Quero agradecer a esta Universidade e também à *Swansea University*, na qual cursei um ano da minha graduação. Agradeço, ainda, à União por investir em educação pública, gratuita e de qualidade para seus cidadãos.

Esta jornada está terminando e eu agradeço, finalmente, a todos que tornaram isso possível. Novas jornadas virão e, certamente, haverá novos e antigos companheiros para ajudar-me a vencê-las, afinal, *hasta la victoria siempre*!

RESUMO

Este trabalho consiste na pesquisa, desenvolvimento e teste de um afinador digital para guitarras empregando o método do espectro do produto harmônico. Tais dispositivos tem sido desenvolvidos já a algum tempo, porém, sem sombra de dúvidas, é um campo onde ainda é possível realizar muito avanços. O estudo aqui apresentado contém uma revisão sobre afinadores automáticos e técnicas para detecção da altura de sinais sonoros, o desenvolvimento e teste de um algoritmo empregando a ferramenta computacional MATLAB, e a implementação em hardware utilizando um microcontrolador PSoC.

Palavras-chave: Afinadores. Processamento digital de sinais. HPS. Detecção de altura de sinais sonoros.

Digital automatic tuner for guitars
based on the harmonic product spectrum technique

ABSTRACT

This work consists of research, development and testing of a digital tuner for guitars using the harmonic product spectrum method. For quite a while, such devices have been developed, although it is a field where it is still possible to achieve plenty of progress. The study presented here contains an overview of automatic tuners and techniques for detecting the pitch of a sound, the test and development of an algorithm using the MATLAB software tool, and the hardware implementation using a PSoC microcontroller.

Key-words: Tuners. Digital signal processing. HPS. Sound pitch detection.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ADC	Conversor analógico/digital
AMDF	Função de diferença média de magnitude
CPU	Unidade de processamento central
CMSIS	Interface Padrão de Software para Microcontroladores Cortex
DFB	Bloco de filtro digital
DFT	Transformada discreta de Fourier
DMA	Aceso direto à memória
DSP	Processamento digital de sinais
fACF	Função autocorrelação no domínio da frequência
FFT	Transformada rápida de Fourier
HPS	Espectro do produto harmônico
PSoC	Sistema-em-um-chip programável
SRAM	Memória estática de acesso aleatório
tACF	Função autocorrelação no domínio do tempo

SUMÁRIO

1 INTRODUÇÃO

Guitarras produzem som quando suas cordas são, em geral, percutidas ou puxadas, fazendo-as vibrar. Essas vibrações geram ondas sonoras, amplificadas pelo próprio corpo do instrumento, que age como uma caixa de ressonância, ou podem ser captadas diretamente por dispositivos que transformam tais vibrações em ondas elétricas que são amplificadas e convertidas em ondas sonoras através de alto-falantes.

A frequência sonora que cada corda produz quando tocada solta (isto é, quando o músico não a está pressionando contra a escala do instrumento) é definida por três fatores: o comprimento da corda, as características inerentes à corda (material, espessura) e a tensão aplicada à corda. Enquanto, em uma guitarra já configurada, os primeiros fatores são fixos, o terceiro, tensão, pode ser variado. Isso é importante para que se possa corrigir a frequência que cada corda produz (afinar o instrumento), já que ela se altera devido a pequenas variações no comprimento causadas por mudanças de temperatura e umidade, ou quando cordas novas são instaladas.

O processo de afinar uma guitarra tende a ser tedioso e, dependendo da habilidade de quem está afinando e do tipo da guitarra, bastante demorado. O método mais simples é o músico usar a sua audição para identificar se cada corda está soando como deveria. Enquanto esse método não exige nenhum equipamento extra, ele pode não ser tão preciso quanto se gostaria. Dispositivos foram então desenvolvidos para medir a frequência gerada pelas cordas e informar ao músico se elas estão produzindo as notas corretamente ou não com muito mais precisão. Isso certamente tornou a atividade de afinar uma guitarra muito mais fácil, mas o músico ainda precisa operar manualmente as cravelhas do instrumento para deixá-lo afinado. A partir daí, novos dispositivos começaram a ser criados que permitiam afinar o instrumento automaticamente, sem a necessidade de o músico alterar a tensão das cordas de forma manual. É nessa última categoria de dispositivos que este trabalho se inclui.

1.1 Características comuns às guitarras elétricas

Em geral, todas as guitarras possuem estrutura semelhante, e basta ao músico girar as cravelhas para corrigir a afinação das cordas uma a uma de forma independente. Isso é verdadeiro para instrumentos nos quais a ponte é fixa. Porém, existem guitarras que usam um

sistema de ponte flutuante com alavanca, como as consagradas pontes Floyd Rose, por exemplo. Esse sistema funciona da seguinte forma: a ponte é montada podendo mover-se livremente para frente ou para atrás a partir de um ponto que está em contato com o corpo da guitarra. Um conjunto de molas na parte traseira do instrumento puxa a ponte para trás, enquanto o encordoamento puxa a ponte para frente. A posição da ponte é definida pelo equilíbrio entre as tensões exercidas pelas molas por um lado e pelas cordas por outro. Uma alavanca permite ao músico variar a posição manualmente para modificar a frequência das notas produzidas (FENDER, 1988; ROSE, 1979).

Uma desvantagem desse sistema está justamente no processo necessário para afinar o instrumento. A ponte está flutuando entre duas tensões, uma delas sendo a soma das tensões exercidas pelas cordas. Dessa forma, a alteração da tensão de uma corda afeta a posição da ponte como um todo, e por consequência, a afinação das outras cordas. Isso faz com que seja necessário várias iterações para se chegar a uma afinação correta por completo, um processo que pode ser bastante demorado.

Com relação ao sensor utilizado para converter as vibrações das cordas em ondas elétricas, a imensa maioria das guitarras utilizam captadores baseados na variação de fluxo magnético. Abaixo de cada corda existe uma bobina (às vezes mais de uma) com um ímã no núcleo. Esse ímã polariza a corda, que ao vibrar muda sua posição relativa à bobina, gerando uma variação do fluxo do campo magnético no enrolamento. Dessa forma, a corrente induzida na bobina tem a mesma frequência da oscilação da corda (HALLIDAY, RESNICK e WALKER, 2009). A guitarra então mistura o sinal de proveniente de cada bobina em um só sinal antes de enviá-lo à saída. Isso dificulta imensamente o processo de identificar a frequência de cada corda se todas forem tocadas ao mesmo tempo, e mesmo se o sinal de cada bobina seja tratado separadamente, existe um acoplamento, o que significa que a vibração de uma corda acaba sendo captada por outras bobinas que não a(s) que está(ão) imediatamente abaixo dela.

1.2 Objetivos

O objetivo deste trabalho é criar um método para conseguir afinar automaticamente uma guitarra elétrica. Para isso, foi utilizado um microcontrolador da família PSoC. O método foi desenvolvido e testado utilizando uma guitarra com ponte fixa, porém foi pensado de forma a poder ser estendido a guitarras com ponte flutuante.

Outro ponto a ser destacado é que o sistema foi concebido de modo a ser utilizado com sensores que não sofram do problema de acoplamento entre bobinas magnéticas citado acima. Um exemplo de tais sensores são os captadores ópticos. Com eles, é possível isolar completamente uma corda das outras, eliminando as interferências. Além disso, captadores magnéticos podem ser usados apenas com cordas metálicas, enquanto que com um captador óptico será possível usar o dispositivo com instrumentos que usem outros tipos de cordas, como nylon, por exemplo. Captadores ópticos, apesar de serem muito menos comuns que os magnéticos, são uma realidade, e existem diversos modelos disponíveis.

1.3 Organização deste trabalho

Este trabalho está, daqui para a frente, divido em cinco seções: na primeira, é apresentado um estudo sobre o estado da arte em afinadores automáticos; na segunda, um estudo sobre os métodos para obtenção da frequência fundamental de sinais de áudio; o desenvolvimento e teste do algoritmo baseado no método escolhido utilizando a ferramenta computacional MATLAB é apresentado na terceira; na quarta, o desenvolvimento e teste do sistema utilizando o microcontrolador PSoC e uma guitarra elétrica; por fim, na última, as conclusões e trabalhos futuros.

Ao final é apresentado um glossário com termos específicos do mundo das guitarras elétricas.

2 ESTUDO DO ESTADO DA ARTE EM AFINADORES AUTOMÁTICOS

Dispositivos projetados com o intuito de afinar automaticamente um instrumento são quase tão antigos quanto os criados para medir a frequência gerada pelas cordas. De fato, o primeiro passo para conseguir atuar na tensão das cordas de forma a corrigir a afinação é obter, com precisão, qual a nota produzida por elas. Desde os anos 1970 diversas patentes foram registradas descrevendo variados métodos e equipamentos com esse objetivo de afinar guitarras e instrumentos similares.

Apesar de o mais comum ser trabalhar com a frequência de vibração das cordas, alguns aparelhos registrados preferem medir a tensão sob a qual a corda é submetida. É o caso dos dispositivos criados por Scholz (1984) e Cumberland (2001). No entanto, esses dispositivos exigem significativas mudanças na estrutura do instrumento. Além disso, como a frequência gerada pela corda não depende apenas da tensão, mas também da densidade linear da corda (NAVE, 2012), isso poderia levar a uma limitação do uso destes dispositivos com determinados tipos de encordoamentos.

Os equipamentos projetados para trabalhar com a frequência das cordas usam, em geral, captadores magnéticos. No entanto é possível encontrar registros de equipamento que podem ser usados com microfones, como os desenvolvidos por Hedrick (1978), Skinn e Freeland (1990) e Milano, Rastegar e Khorrami (1998). Enquanto a captação via microfone pode ser ideal para instrumentos como pianos, para o qual não existe captadores magnéticos, para uma guitarra praticamente inviabilizaria a possiblidade de afinar todas as cordas ao mesmo tempo, já que o acoplamento entre os microfones de cada corda seria muito grande (assumindo que cada corda tenha seu próprio microfone).

Nem todos as invenções registradas se propõem a afinar todas as cordas ao mesmo tempo. O equipamento criado por Miller e Strock (1994), por exemplo, exige que cada corda seja tocada individualmente, já que ele usa a saída de áudio comum da guitarra, que mistura o sinal de todas as cordas. Alguns dispositivos preveem uma forma de excitar as cordas, podendo afinar o instrumento sem que o músico precise tocá-las. É o caso do dispositivo criado por Oudshoorn, Moler e Akhavein (2002).

Todos os dispositivos pesquisados empregam componentes eletrônicos em sua operação, mas é possível dividi-los em dois grupos: os analógicos e os digitais. Com o avanço dos microprocessadores e das técnicas de processamentos digital de sinais, os desenvolvimentos mais recentes na área de afinadores automáticos todas empregam métodos

digitais em sua concepção. É o caso, por exemplo, dos aparelhos registrados por Wynn (1999), Long *et al.* (2001), Whittall, Dent e Lambert (2002) e Adams (2008). Normalmente um corversor A/D é usado para digitalizar o sinal, que é processado por um microcontrolador ou computador. Essa metodologia oferece uma flexibilidade maior, podendo oferecer mais opções de afinação para o músico, ou até mesmo afinações personalizadas. Afinadores analógicos, como o projetado por Minnick (1986), utilizam amplificadores operacionais na configuração comparador para medir a diferença entre o sinal produzido pelo captador e um sinal de referência.

Além de patentes, no meio acadêmico também há registro de trabalhos com afinadores automáticos, como o sistema criado por Chai *et al.* (2014), estudantes da *University of Alberta*. Esse trabalho possui semelhanças estruturais, inclusive, com o sistema aqui proposto.

Todos os aparelhos pesquisados utilizam o resultado do processamento para acionar o funcionamento de algum tipo de motor que altera a tensão e, por consequência, a frequência de vibração das cordas.

3 REVISÃO SOBRE MÉTODOS PARA DETECÇÃO DA ALTURA DE SINAIS SONOROS

A altura é um dos quatro parâmetros que definem um tom musical, sendo os outros duração, intensidade e timbre. Como ela é o objeto principal deste capítulo, convém, antes de mais nada, defini-la. O Novo Dicionário Aurélio da Língua Portuguesa usa a seguinte definição: "altura. S. f. [...] 12. Fís. Propriedade de uma onda ou vibração sonora, caracterizada pela frequência da vibração." (1986, p. 94). A altura, desse modo, permite classificar os sons entre mais altos ou mais baixos (ou mais agudos e mais graves).

Apesar de estar relacionada à frequência, a altura é, na verdade, um termo subjetivo, sendo objeto de estudo da psicoacústica, e que diz sobre a percepção do som pelo ouvido humano. Pode-se dizer que a altura é uma função da frequência, mas cuja relação não é linear. A pressão sonora é um fator que influencia nessa percepção. Dessa forma, a altura tem sua própria unidade subjetiva de medida, o mel, enquanto a frequência, uma propriedade física, é medida em Hertz (EVEREST e POHLMANN, 2015).

Felizmente, a frequência fundamental de um sinal sonoro periódico e a altura percebida normalmente coincidem. Uma situação em que isso não ocorre é chamada de *fundamental ausente*. Ao escutar três tons com diferenças de frequência iguais entre si, por exemplo 100, 150 e 200 Hz, uma pessoa percebe a altura em 50 Hz pois seu sistema auditivo interpreta os tons como sendo harmônicas de uma fundamental que na verdade não existe (WHITAKER e BENSON, 2001).

Essa discussão sobre a altura, e sua relação com os outros parâmetros sonoros, ainda é objeto de estudo no meio acadêmico e foge ao escopo deste trabalho, portanto, daqui para a frente, será usada uma definição simplificada: altura é a frequência fundamental de um sinal sonoro.

Existem diversos algoritmos para detecção da altura de tais sinais. Tanto é que essa questão chega a ser considerada resolvida por muitos estudiosos, principalmente no caso de gravações (DE LA CUADRA, MASTER e SAPP, 2001). Cada algoritmo tem suas próprias características. Alguns são mais robustos, outros mais eficientes computacionalmente. Alguns operam melhor com sinais que possuem fundamental e harmônicas, outras apenas a fundamental. Podemos dividi-los em três categorias: os que trabalham no domínio do tempo; os que trabalham no domínio da frequência; e os que trabalham nos dois (RABINER, CHENG, *et al.*, 1976).

Nesta revisão serão abordadas as duas primeiras categorias, já que a última é demasiada complexa para o objetivo que este trabalho visa alcançar. Existem ainda os algoritmos que trabalham com modelagens do ouvido humano, mas estes, pelo mesmo motivo, também não serão abordados aqui. Esta lista de técnicas de detecção de altura serviu como base para a escolha de um algoritmo adequado aos propósitos deste trabalho, o que será discutido no próximo capítulo e não se propõem que seja exaustiva no que diz respeito à variedade de métodos que existem para esse fim.

3.1 Detecção no domínio do tempo

Esta categoria inclui algoritmos cuja ideia principal é bastante fácil de entender. Eles operam diretamente sobre a onda sonora, tentando achar um padrão nela que possibilite a identificação de um componente harmônico.

3.1.1 Contador de passagens por zero

Esta, sem dúvida, é a técnica mais simples de todas, mas que também apresenta os piores resultados. Consiste em contar quantas vezes a onda cruza o eixo das abscissas em um determinado período, ou seja, quantas vezes a onda passa pelo zero. Dividindo-se o valor dessa contagem pelo período obtêm-se a frequência. É uma técnica pouco onerosa, porém apresenta resultados pífios se o sinal contiver ruído e também se ele contiver sobretons com mais energia que a fundamental. Uma variação deste algoritmo consiste em contar os picos e os vales do sinal, também com resultados ruins.

3.1.2 Autocorrelação no tempo

A função autocorrelação no domínio do tempo (tACF, na sigla em inglês) é um dos métodos mais usados na detecção de altura, por ser um dos mais robustos e confiáveis (RABINER, 1977). É especialmente popular na área do processamento de voz. Este algoritmo parte da ideia de que um sinal periódico terá uma grande correlação com si mesmo quando atrasado de seu período fundamental. O método consiste, dessa forma, em achar os picos da função autocorrelação (DIAS, VENTURA e GASPAR, 2008):

$$r_{xx}[n] = \frac{1}{N}\sum_{k=0}^{N-n-1} x[k]x[k+n]. \quad (1)$$

O maior pico da função é sempre em zero, ou seja, sem atraso ($n = 0$). O local do pico seguinte dá uma estimativa do período, e sua altura indica o nível de periodicidade do sinal.

Uma alternativa ao tACF é a técnica da função de diferença média de magnitude (AMDF, na sigla em inglês). Esse algoritmo não usa o produto do sinal com ele mesmo, mas sim a diferença e, dessa forma, procura por vales, e não picos. Existe também um ganho computacional, já que subtrações são menos demoradas do que multiplicações. A equação usada é a seguinte:

$$r_{xx}[n] = \frac{1}{N}\sum_{k=0}^{N-1} |x[k] - x[k+n]|. \quad (2)$$

Existe ainda a possibilidade de usar ambos para ganhar em robustez (DE LA CUADRA, MASTER e SAPP, 2001).

O cálculo da autocorrelação pode tornar-se excessivamente grande, dependendo do tamanho da janela, mas felizmente existe a possibilidade do uso de algoritmos do tipo FFT para acelerar a computação (OPPENHEIM, SCHAFER e BUCK, 1998, p. 746).

3.1.3 Máxima verossimilhança no domínio do tempo

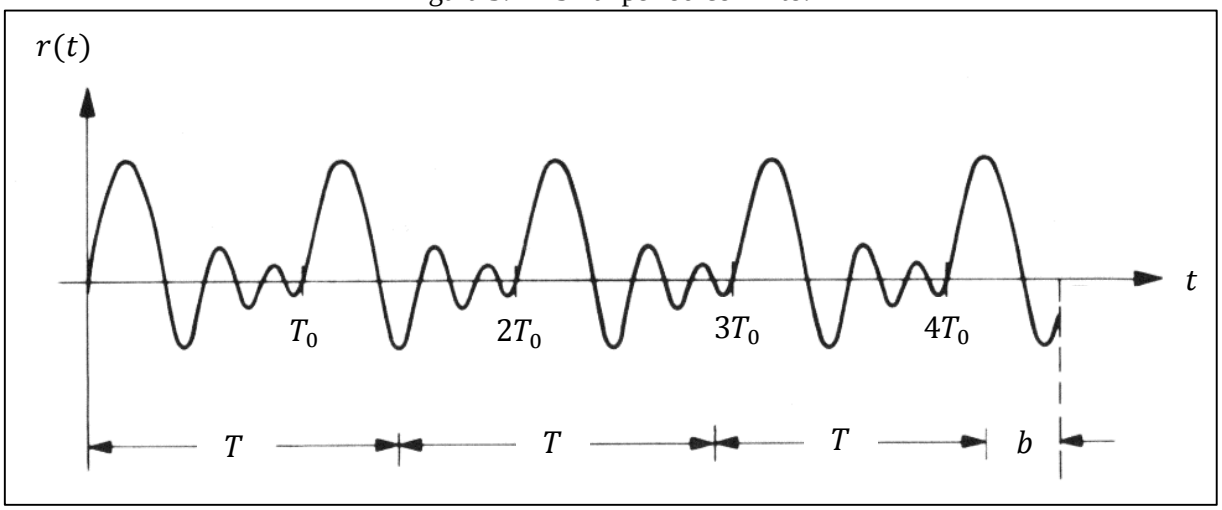
Este seria o método ótimo para a detecção da altura no domínio do tempo (NOLL, 1970). Consiste em estimar por máxima verossimilhança o período do sinal de áudio (ou qualquer outro sinal periódico). Este sinal, que tem período fundamental T_0 é dividido em vários trechos de tamanho T. Estes trechos são somados, e quando $T = T_0$ então estima-se que a verossimilhança seja a máxima possível. Isso fica mais claro ao olhar as seguintes equações e também a Figura 3.1:

$$r(t,T) = \begin{cases} \frac{1}{N+1}\sum_{n=0}^{N} r(t+nT) & ,0 \le t < b \\ \frac{1}{N}\sum_{n=0}^{N-1} r(t+nT) & ,b \le t < T \end{cases}; \quad (3)$$

$$J(T) = (N+1)\sum_{t=0}^{b} r^2(t,T) + N\sum_{t=b+1}^{T} r^2(t,T). \quad (4)$$

Fica claro que a função $J(T)$ é maximizada na forma $J(T = T_0)$.

Figura 3.1 – Sinal periódico finito.



Fonte: Adaptado de Noll (1970).

3.2 Detecção no domínio da frequência

A detecção no domínio da frequência segue, basicamente, a seguinte ordem: o sinal é dividido em quadros; esse quadros são então multiplicados por alguma janela, como Hamming, Hann *etc.*; em cima do resultado desta multiplicação é realizada uma transformada que leva o sinal para o domínio da frequência, sendo a transformada discreta de Fourier (DFT) a mais comum. Quando o sinal é periódico, aparecem picos no local da frequência fundamental e suas harmônicas. Os algoritmos desta categoria utilizam-se de técnicas para identificar e trabalhar com esse picos. Em geral, este algoritmos são mais robustos do que os que trabalham no domínio do tempo (DE LA CUADRA, MASTER e SAPP, 2001).

3.2.1 Autocorrelação e máxima verossimilhança no domínio da frequência

Ambos partem da mesma ideia de seus "irmãos" do domínio do tempo. A diferença é que agora eles trabalham com o espectro de frequência do sinal. No caso da autocorrelação (função autocorrelação no domínio da frequência, fACF, na sigla em inglês), o fato de que o espectro de magnitude de um sinal sonoro é periódico é usado de base para o cálculo. Assim sendo, "quaisquer dois componentes espectrais com intervalo de frequência m, multiplicados

pela frequência de amostragem e o inverso da ordem da FFT (mF_s/K) é um candidato à F_0" (DIAS, VENTURA e GASPAR, 2008, tradução nossa)[1]. A seguinte equação é usada:

$$r[m] = \frac{2}{K} \sum_{k=0}^{\frac{K}{2}-m-1} |X[k]||X[k+m]|, \tag{5}$$

onde K é a ordem da FFT. O segundo valor mais alto dessa função (o máximo dela será sempre em $m = 0$) indica o período fundamental do sinal.

Já a máxima verossimilhança busca, dentre um conjunto de espetros ideais, o que mais aproxima-se da forma do espectro do sinal de entrada. Tais espectros ideais são trens de impulsos com frequências variadas, que são convoluídos com o sinal de áudio. O algoritmo tenta então minimizar o erro entre esse sinal convoluído e o sinal ideal, conforme a equação abaixo:

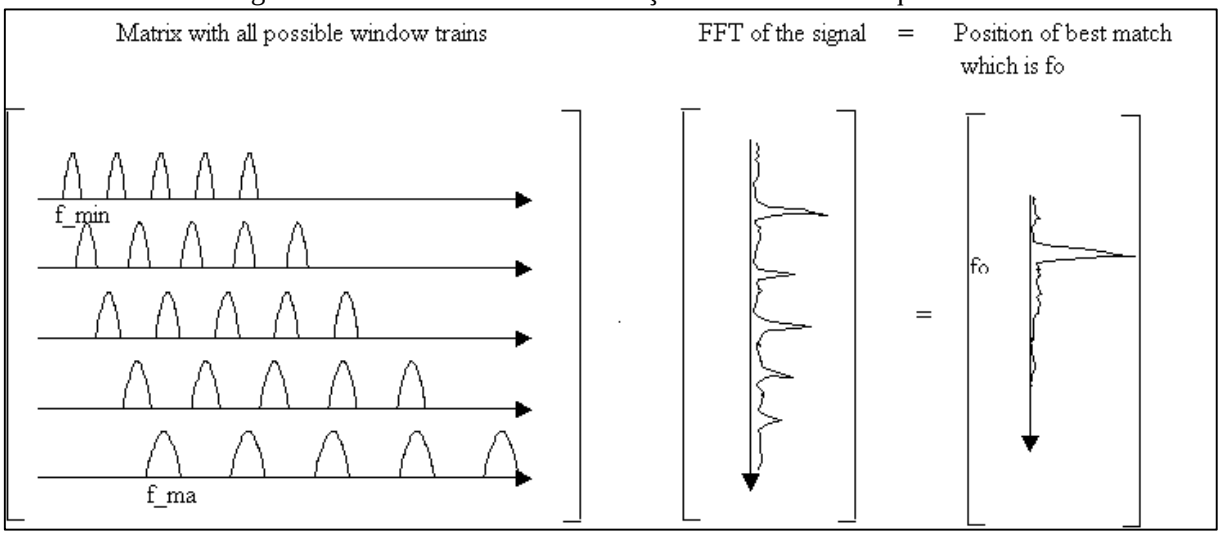
$$E(\omega) = \left\|Y - \tilde{Y}_\omega\right\|^2 = \|Y\|^2 + \left\|\tilde{Y}_\omega\right\|^2 - 2Y\tilde{Y}_\omega{}^T, \tag{6}$$

onde Y é o sinal de entrada convoluído com o trem de impulsos e $\tilde{Y}_\omega$ é o espectro ideal para uma determinada frequência ω. Como tanto $\|Y\|^2$ quanto $\left\|\tilde{Y}_\omega\right\|^2$ permanecem constantes, o valor mínimo da função erro $E(\omega)$ acontece quando o produto $Y\tilde{Y}_\omega{}^T$ é máximo (DE LA CUADRA, MASTER e SAPP, 2001):

$$Y_{f_0} = \min_\omega\{E(\omega)\} = \max_\omega\left\{Y\tilde{Y}_\omega{}^T\right\}. \tag{7}$$

A Figura 3.2, abaixo, mostra o funcionamento do algoritmo por meio de gráficos.

Figura 3.2 – Máxima verossimilhança no domínio da frequência.



Fonte: de la Cuadra, Master e Sapp (2001).

[1] "any two spectral components with a frequency interval m, multiplied by the sampling rate and the inverse of the FFT order (mF_s/K) is a F_0 candidate"

3.2.2 Cepstrum

O termo “cepstrum” é uma modificação da palavra *spectrum* (espectro, em inglês), gerado ao inverter a ordem das quatro primeiras letras desse vocábulo. Outros termos que envolvem o cepstrum também foram criados similarmente, como “quefrência” (frequência) (SUJATHA, 2010). O cepstrum de um sinal é calculado tomando-se a DFT do logaritmo do espectro de magnitude desse sinal, ou a transformada inversa do logaritmo desse espectro. Dessa forma, como a representação no domínio da frequência de um sinal de áudio é periódico, ao tomar-se novamente a DFT o resultado mostrará um pico que corresponde ao período fundamental do sinal.

A análise cepstral está entre os métodos mais populares para determinação da altura de sinais sonoros. No entanto é necessário tomar o cuidado de inspecionar-se um segundo e um terceiro pico com igual distância do que corresponde ao período fundamental, pois outros picos, que não estão relacionado ao T_0, também surgem no resultado do processamento (MASTER, 2000, p. 10).

3.2.3 Espectro do produto harmônico

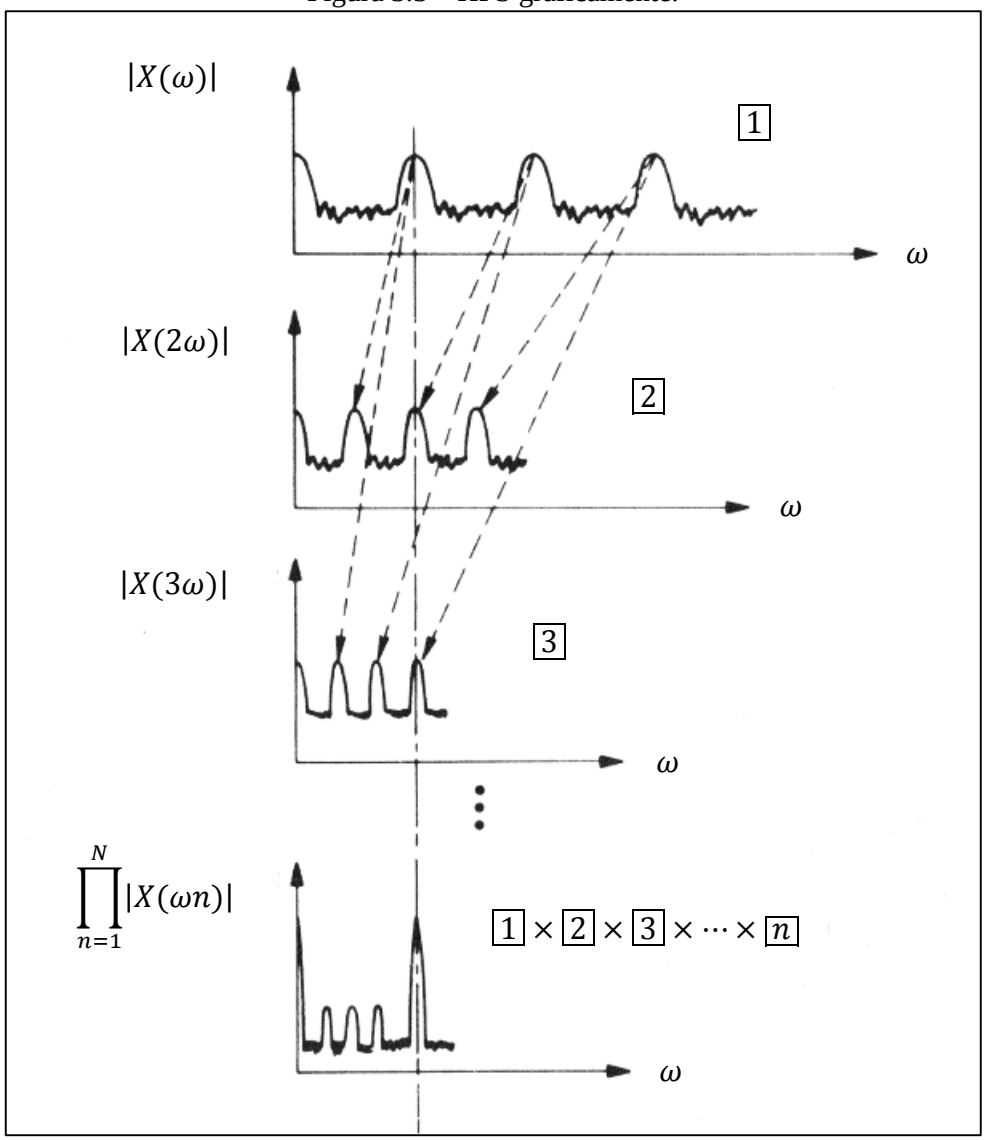
O espectro do produto harmônico (HPS, na sigla em inglês) parte do princípio de que o espectro de magnitude de um sinal sonoro que contenha múltiplos harmônicos da frequência fundamental contém uma série de picos, que estão localizados exatamente nos locais da f_0 e sua harmônicas. Desse modo, ao comprimir-se o espectro de frequência (subamostrá-lo) por um número inteiro e compará-lo ao original, os picos da frequência fundamental e de alguma de suas harmônicas se alinham. Se o espectro foi comprimido por uma fator de dois, então seu segundo pico alinha-se com o primeiro; se ele foi comprimido por um fator de três, então o terceiro pico é que se alinha. Ao multiplicar-se o espectro original pelas suas versões comprimidas, um grande pico surge exatamente no local da frequência fundamental (DIAS, VENTURA e GASPAR, 2008). As seguintes equações resumem matematicamente o processo:

$$Y(\omega) = \prod_{n=1}^{N} |X(\omega n)|\,; \tag{8}$$

$$\hat{Y} = \max_{\omega}\{Y(\omega)\}, \tag{9}$$

onde N é o número de compressões, ou seja, de harmônicas a ser considerado e $\hat{Y}$ corresponde ao local da frequência fundamental. A Figura 3.3 mostra com mais clareza a técnica empregada no algoritmo:

Figura 3.3 – HPS graficamente.



Fonte: Adaptado de Noll (1970).

Existe também uma variação deste algoritmo chamado de espectro da soma harmônica, cuja única diferença é que os vários espectros são somados, ao invés de multiplicados:

$$Y(\omega) = \sum_{n=1}^{N} |X(\omega n)|. \tag{10}$$

Dessa forma, o algoritmo fica mais "leve" computacionalmente, porém fica muito mais sensível a ruídos, e portanto acaba por ter uma eficácia bastante reduzida em relação à versão

com multiplicações. Existe também a possibilidade combinar as técnicas das últimas duas subseções para criar um algoritmo mais eficaz, chamado de Cepstrum-Biased HPS (MASTER, 2000 *apud* DE LA CUADRA, MASTER e SAPP, 2001).

4 DESENVOLVIMENTO E TESTE NO MATLAB

Neste capítulo serão descritas as etapas para a escolha, desenvolvimento e teste do algoritmo a ser usado para detectar a altura do sinal proveniente do captador de uma guitarra elétrica. Avaliou-se que seria importante usar uma ferramenta computacional como o MATLAB para desenvolver tal algoritmo, já que ele oferece muitas facilidades na programação e, dessa forma, o código poderia ser extensivamente testado para que não houvesse dúvidas e não se perdesse tempo programando em C no PSoC algo que tivesse que ser alterado depois.

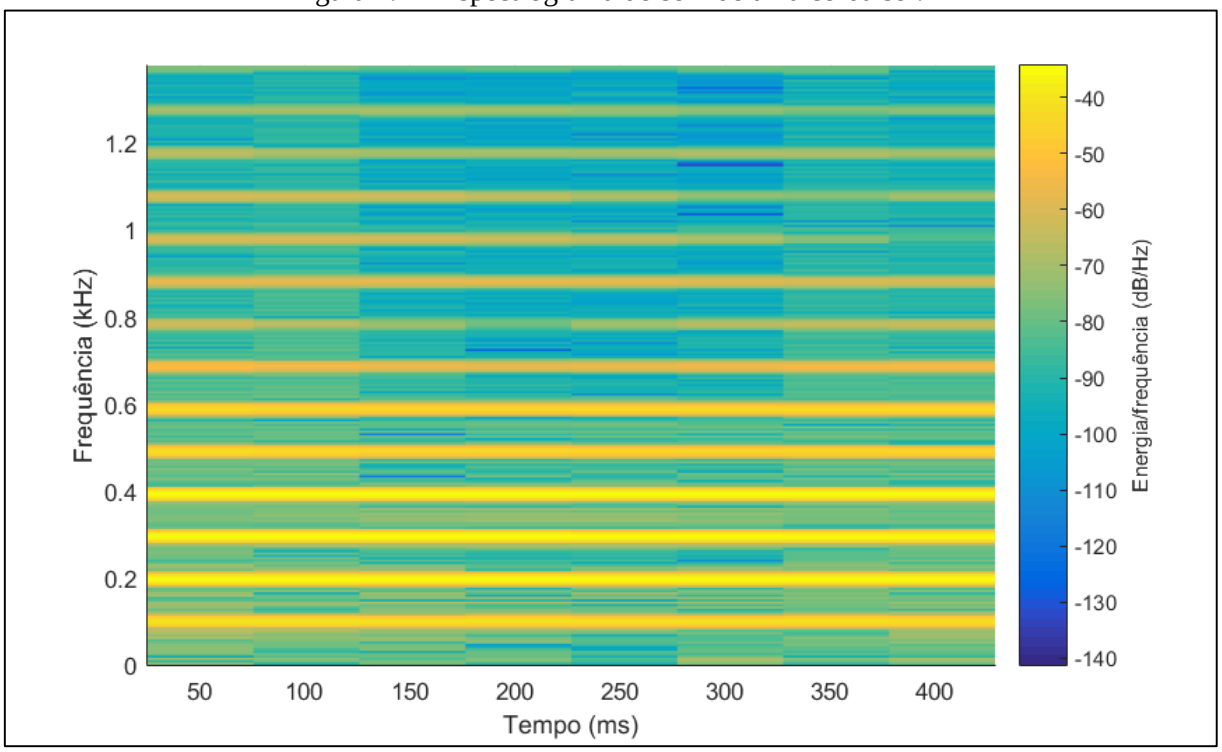
4.1 Características do sinal de áudio de uma guitarra elétrica

Para ponderar sobre o método para detecção de altura, é necessário avaliar as características do sinal de áudio que será tratado pelo algoritmo. Esse sinal será gerado pelo captador de uma guitarra elétrica, portanto é possível considerar que ele conterá um pouco de ruído, mas, em compensação, zero de som ambiente, o que poderia ocorrer se fosse empregado um microfone, por exemplo.

Outra questão relevante sobre o áudio diz respeito às características das ondas sonoras que serão captadas. Guitarras, assim como violinos, contrabaixos *etc.*, produzem som ao fazer suas cordas vibrarem. Ao vibrar, essa cordas produzem sobretons que são todos múltiplos da frequência fundamental, e que portanto podem ser chamados de harmônicos (EVEREST e POHLMANN, 2015). Isso é um fator importante para esta análise. Alguns instrumentos de sopro, por exemplo, tem sons cujos sobretons harmônicos possuem pouca energia, que está concentrada na frequência fundamental. Há ainda instrumentos cujos sobretons não são harmônicos, como os de percussão, ou são um misto de harmônicos e não-harmônicos, como o piano. A Figura 4.1 mostra o espectrograma de um trecho de áudio de um baixo elétrico com sua primeira corda soando ($sol_2 = 98\,Hz$). Fica claro que há bastante energia tanto na fundamental quanto nas suas harmônicas.

Quanto ao tamanho do espectro de frequências com o qual um afinador digital precisa trabalhar, é só lembrar que a frequência mais baixa é a da última corda da guitarra soando solta, $mi_2 = 82{,}41\,Hz$, enquanto a mais alta será algum múltiplo da frequência da primeira corda soando solta, $mi_4 = 329{,}63\,Hz$. Dessa forma, o espectro de frequência se estende por umas quatro oitavas acima da menor frequência.

Figura 4.1 – Espectrograma do som de uma corda sol.



Fonte: elaborada pelo autor.

4.2 Escolha de um método de detecção de altura adequado

Com as informações da seção acima é possível fazer a opção por uma técnica adequada, conforme o estudo feito no capítulo 3. O primeiro método a ser descartado é do contador de passagens por zero, pois não funciona com sinais complexos como os que um afinador para guitarras trabalha.

Optou-se também por descartar os outros métodos que trabalham no domínio do tempo. Como dito, os métodos que trabalham no domínio da frequência são, em geral, mais robustos. Robustez é um aspecto importante em um afinador automático, já que uma aferição incorreta da altura poderia levar a consequências desastrosas para a guitarra (o algoritmo poderia instruir o afinador a puxar demais determinada corda, partindo-a).

Dentre estes métodos que trabalham no domínio da frequência, o primeiro a ser descartado foi o baseado no cepstrum, pois emprega dupla transformação (DFT do logaritmo da DFT do sinal), o que poderia deixar o algoritmo excessivamente lento. O método da máxima verossimilhança também foi descartado, pois trabalha melhor com instrumentos que produzam

um número discreto de frequências, como teclados. As guitarras, de forma contrária, podem produzir qualquer frequência (DE LA CUADRA, MASTER e SAPP, 2001).

Dentre os métodos restantes, autocorrelação e espectro do produto harmônico, poder-se-ia escolher qualquer um, mas a preferência foi pelo HPS por ser o de mais fácil implementação. Cabe notar que o som produzido pela guitarra cumpre o pré-requisito básico do HPS: possui energia tanto na frequência fundamental como suas harmônicas.

4.3 Proposta de algoritmo

A primeira etapa necessária para o processamento digital do áudio é amostrá-lo, convertendo o sinal de entrada $x_i(t)$ na sequência $x_i[n]$:

$$x_i[n] = x_i\left(\frac{n}{f_s}\right), \tag{11}$$

onde n é qualquer numero inteiro e f_s é a frequência de amostragem. É necessário então dividir esta sequência em janelas de largura L e multiplicá-las por uma função janela $w[n]$, a fim de minimizar as ondulações que serão posteriormente causadas pela DFT:

$$x_w[n] = x_i[n]w[n].$$

O próximo passo é obter a tranformada discreta de Fourier do sinal e depois seu espectro de magnitude:

$$X[k] = \sum_{n=0}^{N-1} x_w[n]e^{-j\frac{2\pi}{N}}; \tag{12}$$

$$X_{mag}[k] = |X[k]|, \tag{13}$$

onde N é a ordem da DFT. A próxima etapa é a mais marcante do processo, que é gerar o espectro do produto harmônico:

$$Y[k] = \prod_{h=1}^{H} X_{mag}[hk], \tag{14}$$

onde H é a quantidade de vezes que o sinal é subamostrado. Agora é necessário achar o máximo dessa função, que é o local em que se encontra a candidata a frequência fundamental:

$$M = \max_k\{Y[k]\}. \tag{15}$$

Para evitar que um resultado possívelmente incorreto seja tido como certo, e também não gerar resultado quando o sinal de áudio contiver ruídos ou silêncio apenas, é necessário

lançar mão do teorema de Parseval para calcular tanto a energia total do sinal quanto a energia nos arredores das D primeiras possíveis harmônicas dele, com base na candidata a f_0:

$$E_t = \frac{1}{N}\sum_{k=0}^{N-1}\left(X_{mag}[k]\right)^2; \tag{16}$$

$$E_h = \frac{1}{N}\sum_{d=1}^{D}\left(\sum_{k=(M\times d)-a}^{(M\times d)+a}\left(X_{mag}[k]\right)^2\right), \tag{17}$$

onde a é o parâmetro que define o que é considerado "arredor" das harmônicas. Com esses resultados é possível definir um critério para aceitação do resultado: se a razão entre a energia nos entornos das harmônicas e a energia total for maior ou igual do que r, então aceita, senão rejeita:

$$\begin{aligned} \frac{E_h}{E_t} \geq r \quad &\Longrightarrow \quad f_0 = M \times \frac{f_s}{N}; \\ \frac{E_h}{E_t} < r \quad &\Longrightarrow \quad f_0 \; não \; existe. \end{aligned} \tag{18}$$

De forma mais simplificada, algoritmo proposto consiste das seguinte etapas (por enquanto, considera-se que o sinal de entrada já foi devidamente filtrado de modo a evitar *aliasing*):

1. Digitalização do sinal de áudio com frequência de amostragem f_s;
2. Divisão do sinal em janelas de largura L;
3. Multiplicação do sinal por uma função janela (Hamming, Kaiser *etc.*);
4. Transformação do sinal para o domínio da frequência por meio de uma transformada rápida de Fourier de tamanho N;
5. Obtenção do módulo desse sinal transformado;
6. Criação de $H - 1$ cópias subamostradas do sinal, com taxas de compressão 2 até H;
7. Multiplicação do resultado da etapa 4 por suas cópias comprimidas;
8. Descoberta do ponto de máximo M do resultado da etapa 6, que é o local da candidata a f_0;
9. Através do teorema de Parseval, cálculo da energia total do sinal e também da energia nos entornos das D primeiras possíveis harmônicas dele, com base na candidata a f_0;

10. Se a razão entre a energia nesses arredores e a energia total for maior ou igual do que r, então aceita o resultado como frequência fundamental, caso contrário rejeita.

4.4 Teste e ajuste do algoritmo

Existe uma série de parâmetros no algoritmos que precisam ser definidos: a frequência de amostragem f_s, a largura de janela L, a função janela $w[n]$, a ordem N da DFT, a quantidade H de vezes que o sinal será subamostrado, a quantidade D de harmônicas a ser levado em consideração, o parâmetro a que define o que é considerado entorno de uma frequência e, por fim, o valor r que define a proporção da energia que deve estar no entorno das harmônicas.

O primeiro parâmetro que é necessário definir é D. Com ele é possível saber qual a frequência de trabalho máxima para o algoritmo. Considerando D igual a quatro, o que, por agora, parece um valor razoável, temos que a frequência máxima é:

$$f_{max} = D \times mi_4 = 4 \times 329{,}63\ Hz = 1318{,}52\ Hz. \quad (19)$$

Com este dado, é possível estabelecer um valor mínimo para a frequência de amostragem f_s que, de acordo com o teorema de Nyquist–Shannon, tem que ser igual ou maior do que a frequência máxima, para evitar a ocorrência de *aliasing* nas frequências de interesse. Portanto:

$$f_s \geq (2 \times f_{max} = 2637{,}04\ Hz). \quad (20)$$

O próximo parâmetro a ser definido é o tamanho N da DFT. Cabe, antes disso, um reflexão: qual a resolução desejada para o resultado do processamento? Decidiu-se que $2\pi\ rad/s$, ou 1 Hz, já seria satisfatório. Desse modo, conclui-se que a ordem da DFT tem que ser igual à frequência de amostragem, já que o espaçamento entre frequências da DFT é $2\pi f_s/N$:

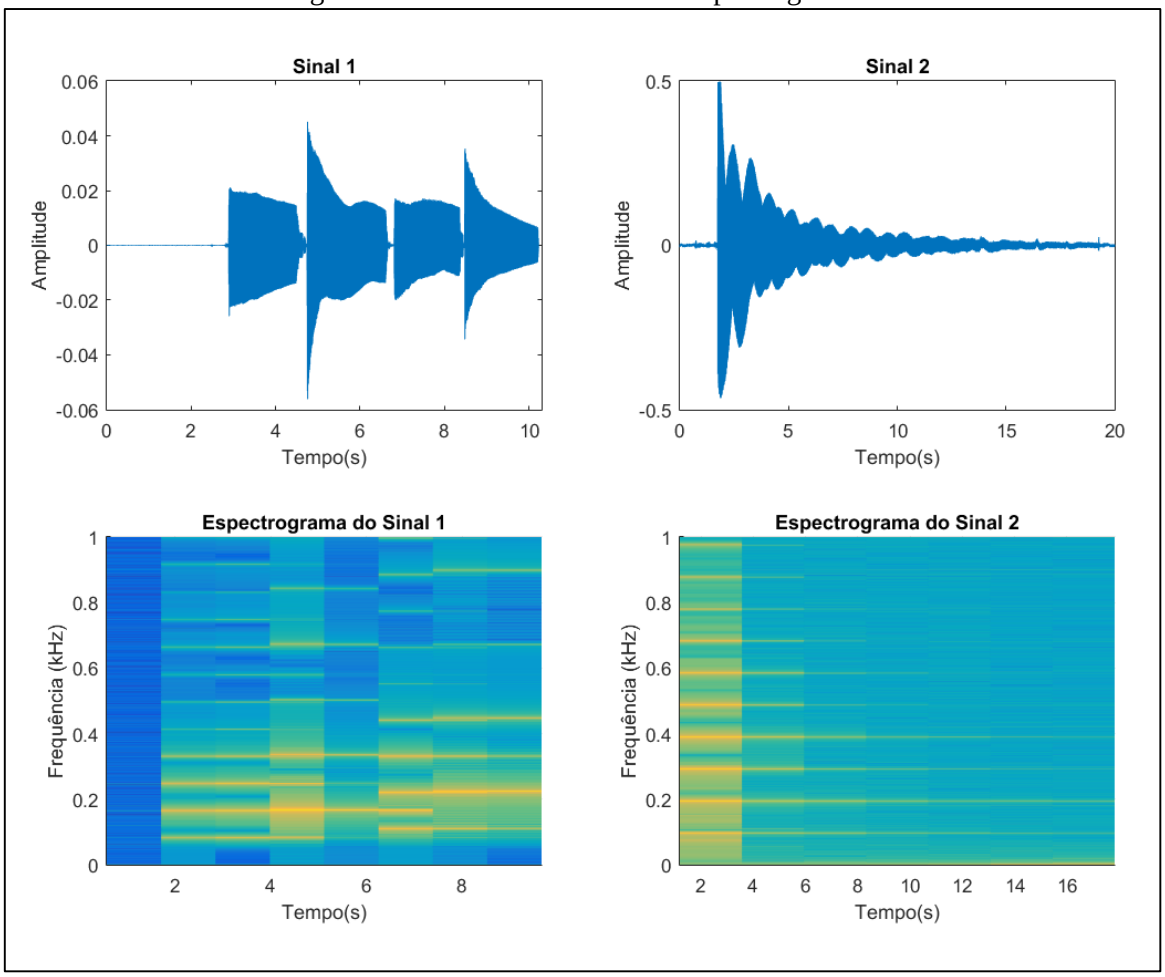
$$\frac{2\pi f_s}{N} = 2\pi\ \frac{rad}{s} \quad \Longrightarrow \quad f_s = N. \quad (21)$$

Considerando que para a aferição da DFT será usado uma técnica de FFT, e tais algoritmos funcionam muito mais rapidamente quando a ordem N é igual a uma potência de dois, o menor valor possível tanto para N quanto para f_s é 4096.

Com estes parâmetros definidos já é possível começar a realizar testes no MATLAB de modo a definir os restantes. Para esses testes foram usados dois arquivos de áudio: o primeiro é uma guitarra elétrica tocando quatro notas em sequência: mi_2, mi_3, $l\acute{a}_2$ e $l\acute{a}_3$, cujas

frequências fundamentais são, respectivamente, $82{,}4\ Hz$, $164{,}8\ Hz$, $110\ Hz$ e $220\ Hz$; o segundo consiste de um baixo elétrico tocando apenas uma nota, o sol_2, cuja fundamental é $98\ Hz$, mas com a diferença de que a nota é deixada soar até perder completamente sua energia. A Figura 4.2 mostra as formas de onda desses sinais e seus respectivos espectrogramas.

Figura 4.2 – Sinais de teste e seus espectrogramas.



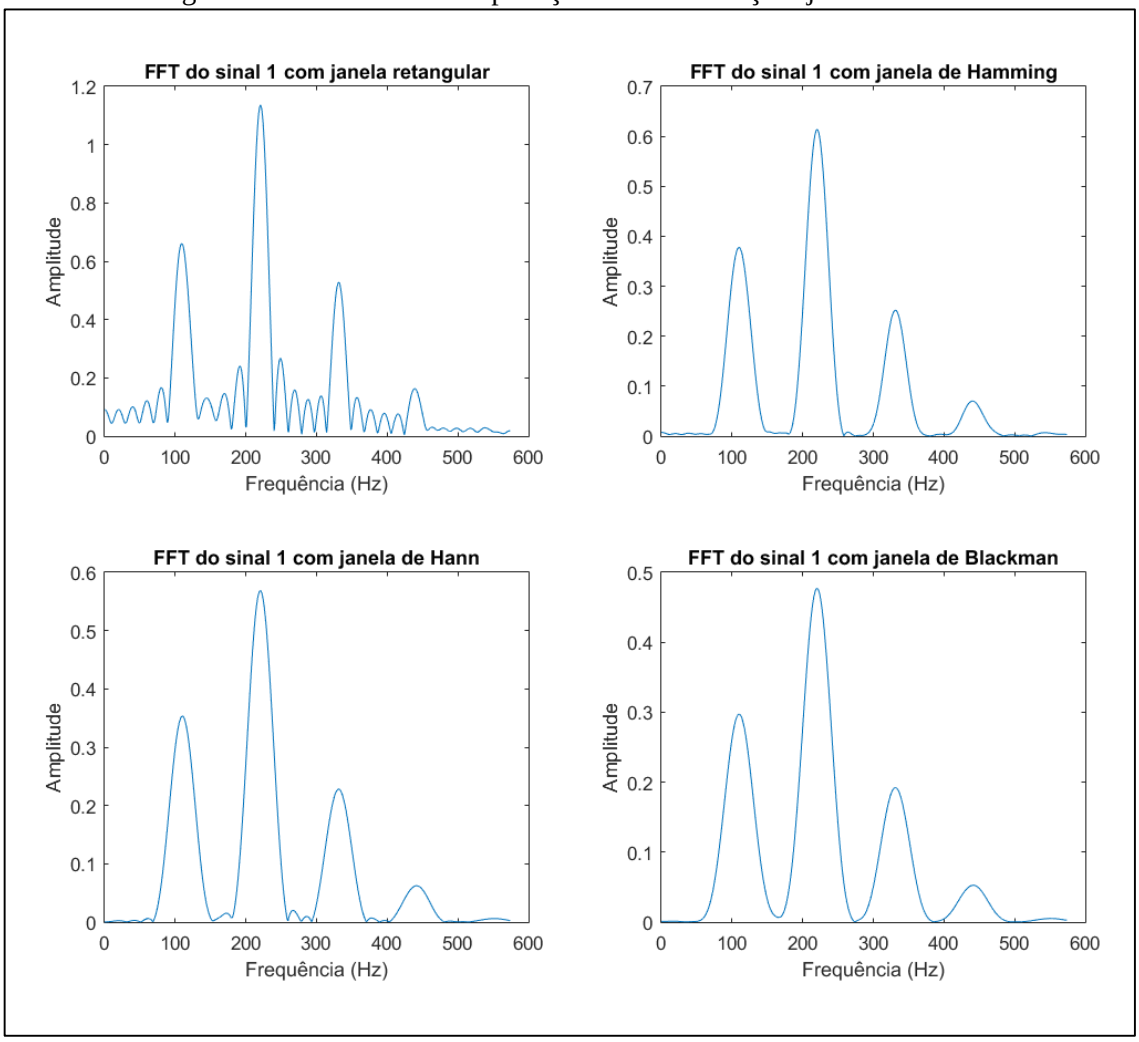
Fonte: elaborada pelo autor.

Do espectrograma do sinal 2 é possível notar que as harmônicas de ordem mais alta perdem energia mais rapidamente do que as de ordem mais baixa. Esta conclusão será importante um pouco mais à frente no texto.

Os parâmetros L, $w[n]$, H, a e r foram sendo definidos ao mesmo tempo, já que um acaba dependo do outro. Entretanto, discutir-se-á um só por vez neste texto, a fim de não alongá-lo excessivamente com todas as possibilidades de combinação. Os seguintes testes foram realizados utilizando trechos de sinal 1 do tamanho de uma janela.

Foram testadas três funções janelas: Hann, Hamming e Blackman. O objetivo é minimizar as ondulações geradas pela FFT ao mesmo tempo que alarga o mínimo possível os lobos principais da transformada, concentrando a energia no entorno das harmônicas. A Figura 4.3 mostra os resultados.

Figura 4.3 – Resultados da aplicação de várias funções janela ao sinal.



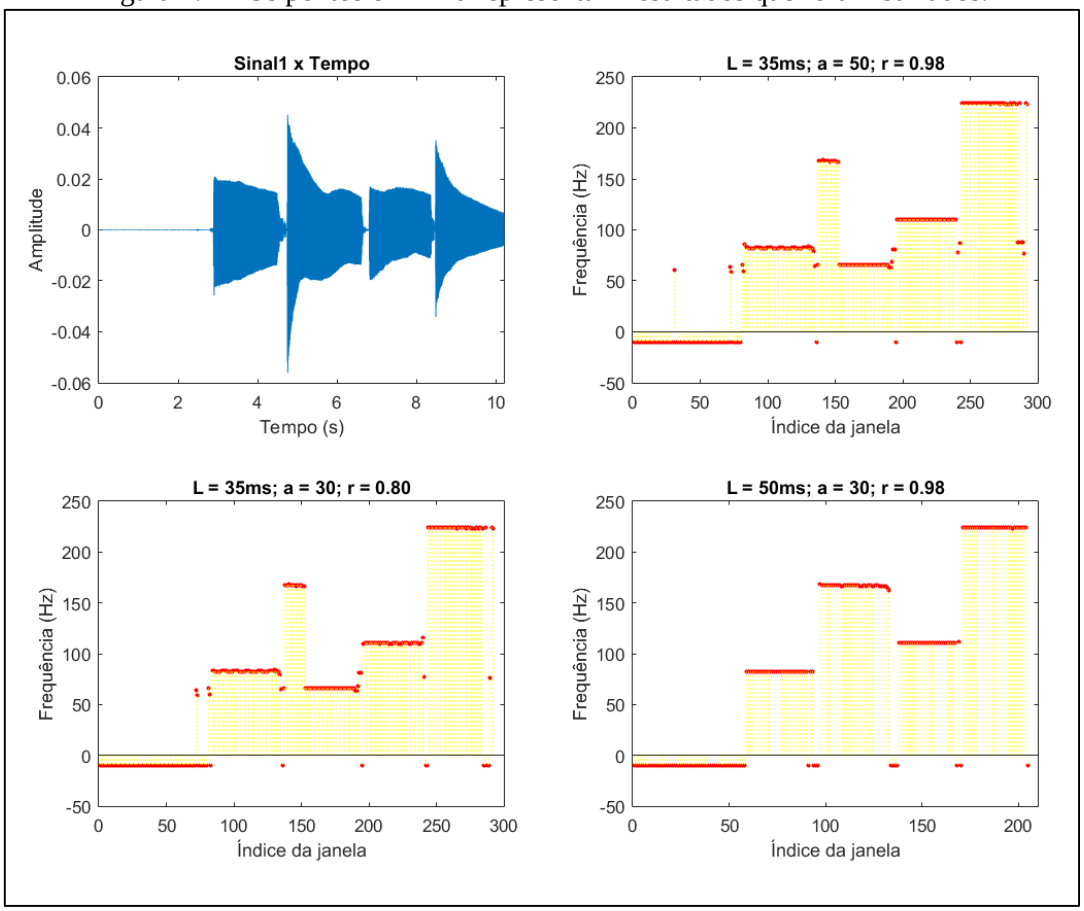
Fonte: elaborada pelo autor.

Pelos gráficos é possível ver que a janela de Blackman obtém os melhores resultados no quesito minimizar as ondulações, no entanto ao custo de alargar demasiadamente os lobos principais, a ponto de fazê-los juntarem-se. As janelas de Hann e Hamming obtiveram resultados melhores e muito similares entre si, de modo que é indiferente qual janela é usada. Assim sendo, a janela de Hann foi escolhida e será usada daqui para a frente.

Com a janela função escolhida deve-se, então, decidir-se o tamanho L da janela. Esse parâmetro influencia diretamente nos valores de a e r. O ideal é que ela seja o menor possível,

para que o afinador possa ser mais rápido. O problema é que quanto menor a janela, menos resolução tem-se em frequência e, dessa forma, os lobos principais passam a ficar mais largos; isso favorece a geração de resultados incorretos: como os lobos principais contém quase toda a energia do sinal, quantos mais largos eles forem, mais "frouxo" tem que ser o critério para descarte de resultados indesejados, o que pode acabar deixando alguns deles passar, ou então, se for mantido um critério mais rígido, muitos resultados corretos acabam sendo barrados. A Figura 4.4 mostra algumas combinação de L, a e r e os resultados.

Figura 4.4 – Os pontos em -10 representam resultados que foram barrados.



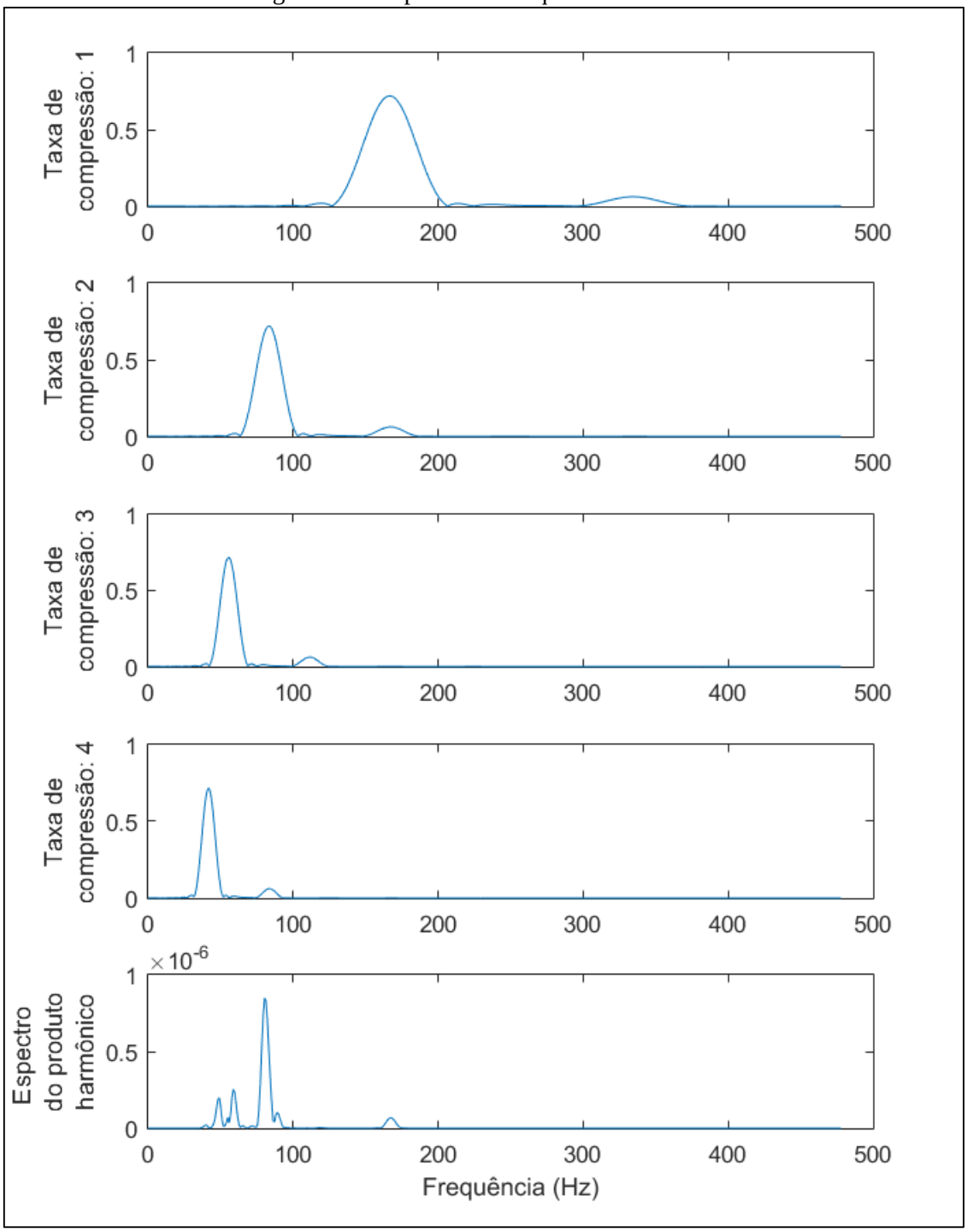
Fonte: elaborada pelo autor.

A partir do quarto gráfico conclui-se que com uma janela de $50\ ms$ é possível aplicar um critério bastante rígido e, mesmo assim, não barrar bons resultados. Com a janela de $35\ ms$, muitos resultados incorretos surgiram. Desse modo, os parâmetros ficaram assim: $L = 50\ ms$, $a = 30$ e $r = 0{,}98$.

O último parâmetro a ser definido é o H, a quantidade de vezes que o sinal é comprimido. Inicialmente, trabalhava-se com H igual a quatro, porém, notou-se que a quarta harmônica estava "atrapalhando". A explicação é simples: como as harmônicas de ordem

superior perdem energia mais rapidamente do que as de ordem inferior (rever Figura 4.2), essa quarta harmônica acaba, algumas vezes, “matando” o grande pico do resultado do espectro do produto harmônico, deixando, como ponto de máximo, um pico que marca metade da frequência correta, gerando um resultado indesejado que poderia, até mesmo, passar pelo critério de descarte. A Figura 4.5 mostra isso mais claramente.

Figura 4.5 – O problema da quarta harmônica.

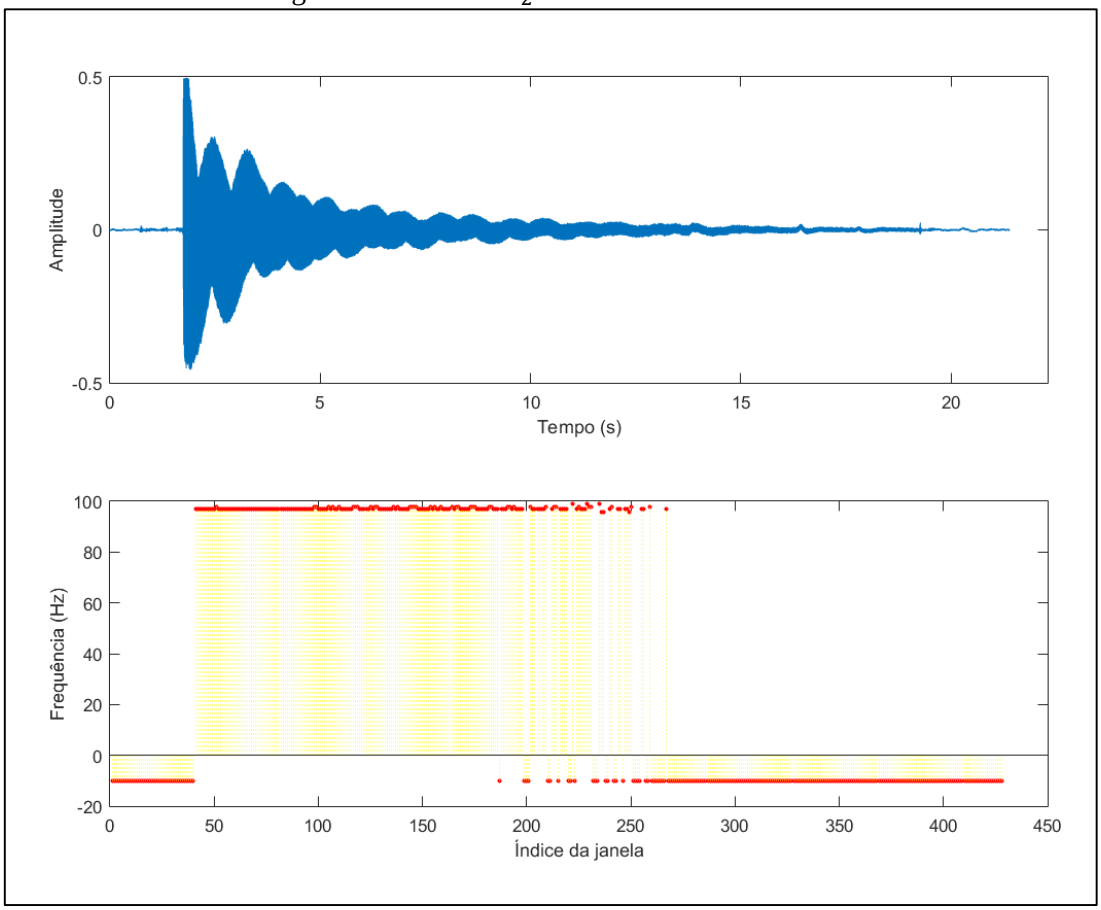


Fonte: elaborada pelo autor.

A solução para contornar esse problema é usar a taxa de compressão máxima igual a três, ou $H = 3$. No entanto, como na grande maioria das vezes a quarta harmônica possui grande parte da energia total do sinal, é necessário considerá-la para o cálculo de r, portanto D permanece sendo quatro.

A Figura 4.6 mostra um último teste, com o trecho de áudio com uma nota soando até voltar ao silêncio. Destaca-se o fato de que, ao passo que o sinal vai perdendo a energia, o algoritmo começa a falhar em detectar sua altura, porém, em nenhum momento, um resultado incorreto é gerado, o que é de suma importância para um afinador automático. Esse sinal também mostrou necessária uma última modificação: como, em seu início, a quinta e sexta harmônicas detém boa parte da energia, foi necessário aumentar o valor de D para seis, seu valor final. Os valores de f_s e N não precisam ser alterados, no entanto, apesar dessa modificação.

Figura 4.6 – Nota sol_2 soando até ficar silêncio.



Fonte: elaborada pelo autor.

Os códigos do MATLAB desenvolvidos para os testes estão apresentados como anexos ao fim desta monografia.

5 IMPLEMENTAÇÃO E TESTE NO PSOC

A última etapa do desenvolvimento do sistema é a implementação do algoritmo em um microcontrolador e o projeto do hardware de apoio. Após isso é apresentado o teste final, plugando-se um baixo elétrico à entrada de áudio do sistema.

O microcontrolador escolhido foi o PSoC (sistema-em-um-chip programável, na sigla em inglês), desenvolvido pela companhia americana *Cypress Semiconductor Corporation*. O PSoC é único no que diz respeito a ser um sistema-em-um-chip programável. Ele contém um microcontrolador, memória, blocos analógicos e digitais configuráveis e roteamento e interconexão programáveis. Isso permite ao projetista trabalhar com aplicações de sinais misto naturalmente. Tais blocos podem ser configurados como temporizadores, contadores, PWMs, ADCs, DACs, controlador USB, amplificadores operacionais *etc*. Além disso, o aplicativo PSoC Creator permite desenvolver facilmente o sistema, por meio de um editor gráfico e um poderoso *debugger*, além do ambiente de programação.

Por tudo isso, a escolha pelo PSoC é algo natural para um trabalho como este apresentado aqui. A versão do dispositivo usado é a 5/5LP. Esse PSoC contém uma CPU ARM Cortex-M3 de 32 bits, 64 KB de memória SRAM para dados e 256 KB de memória *flash* para instruções e constantes. A programação é toda feita em linguagem C (alguns blocos digitais precisam ser programados em *assembly*, mas isso não chegou a ser usado neste trabalho).

O sistema utiliza um motor de passo, que infelizmente não pôde ser acoplado ao instrumento, o que não impede que ele, sistema, possa ser testado e comprovado que funcione.

5.1 Configuração e programação do PSoC

Essa tarefa consiste em traduzir o código já testado no MATLAB para C e também programar os controles necessários à operação do PSoC e os blocos internos necessários.

5.1.1 Configuração dos blocos internos ao PSoC

O sistema projetado utiliza-se de cinco componentes internos programáveis do PSoC: um conversor analógico/digital delta-sigma, um amplificador operacional, um controlador de acesso direto à memória e dois registradores.

O primeiro a ser configurado é o conversor A/D. Apesar de o PSoC também possuir dois conversores A/D SAR, o delta-sigma foi escolhido por ele apresentar características mais compatíveis com as necessárias para processamento de sinais e áudio, como uma resolução maior, alta velocidade de conversão e presença de filtro interno para eliminação de ruído. (CYPRESS SEMICONDUCTOR CORPORATION, 2012). O modo de conversão foi definido como contínuo, já que o ideal é que sempre haja dados no *buffer* de saída dele prontos para o processamento. A resolução foi ajustada em 16 bits, que é um valor padrão em aplicações de áudio (8 bits introduziria muito ruído de quantização, e qualquer valor entre isso seria desperdício de banda, já que a transferência de memória é feita em 8 ou 16 bits). A taxa de amostragem é 4096 amostras por segundo, como foi definido anteriormente.

O modo de entrada do conversor é diferencial, de modo a permitir valores menores e maiores que zero no *buffer* de saída. Nesse modo a entrada negativa do ADC é subtraída da positiva. Como o PSoC não trabalha com tensões inferiores a zero, é necessário aplicar uma tensão de referência maior que zero na entrada negativa, e assegurar que sinal, que está ligado à entrada positiva, apresente o mesmo nível DC dessa tensão de referência. Essa tensão é gerada internamente no PSoC e foi configurada como $Vdda/2 = 1{,}65\,V$. Dessa forma a faixa de entrada foi definida como $\pm 1{,}024\,V$, a maior possível (a seguinte, $\pm 2{,}048\,V$, extrapolaria o valor máximo $Vdda = 3{,}3\,V$, já que $1{,}65\,V + 2{,}048\,V = 3{,}7\,V$, e também o limite inferior de $0\,V$).

O segundo bloco adicionado é um amplificador operacional na configuração seguidor de tensão. Este AmpOp serve para fazer aparecer a tensão de referência do ADC em um pino externo do PSoC. Essa tensão é usada para aplicar um nível DC ao sinal de áudio, conforme discutido no parágrafo anterior.

O bloco seguinte é o de controle de acesso direto à memória (DMA). O DMA serve para fazer transferências de dados sem a intervenção da CPU, poupando-a em processamento. É o método ideal para transferir conversões do ADC para a memória para serem posteriormente processadas. O DMA foi configurado conforme nota de aplicação da própria Cypress (ANU e MOHAN, 2010). Toda vez que o ADC converte uma amostra, ele gera uma interrupção. O DMA foi programado então para transferir dois bytes (16 bits) da saída do ADC para um vetor na memória a cada sinal de "pronto" do conversor. Depois da transferência, o DMA incrementa a posição da memória em dois.

Outra função útil a esta aplicação que o DMA realiza é dividir o sinal de entrada em janelas. Ele foi programado para, a cada 205 transferências ($205 \times (1/4096\ Hz) = 50\,ms$),

gerar uma interrupção avisando que um janela foi transferida, e recomeçar a preencher o vetor do início novamente.

Os dois últimos blocos são registradores de controle. Um serve para gerar um sinal de em um pino externo do PSoC usado para teste: medir o tempo que o programa demora para rodar. O outro serve para ativar as bobinas do motor.

5.1.2 Programação em linguagem C do algoritmo

Como este dispositivo é dotado de uma CPU ARM Cortex-M3, decidiu-se usar a Interface Padrão de Software para Microcontroladores Cortex (CMSIS). Esse software é desenvolvido pela própria ARM e foi desenvolvido para ser uma camada de abstração de hardware que funcione com quaisquer dispositivos que usam CPUs da linha Cortex, independente de fabricante (CMSIS - Cortex Microcontroller Software Interface Standard - ARM, 2014).

A CMSIS possui uma biblioteca de DSP com diversas funções otimizadas para rodar nos processadores Cortex, como algoritmos de FFT, filtros, estatísticas *etc*. Com essa biblioteca, a tarefa de traduzir o algoritmo do MATLAB ficou facilitada. Além do mais, por ser um software testado e conhecido, o sistema ganhou em robustez.

Foi necessário tomar um cuidado extra para não extrapolar a quantidade de memória do dispositivo. O vetor que recebe o resultado da FFT, por exemplo, por ser do tipo *float*, possui 4096 palavras de 4 bytes tanto para a parte real, quanto para a imaginária, totalizando 32 KB. Isso já dá metade da memória de dados. Sendo assim, esse vetor é reutilizado quando do cálculo do espectro de frequência. O vetor com a janela de Hann, por ser um dado estático, foi declarado como *const*, de modo a ser armazenado na memória *flash*, economizando mais espaço para o programa utilizar.

Com relação ao código escrito para o MATLAB, existem poucas alterações. No início é necessário inicializar todos os componentes usados, como o conversor A/D e o DMA. O algoritmo também foi modificado de modo a continuar rodando até que a corda da guitarra esteja afinada. A fim de minimizar pequenas alterações nos resultados da detecção da altura, uma média dos três últimos é usada como critério, sendo que a variância entre eles não pode ser maior do que dois.

Também foi escrita uma rotina que calcula quantos passos o motor de passo tem que dar para girar as cravelhas até a posição necessária para que a corda da guitarra soe na nota certa. O algoritmo, conforme ele foi implementado no PSoC, consiste das seguinte etapas:

1. Inicialização dos componentes do PSoC;
2. Espera por sinalização do DMA de que uma janela está pronta para o processamento;
3. Converte janela de inteiros para ponto flutuante;
4. Multiplicação do sinal pela função janela de Hann;
5. Transformação do sinal para o domínio da frequência por meio de uma FFT de 4096 pontos;
6. Obtenção do módulo da FFT;
7. Geração do vetor com o HPS;
8. Descoberta do ponto de máximo M do vetor HPS, que é o local da candidata à f_0;
9. Através do teorema de Parseval, cálculo da energia total do sinal e também da energia nos entornos das seis primeiras possíveis harmônicas dele, com base na candidata à f_0;
10. Se a razão entre a energia nesses arredores e a energia total for maior ou igual do que 0,98, então aceita o resultado como frequência fundamental, caso contrário retorna -1;
11. Atualiza um vetor com os três últimos resultados obtidos;
12. Cálculo da média e variância desses resultados;
13. Se a média estiver próxima ao resultado desejado, a corda está afinada, então termina;
14. Se a média estiver longe do desejado e a variância for menor que dois, chama a rotina que aciona os motores e volta à etapa 2;
15. Se a variância for maior que dois, nenhuma altura foi detectada, volta à etapa 2.

Uma cópia dos códigos em linguagem C usados neste trabalho estão em anexo ao fim desta monografia.

5.2 Projeto do hardware auxiliar

A parte eletrônica externa ao PSoC não chega a ser muito complexa. Pode ser dividida em dois grupos: um para tratar o sinal de áudio proveniente do instrumento e outro que lida com o motor de passo.

O tratamento do sinal da guitarra é feito em três estágios. O primeiro é necessário para eliminar o nível DC presente nele. Isso é feito com um arranjo com um resistor ligado ao terra e um capacitor na entrada do sistema, formando um filtro passa-baixas. Os valores de R_5 e C_3 foram ajustados de modo que a frequência de corte seja bem baixa.

O segundo estágio consiste em um amplificador inversor. Ele é necessário pois o sinal proveniente da guitarra possui uma amplitude muito baixa, que não explora toda a faixa de entrada do ADC, deixando espaço para erros de quantização. O fato de ele inverter o sinal não atrapalha, pois o que interessa ao sistema é a frequência do sinal, e não sua fase.

Nesse segundo estágio também foi adicionado um capacitor ao arranjo, tornando o amplificador também um filtro passa-baixas. Esse filtro foi projetado como um *antialiasing*, apesar de que o ADC delta-sigma não necessita de um, já que ele usa sobreamostragem para evitar erros de quantização, e possui filtro interno (CYPRESS SEMICONDUCTOR CORPORATION, 2012). Esse passa-baixas acaba cumprindo função de evitar que ruídos externos interfiram nas medições, e aumenta a estabilidade do sistema. Seu ganho G e frequência de corte f_c são:

$$G = -\frac{R_4}{R_3} = 17{,}85\ \frac{V}{V}; \tag{22}$$

$$f_c = \frac{1}{2\pi R_4 C_2} = 1326{,}29\ Hz. \tag{23}$$

O amplificador operacional empregado nesse estágio foi um OPA134, que é um dispositivo de alta performance projetado especificamente para aplicações de áudio (TEXAS INSTRUMENTS INCORPORATED, 2014).

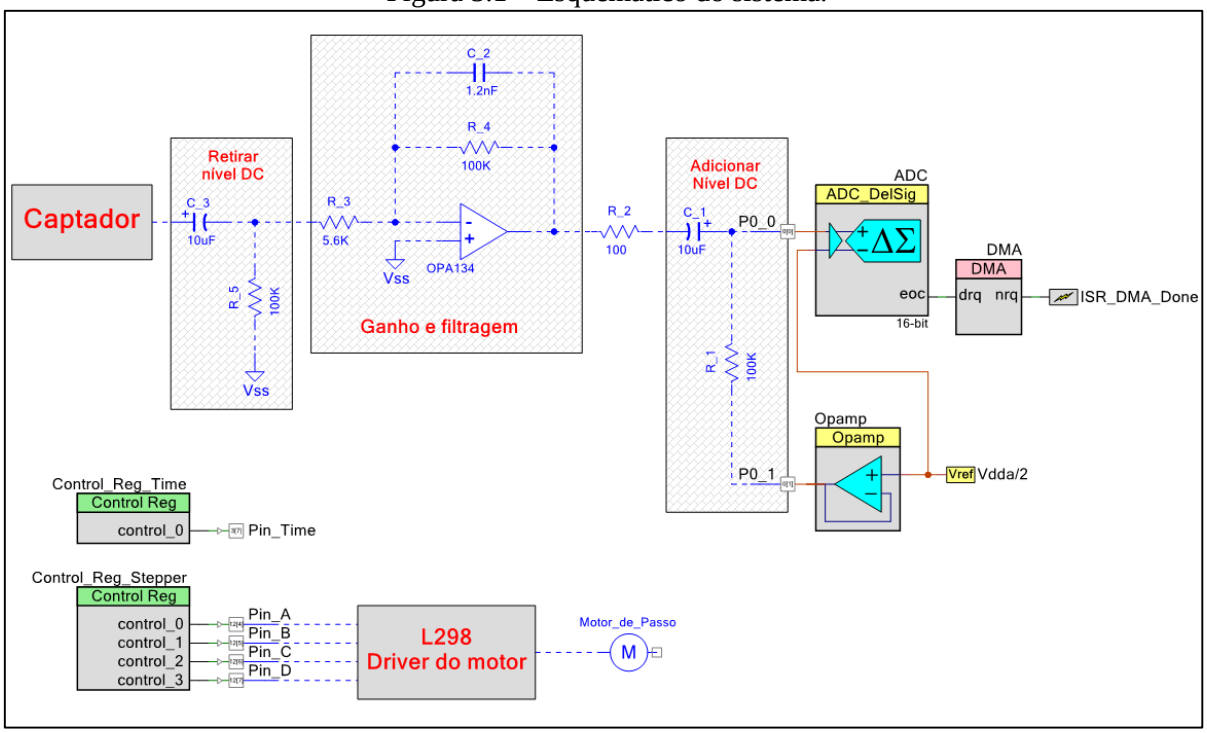
O terceiro e último estágio é muito similar ao primeiro, mas ao invés de retirar o nível DC ele adiciona um, de modo a fazer o sinal de áudio ter o mesmo nível DC que a tensão de referência do conversor A/D. Isso é feito ligando ao resistor R_1 à saída do seguidor de tensão interno do PSoC. Os componentes R_1 e C_1 têm os mesmos valores dos do primeiro estágio, de modo a ter, outrossim, uma frequência de corte muito baixa.

O segundo grupo de componentes inclui apenas dois dispositivos. O primeiro é o driver de motor de passo L298, que é capaz de controlar motores de passo bipolares (STMICROELECTRONICS, 2000). As entradas do driver estão conectadas às saídas de um

registrador interno ao PSoC, que é usado para controlar o movimento do motor. O segundo dispositivo é próprio motor, um modelo M42SP-4, que é bastante preciso, possuindo um ângulo de passo de apenas 3,75 graus (MITSUMI).

Na Figura 5.1 está o diagrama completo dos componentes tanto externos como internos ao PSoC, e suas interconexões.

Figura 5.1 – Esquemático do sistema.



Fonte: elaborada pelo autor.

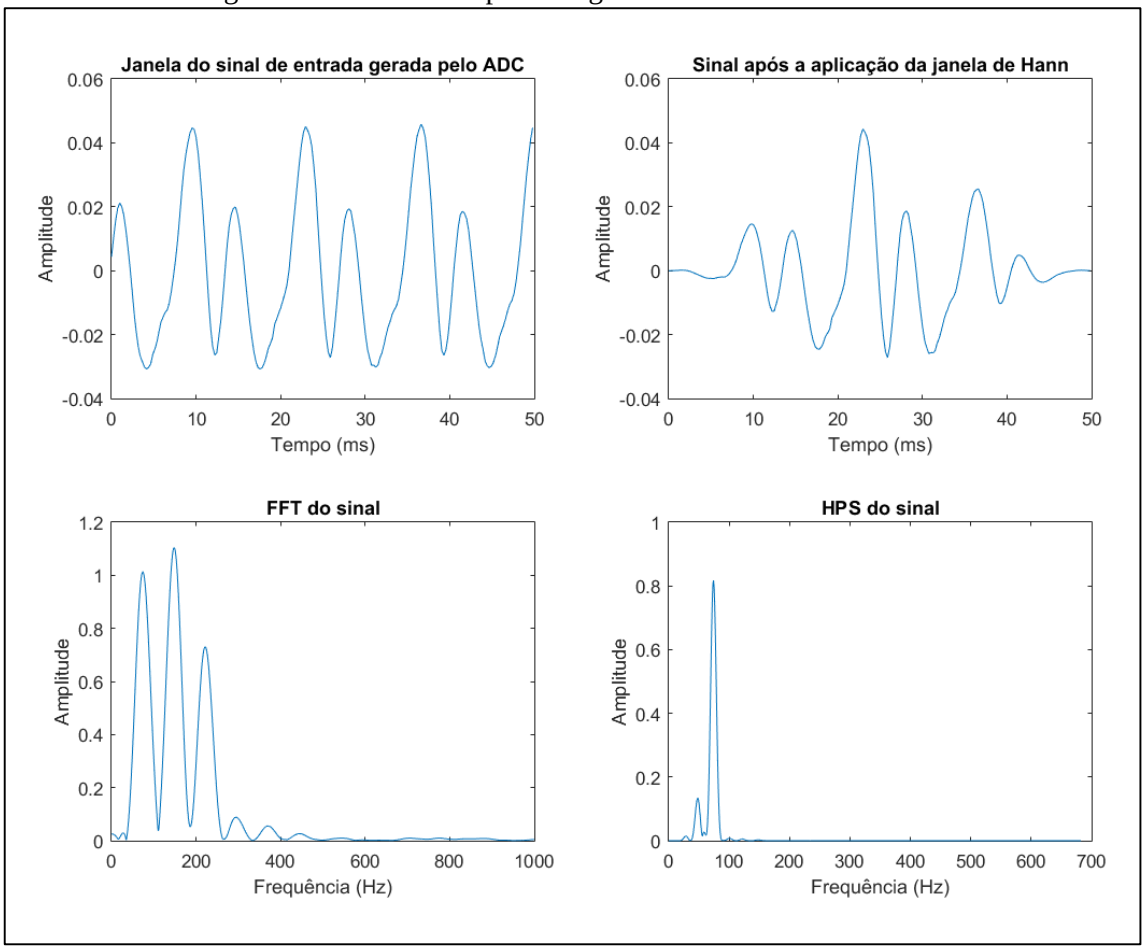
5.3 Resultados

Para testar o sistema foi plugado, na ausência de uma guitarra elétrica comum, um baixo elétrico à entrada de áudio. Usando o *debugger* do aplicativo de desenvolvimento do PSoC, foi possível extrair os dados gerados internamente ao programa. Na Figura 5.2 vê-se, a partir de gráficos, os resultados do processamento ao tocar-se a segunda corda do baixo, $ré_2 = 73{,}42\ Hz$ e na Figura 5.3, ao tocar-se a primeira corda, $sol_2 = 98\ Hz$. Os resultados estão dentro do esperado.

Na Figura 5.4 vê-se a captura de tela de um osciloscópio que foi usado para medir o tempo de execução de programa. Um pulso começa quando o *loop* principal inicia e termina quando esse *loop* finaliza. É possível ver vários pulsos em sequência na figura. Como cada

divisória horizontal do gráfico representa $250\ ms$, conclui-se que o algoritmo demora em torno de $800\ ms$ no total para rodar.

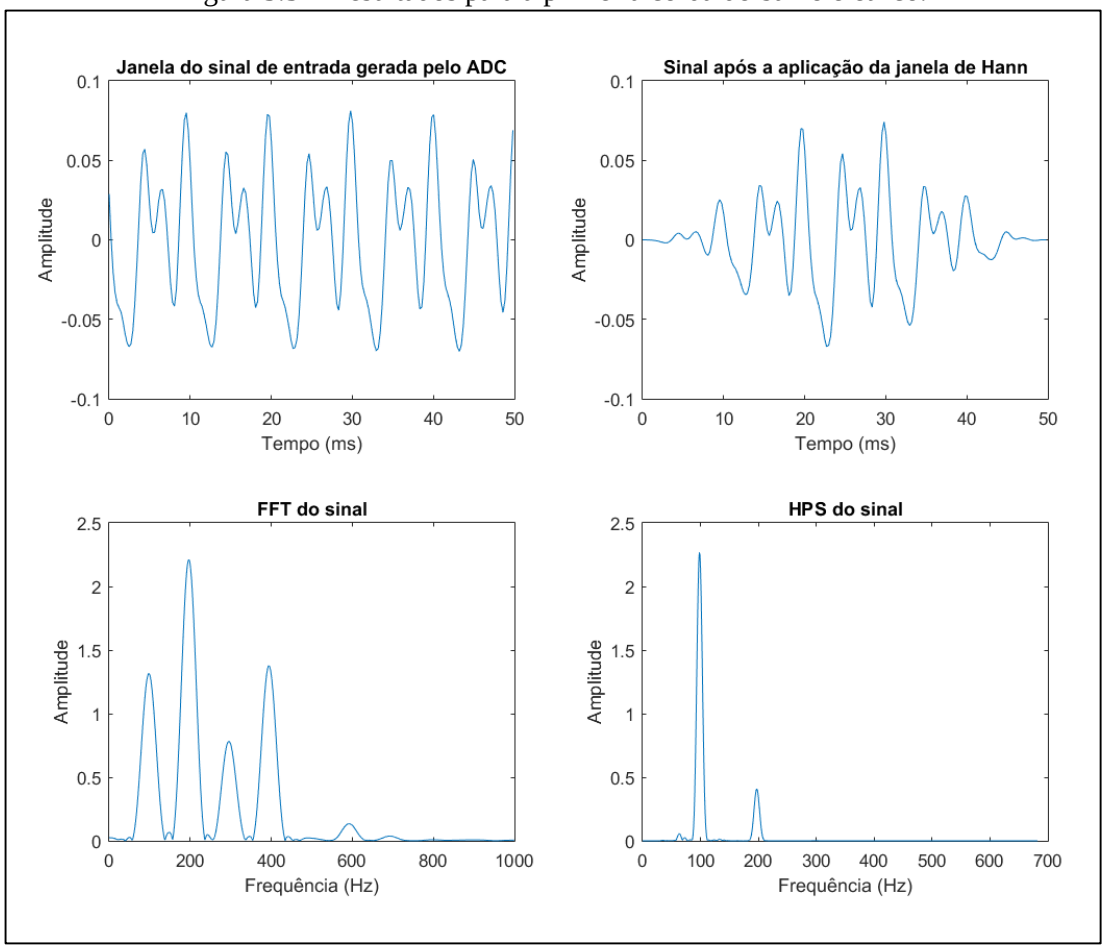
Figura 5.2 – Resultados para a segunda corda do baixo elétrico.



Fonte: elaborada pelo autor.

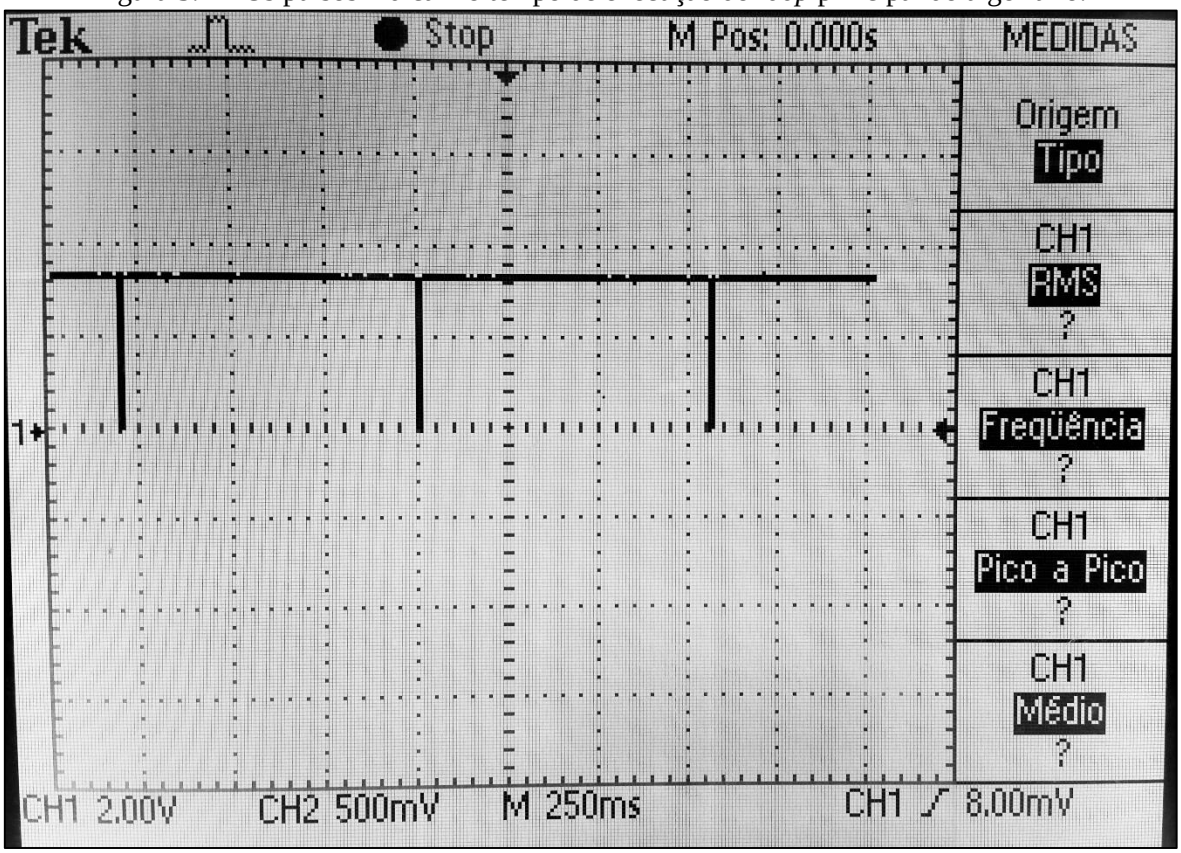
Como não foi possível acoplá-lo ao baixo elétrico, o motor foi testado da seguinte forma: um valor de referência, trinta passos por hertz, foi configurado. Isso significa que a cada hertz de diferença entre a frequência captada e a desejada, o motor gira trinta passos de modo a aumentar ou diminuir a tensão da corda. O sistema foi então estimulado pelo baixo desafinado, e o motor respondeu de forma adequada, girando conforme a diferença das frequências, e também em direções opostas quando essa diferença era positiva ou negativa.

Figura 5.3 – Resultados para a primeira corda do baixo elétrico.



Fonte: elaborada pelo autor.

Figura 5.4 – Os pulsos indicam o tempo de execução do *loop* principal do algoritmo.



Fonte: elaborada pelo autor.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi apresentado o desenvolvimento de um sistema digital para a afinação automática de uma guitarra elétrica. Este não é um tópico novo. Como mostrado, há décadas existem projetos e tentativas de construir tais aparelhos, com mais ou menos sucesso. No entanto, o fato de a imensa maioria de guitarristas ainda afinar manualmente seus instrumentos mostra que, talvez, ainda não se chegou ao nível de excelência exigido pelos músicos, e portanto ainda é necessário e possível avançar nesta área.

O sistema aqui descrito exibiu resultados satisfatórios. Os testes mostraram que ele cumpriu seus pré-requisitos descritos na Introdução. O PSoC, junto ao seu ambiente de desenvolvimento PSoC Creator, mostrou-se muito flexível e possibilitou um trabalho bastante preciso e um controle muito amplo sobre todas as variáveis do projeto.

Uma série de melhorias podem ser aplicadas ao sistema proposto, de modo a aumentar seu desempenho e/ou sua precisão. A mais óbvia parecer a diminuição do tamanho da FFT. Como este algoritmo representa a maior parte do tempo de execução do código, a diminuição de sua ordem representaria grande melhora no desempenho. Para não perder em resolução, teria que se diminuir na mesma proporção a frequência de amostragem. No entanto, é preciso tomar cuidado para não acabar cortando frequências importantes para o bom funcionamento do algoritmo.

Uma técnica que possibilitaria diminuir a ordem da FFT e manter a frequência de amostragem sem perder em resolução seria usar um método de interpolação adequado, que poderia ser aplicado apenas nas faixas de frequência de interesse, em contraste ao *zero-padding*, que acaba interpolando todo o espectro, desperdiçando tempo de processador.

Por falar em processador, o uso de um com maior poder de processamento, evidentemente, faria o tempo de execução diminuir. É claro que muitas vezes deseja-se que o sistema tenho baixo gasto energético, e um processador muito potente poderia entrar em conflito com essa limitação.

Outra possível melhoria, agora em específico para o PSoC, seria o uso do bloco de filtro digital (DFB, na sigla em inglês). Esse bloco é programado em *assembly*, e pode funcionar como um coprocessador DSP. Ele poderia ser usado para acelerar partes do algoritmo, em especial a FFT, que apesar de ser feita neste sistema por funções específicas e otimizadas para microcontroladores ARM, roda inteiramente em cima do processador. O uso do DFB poderia liberar tempo da CPU e introduzir até mesmo um certo paralelismo ao programa.

Só após otimizações (essas sugeridas ou outras) é que o sistema estaria apto a afinar uma guitarra com ponte flutuante. Ele precisa ser rápido o suficiente para corrigir a mesma corda várias vezes antes que ela pare de vibrar, já que a tensão de uma corda influencia a tensão de todas as outras.

REFERÊNCIAS

ADAMS, C. **Method for Automatically Tuning a String Instrument, Particularly an Electric Guitar**. US20080105107 A1, 8 Maio 2008.

ANU, M. D.; MOHAN, A. **PSoC® 3 and PSoC 5LP - ADC Data Buffering Using DMA**. [S.l.]: [s.n.], v. único, 2010. Disponível em: <http://www.cypress.com/?docID=50035>.

CMSIS - Cortex Microcontroller Software Interface Standard - ARM. **ARM**, 2014. Disponível em: <http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php>. Acesso em: 02 jul. 2015.

CUMBERLAND, T. **Apparatus for tuning stringed instruments**. US6278047 B1, 21 Agosto 2001.

CYPRESS SEMICONDUCTOR CORPORATION. **Delta Sigma Analog to Digital Converter (ADC_DelSig)**. 2.30. ed. [S.l.]: [s.n.], v. único, 2012. Disponível em: <http://www.cypress.com/?docID=36745>.

DE LA CUADRA, P.; MASTER, A.; SAPP, C. **Efficient Pitch Detection Techniques for Interactive Music**. Proceedings of the International Computer Music Conference 2001. Havana: [s.n.]. 2001.

DIAS, B.; VENTURA, R.; GASPAR, J. **Automatic Transcription of Musical-Whistling:** Comparing Pitch Detection Methods. Proc. of IV Jornadas de Engenharia Electrónica e Telecomunicações e de Computadores. Lisboa: [s.n.]. 2008. p. 186-191.

EVEREST, F. A.; POHLMANN, K. C. Pitch versus Frequency. In: _____ **Master Handbook of Acoustics**. 6ª. ed. Nova Iorque, Chicago, São Francisco, Atenas, Londres, Madri, Cidade do México, Milão, Nova Délhi, Singapura, Sydney, Toronto: McGraw-Hill, v. único, 2015. Cap. 4.8. ISBN 9780071841047.

FERREIRA, A. B. de H. altura. In: _____ **Novo Dicionário da Língua Portuguesa**. 2ª. ed. Rio de Janeiro: Nova Fronteira, v. único, 1986. p. 94. ISBN 85-209-0411-4.

HALLIDAY, D.; RESNICK, R.; WALKER, J. **Fundamentos de Física**. 8. ed. Rio de Janeiro: LTC, v. 3, 2009. 268-269 p.

HEDRICK, D. W. **String instrument tuning apparatus**. US4088052 A, 9 Maio 1978.

LONG, P. G. et al. **Tuning of musical instruments**. US6184452 B1, 6 Fevereiro 2001.

MASTER, A. S. **Speech Spectrum Modelling from Multiple Sources**. Universidade de Cambridge, Departamento de Engenharia, Cambridge, 2000. 8-10 p. Disponível em: <https://ccrma.stanford.edu/~asmaster/MPhilThesis/mybook2.pdf>.

MILANO, L. M.; RASTEGAR, J.; KHORRAMI, F. **Automatic string instrument tuner**. US5767429 A, 16 Junho 1998.

MILLER, M. D.; STROCK, J. M. **Device and method for automatically tuning a stringed musical instrument**. US5323680 A, 28 Junho 1994.

MINNICK, G. B. **Self tuning tail piece for string instruments**. US4584923 A, 29 Abril 1986.

MITSUMI. **M42SP-4**. [S.l.]: [s.n.]. Disponível em: <http://www.danielstolfi.com/recursos/M42SP-4.pdf>.

NAVE, R. Standing Waves on a String. **HyperPhysics**, 2012. Disponível em: <http://hyperphysics.phy-astr.gsu.edu/hbase/waves/string.html>. Acesso em: 04 Novembro 2014.

NOLL, A. M. **Pitch Determination of Human Speech by the Harmonic Product Spectrum, the Harmonic Sum Spectrum and a Maximum Likelihood Estimate**. Proceedings of the Symposium on Computer Processing in Communications. Nova Iorque: Polytechnic Press. 1970. p. 779-797.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. **Discrete-time signal processing**. 2ª. ed. Nova Jersey: Prentice-Hall, v. único, 1998. ISBN 0-13-754920-2.

OUDSHOORN, M.; MOLER, J.; AKHAVEIN, R. G. **Method and system for automatically tuning a stringed instrument**. US6437226 B2, 20 Agosto 2002.

RABINER, L. et al. A comparative performance study of several pitch detection algorithms. **IEEE Transactions on Acoustics, Speech and Signal Processing**, v. 24, n. 5, p. 399-418, Outubro 1976. ISSN 0096-3518. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1162846&isnumber=26120>.

RABINER, L. R. On the Use of Autocorrelation Analysis for Pitch Detection. **IEEE Transactions on Acoustics, Speech and Signal Processing**, v. 25, n. 1, p. 24-33, Fevereiro 1977. ISSN 0096-3518. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1162905>.

SCHOLZ, D. T. **Automatic tuning device**. US4426907 A, 24 Janeiro 1984.

SKINN, N. C.; FREELAND, S. J. **Automatic musical instrument tuning system**. US4909126 A, 20 Março 1990.

STMICROELECTRONICS. **L298**. [S.l.]: [s.n.], 2000. Disponível em: <http://www.st.com/web/en/resource/technical/document/datasheet/CD00000240.pdf>.

SUJATHA, C. Cepstrum (Quefrency Domain) Analysis. In: _____ **Vibration and Acoustics:** Measurement and Signal Analysis. Nova Iorque, Chicago, São Francisco, Atenas, Londres, Madri, Cidade do México, Milão, Nova Délhi, Singapura, Sydney, Toronto: McGraw-Hill, v. único, 2010. Cap. 8.15. ISBN 9780070148789.

TEXAS INSTRUMENTS INCORPORATED. **OPA134**. Dallas: [s.n.], v. único, 2014. Disponível em: <http://www.ti.com/lit/ds/sbos058/sbos058.pdf>.

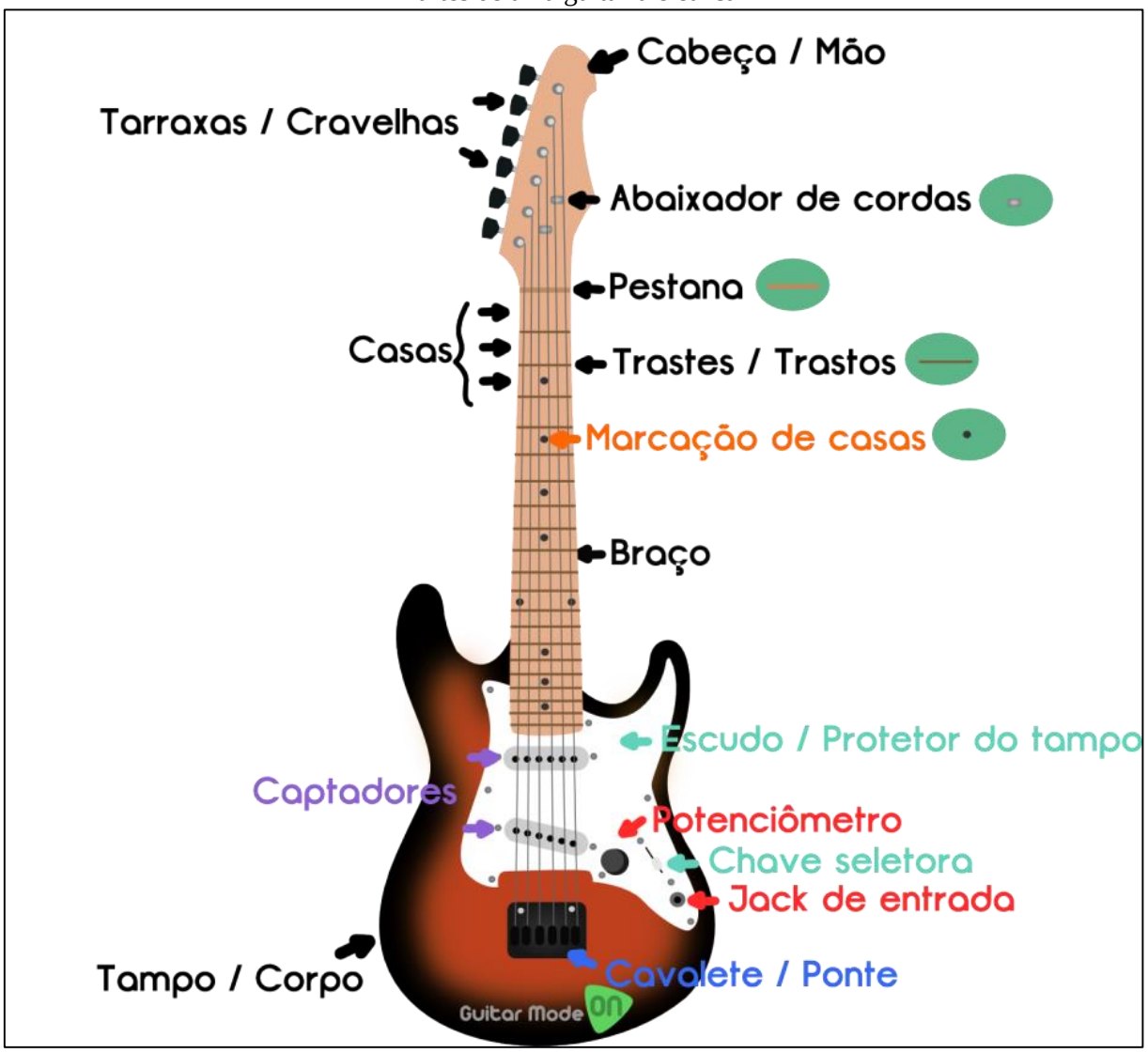
WHITAKER, J.; BENSON, B. Pìtch. In: _____ **Standard Handbook of Audio Engineering**. 2ª. ed. Nova Iorque, Chicago, São Francisco, Atenas, Londres, Madri, Cidade do México, Milão, Nova Délhi, Singapura, Sydney, Toronto: McGraw-Hill, v. único, 2001. Cap. 4.4. ISBN 9780070067172.

WHITTALL, R. J.; DENT, N. A.; LAMBERT, A. T. **Tuning means for tuning stringed instruments, a guitar comprising tuning means and a method of tuning stringed instruments**. US6415584 B1, 9 Julho 2002.

WYNN, D. S. **Electormechanical tuner for stringed instruments**. US5886270 A, 23 Março 1999.

GLOSSÁRIO

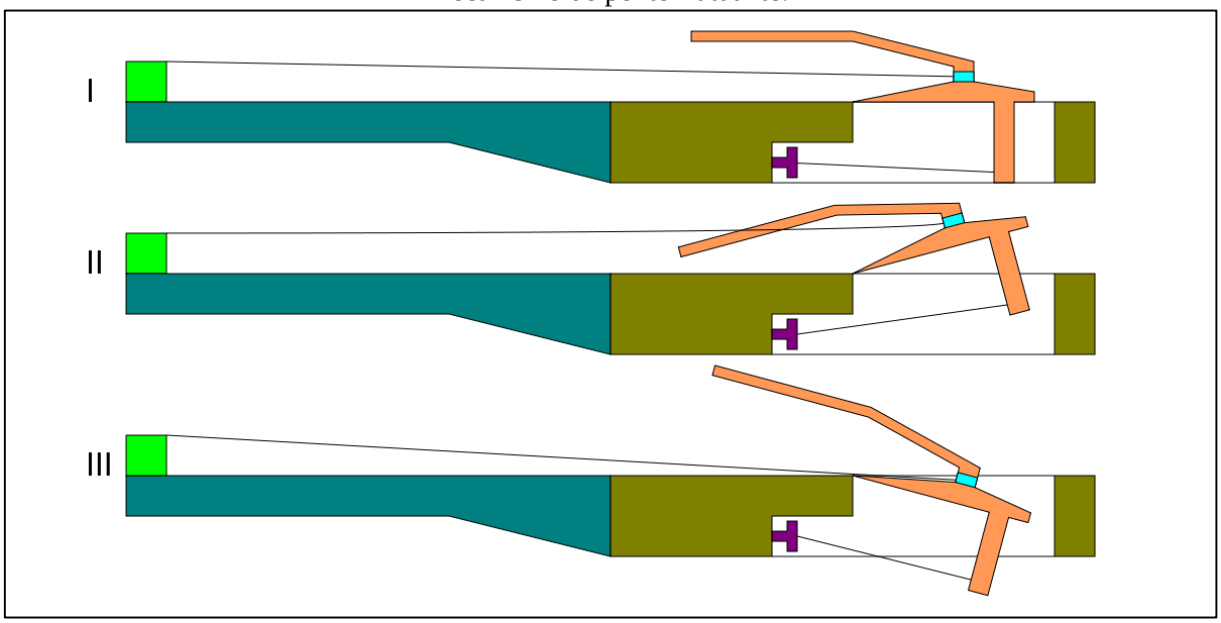
Partes de uma guitarra elétrica



Fonte: Guitar Mode ON[2]

[2] Disponível em <https://guitarmodeon.wordpress.com/tag/guitarra-eletrica/>. Acesso em julho de 2015.

Mecanismo de ponte flutuante.



Fonte: Vibrato systems for guitar – Wikipedia.[3]

[3] Disponível em <https://upload.wikimedia.org/wikipedia/commons/thumb/0/05/Floyd_Rose.svg/1200px-Floyd_Rose.svg.png>. Acesso em julho de 2015.

APÊNDICE A – CÓDIGO DE MATLAB

```
% Código para a análise de uma janela de áudio

clear

% Carregar amostras de áudio com características diferentes
% [stream,Fs] = wavread('guitar1.wav', [1,2206]); %silêncio
% [stream,Fs] = wavread('guitar1.wav', [121000,123205]); %transição
% [stream,Fs] = wavread('guitar1.wav', [132300,134505]); %82,4 Hz
% [stream,Fs] = wavread('guitar1.wav', [308700,310905]); %110 Hz

% [stream,Fs] = wavread('guitar1.wav', [210000,212205]); %164,8 Hz
% [stream,Fs] = wavread('guitar1.wav', [275000,277205]); %164,8 Hz
[stream,Fs] = wavread('bass.wav', [132300,134505]); %164,8 Hz

% Subamostrar o sinal de 44,1 KHz para um valor próximo ao que será
% usado no PSoC
 stream = downsample(stream,11);
Fs = Fs/11; %44100/11 = 4009
T = 1/Fs;
L = 201;  %largura da janela: 50 ms
NFFT = 4096;

t = 0:T:(L-1)*T;

% Sinal artificial que pode ser usado para testar o algoritmo na falta
% de um sinal real de áudio
% x = 100;  % Frequência em Hz
% stream = 0.25*sin(2*pi*x*t) + 0.25*sin(2*pi*2*x*t) +
0.25*sin(2*pi*3*x*t) + 0.25*sin(2*pi*4*x*t);
% stream = stream';

% Sinal com ruído branco gaussiano
% stream = wgn(L,1,0);

% Gráfico do sinal pelo tempo
% figure;
% plot(t,stream);

% Espectrograma mostra a energia concentrada nos harmônicos
figure;
spectrogram(stream,[],[],[],Fs,'yaxis');

% Aplica janela de Hann
windowed = stream.*hann(L, 'periodic');

FFT = fft(windowed,NFFT);
% Descarta a metade redundante da FFT
FFT = FFT(1 : size(FFT,1) / 2);
FFT = abs(FFT);

% Gráfico da FFT até a frequência de 1,4KHz
f = Fs/2*linspace(0,1,NFFT/2);
S = floor(1400*Fs/NFFT);    % amostra que corresponde à frequência
1400Hz,
                            % máxima frequência de interesse
```

```
figure;
plot(f(1:S),FFT(1:S));

% Subamostra (comprime) o sinal
hps1 = downsample(FFT,1);
% figure;
% plot(hps1(1:500));
hps2 = downsample(FFT,2);
% figure;
% plot(hps2(1:500));
hps3 = downsample(FFT,3);

HPS0 = zeros(1, length(hps3));

% Multiplica os sinais de modo a ressaltar a fundamental
for j=1:length(hps3)
    HPS0(j) =   hps1(j) * hps2(j) * hps3(j);
end

% Gráfico do resultado do HPS
figure;
plot(HPS0);

% Descobrir em que ponto está a f0, se ela existir
[m,n]=findpeaks(HPS0, 'SORTSTR', 'descend');
M = n(1);   % Depois de ordenado a f0 estará no primeiro elemento

% Cálculo da energia total do sinal e também da energia nos arredores f0
E = sum(FFT.^2);    % Parseval
if (M >= 60)     % Se M < 60 a energia de certas frequências seria somada
duas vezes.
    Eh = sum(FFT(M-30:M+30).^2 + FFT(2*M-30:2*M+30).^2 + FFT(3*M-
30:3*M+30).^2 + FFT(4*M-30:4*M+30).^2 + FFT(5*M-30:5*M+30).^2 + FFT(6*M-
30:6*M+30).^2);
else
    Eh = 0;
end

% Se mais de 97% da energia do sinal estiver no entorno da f0 calculada e
% suas harmônicas então aceita
if (Eh/E > .99)
    f0 =  (M - 1) * (Fs/NFFT)    % Converte o ponto do HPS em frequência
em Hz
else
    f0 = -1
end
```

```
% Código para a análise um grande trecho de áudio, com inúmeras janelas
clear

% Carregar sinal de áudio
[stream,Fs] = wavread('bass.wav');
stream = downsample(stream,11);

% Subamostrar o sinal de 44,1 KHz para um valor próximo ao que será
% usado no PSoC
```

```
Fs = Fs/11; %44100/11 = 4009
T = 1/Fs;
L = floor(.05*Fs);  %largura da janela: 50 ms
W = floor(length(stream)/L);
NFFT = 4096;
a = 30;

figure;
t = 0:T:(length(stream)-1)*T;
plot(t, stream);

% Espectrograma mostra a energia concentrada nos harmônicos
figure;
spectrogram(stream,[],[],[],Fs,'yaxis');

% Vetor com o resultado do HPS a cada janela
Fv = zeros(1, W);

% Loop para calcular a f0 de cada janela
for i=1:W
    % Aplica janela de Hanning
    windowed = stream(((i-1)*(L)+1):((i)*(L))).*hann(L);

    FFT = fft(windowed,NFFT);
    % Descarta a metade redundante da FFT
    FFT = FFT(1 : size(FFT,1) / 2);
    FFT = abs(FFT);

    % Subamostra (comprime) o sinal
    hps1 = downsample(FFT,1);
    hps2 = downsample(FFT,2);
    hps3 = downsample(FFT,3);

    HPS0 = zeros(1, length(hps3));

    % Multiplica os sinais de modo a ressaltar a fundamental
    for j=1:length(hps3)
       HPS0(j) =   hps1(j) * hps2(j) * hps3(j);
    end

    % Descobrir em que ponto está a f0, se ela existir
    [m,n]=findpeaks(HPS0, 'SORTSTR', 'descend');
    M = n(1);   % Depois de ordenado a f0 estará no primeiro elemento

    % Cálculo da energia total do sinal e também da energia nos arredores
f0
    E = sum(FFT.^2);    %Parseval
    if (M >= 60 && M < 335)     % Se M < 60 a energia de certas
frequências seria somada duas vezes.
        Eh = sum(FFT(M-a:M+a).^2 + FFT(2*M-a:2*M+a).^2 + FFT(3*M-
a:3*M+a).^2 + FFT(4*M-a:4*M+a).^2 + FFT(5*M-a:5*M+a).^2 + FFT(6*M-
a:6*M+a).^2);
    else
        Eh = 0;
    end

    % Se mais de 98% da energia do sinal estiver no entorno da f0
calculada e
```

```
    % suas harmônicas então aceita
    if (Eh/E > .98)
        f0 =  (M - 1) * (Fs/NFFT);    % Converte o ponto do HPS em
frequência em Hz
    else
        f0 = -10;
    end

    Fv(i) = f0;
end

% Gráfico com o resultado do HPS a cada janela

figure;
stem(Fv, 'filled', 'LineWidth', 0.00005, 'LineStyle', ':', ...
        'MarkerSize', 2, 'Color', 'y', 'MarkerEdgeColor', 'r', ...
        'MarkerFaceColor', 'r');
```

APÊNDICE B – CÓDIGO EM LINGUAGEM C

```
/* ========================================
 * main.c
 * ========================================
*/
#include <device.h>
#include <afinador.h>

uint8 DMA_Chan;
volatile uint8 DMA_Done_Flag;
volatile uint8 Tuning_Done_Flag = 0;
arm_status status;
arm_cfft_radix2_instance_f32 S;

void main()
{
    int16 i = 0;
    int16 ADC_Buffer[WINDOW_LENGTH] = {0};
    float ADC_Buffer_Float[WINDOW_LENGTH] = {0};
    float Windowed[NFFT] = {0};
    float FFT_Abs[NFFT*2] = {0};
    float HPS_Vector[SIZE_HPS_VECTOR] = {0};
    int32 f0 = 0;
    float Last_Results[3] = {0};
    float LR_Mean = 0;
    float LR_Var = 0;

    // Inicialização do hardware
    Opamp_Start();
      ADC_Start();
    ADC_IRQ_Disable();
      ADC_StartConvert();
    DMA_Config(ADC_Buffer);
    CyDmaChEnable(DMA_Chan, 1);

    CYGlobalIntEnable;
    ISR_DMA_Done_Start();

    // Inicialização para a função FFT
    status = arm_cfft_radix2_init_f32(&S, NFFT, IFFT_FLAG,
DO_BIT_REVERSE);

    for(;;)
    {
        // Loop infinito para quando o processamento termina
        while(Tuning_Done_Flag);

        // Toda a vez que a tranferência do ADC para a memória estiver
        // completa, inicia o processamento
        if(DMA_Done_Flag)
        {
            // Desabilita a DMA para que seja feita a leitura do buffer
do ADC
            CyDmaChDisable(DMA_Chan);

            // Converter o vetor buffer do ADC para ponto flutuante
            arm_q15_to_float(ADC_Buffer, ADC_Buffer_Float,
WINDOW_LENGTH);
```

```
            // Habilita novamente a transferência do ADC para o buffer,
de modo a ter sempre a
            // amostra mais recente possível lá
            CyDmaChEnable(DMA_Chan, 1);

            // Cálculo da FFT
            Control_Reg_Time_Write(1);
            FFT(ADC_Buffer_Float, Windowed, FFT_Abs);

            // Realização do algoritmo Harmonic Product Spectrum
            HPS(HPS_Vector, FFT_Abs, &f0);
            Control_Reg_Time_Write(0);

            // Atualizar o vetor com os últimos resultados do HPS
            Last_Results[2] = Last_Results[1];
            Last_Results[1] = Last_Results[0];
            Last_Results[0] = (float)f0;

            // Média e variância dos últimos três resultados
            arm_mean_f32(Last_Results, 3u, &LR_Mean);
            arm_var_f32(Last_Results, 3u, &LR_Var);

            // Se a frequência medida estiver próxima do objetivo,
termina o processamento
            if(LR_Mean >= (TUNING_FREQ - 1) && LR_Mean <= (TUNING_FREQ +
1))
            {
                Tuning_Done_Flag = 1;
                CyDmaChDisable(DMA_Chan);
            }
            else
                // Se os últimos três resultados forem válidos e não
muito distantes entre
                // si, começa a afinar a corda
                if(Last_Results[2] > 0 && Last_Results[1] > 0 &&
Last_Results[0] > 0 && LR_Var <= 2)
                    Tune((int)LR_Mean);     // Função que aciona os
motores. Não implementada

            DMA_Done_Flag = 0;
        }
    }
}

void DMA_Config (int16* ADC_Buffer)
{
    uint8 DMA_TD[1];

    #define DMA_BYTES_PER_BURST 2
    #define DMA_REQUEST_PER_BURST 1
    #define DMA_SRC_BASE (CYDEV_PERIPH_BASE)
    #define DMA_DST_BASE (CYDEV_SRAM_BASE)
    DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST,
DMA_REQUEST_PER_BURST,
        HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));
    DMA_TD[0] = CyDmaTdAllocate();
    CyDmaTdSetConfiguration(DMA_TD[0], 2 * WINDOW_LENGTH, DMA_TD[0],
DMA__TD_TERMOUT_EN | TD_INC_DST_ADR);
```

```
    CyDmaTdSetAddress(DMA_TD[0], LO16((uint32)ADC_DEC_SAMP_PTR),
LO16((uint32)ADC_Buffer));
    CyDmaChSetInitialTd(DMA_Chan, DMA_TD[0]);
}

void FFT(
    float* ADC_Buffer_Float,
    float* Windowed,
    float* FFT_Abs)
{
    int i;

    // Aplicar a janela ao sinal que veio do ADC
    arm_mult_f32(ADC_Buffer_Float, Hann, Windowed, WINDOW_LENGTH);
    // Deixar o sinal no formato exigido para a realização da FFT: real,
comp, real, comp,...
    for(i = 0; i < NFFT; i++)
    {
        FFT_Abs[i*2] = Windowed[i];
        FFT_Abs[i*2 + 1] = 0.0f;
    }
    // FFT
    arm_cfft_radix2_f32(&S, FFT_Abs);
    // Módulo da FFT
    arm_cmplx_mag_f32(FFT_Abs, FFT_Abs, NFFT);
}

void HPS(
    float* HPS_Vector,
    float* FFT_Abs,
    int32* f0)
{
    int16 i;
    float HPS_Max_Value = 0;
    uint32 HPS_Max_Index = 0;
    float E = 0;
    float E0 = 0;
    float Eh = 0;

    // Geração do vetor HPS
    for(i = 0; i < SIZE_HPS_VECTOR; i++)
        HPS_Vector[i] = FFT_Abs[i];// * FFT_Abs[i*2] * FFT_Abs[i*3];
    // Descobrir o ponto máximo do HPS, onde possivelmente estará a f0
    arm_max_f32(HPS_Vector, SIZE_HPS_VECTOR, &HPS_Max_Value,
&HPS_Max_Index);
    // Cálculo da energia total do sinal e também da energia nos
arredores da possível f0
    // Parseval
    arm_power_f32(FFT_Abs, NFFT/2, &E);
    if(60 < HPS_Max_Index && HPS_Max_Index < 335)
        for(i = 1, Eh = 0.0f; i <= 6; i++)
        {
            arm_power_f32(&FFT_Abs[HPS_Max_Index*i-30], 61, &E0);
//Energia nos arredores da f0*i
            Eh += E0;
        }
    // Se mais de 98% da energia do sinal estiver no entorno da f0
calculada e
    // de suas harmônicas então aceita
    if(Eh/E > 0.98f)
```

```
        *f0 = HPS_Max_Index;
    else
        *f0 = -1;
}

void Tune(int freq)
{
    int i;
    // Se o frequência está abaixo do desejado, gira para um lado
    if (freq < TUNING_FREQ)
        for (i = (TUNING_FREQ - freq)*ONE_HZ_STEPS; i > 0; i--)
        {
            Control_Reg_Stepper_Write(Step[i%4]);
            CyDelay(15);
        }
    // Senão gira para o outro
    else
        for (i = 0; i < (freq - TUNING_FREQ)*ONE_HZ_STEPS; i++)
        {
            Control_Reg_Stepper_Write(Step[i%4]);
            CyDelay(15);
        }
}

/* [] END OF FILE */
```

```
/* ========================================
 * afinador.h
 * ========================================
*/
#include <arm_math.h>

#define SAMP_RATE       4096
#define WINDOW_LENGTH   205     // 4096*50ms
#define NFFT            4096
#define SIZE_HPS_VECTOR 683     // NFFT/6
#define IFFT_FLAG       0
#define DO_BIT_REVERSE  1
#define TUNING_FREQ     98      // Frequência para a qual a corda deve
ser afinada
#define ONE_HZ_STEPS    30      // Número de passos do motor para alterar
a frequência da corda em 1 Hz

void DMA_Config (int16*);
void FFT(float*, float*, float*);
void HPS(float*, float*, int32*);
void Tune(int);

const int8 Step[4] = {8, 2, 4, 1}; // Vetor com a sequência que ativa os
passos do motor

const float Hann[WINDOW_LENGTH] =
{
0, 0.000234832404453433, 0.000939109032781049, 0.00211216833708677,
0.00375290842802251, 0.00585978810982574, 0.00843082832801023,
0.0114636140283504, 0.0149552964254134, 0.0189025956785078,
0.0233018039725347, 0.0281487890008486, 0.0334389978468553,
```

```
0.0391674612607014, 0.0453287983270376, 0.0519172215194722,
0.0589265421369659, 0.0663501761170618, 0.0741811502204907,
0.0824121085813410, 0.0910353196166412, 0.100042683288865,
0.109425738714537, 0.119175672111791, 0.129283325079413,
0.139739203199606, 0.150533484956371, 0.161656030961151,
0.173096393477057, 0.184843826232741, 0.196887294516681,
0.209215485542417,
0.221816819074985, 0.234679458308573, 0.247791320985187,
0.261140090743869, 0.274713228689823, 0.288497985172571,
0.302481411762073,
0.316650373411576, 0.330991560795749, 0.345491502812526,
0.360136579236916, 0.374913033514877, 0.389806985685261,
0.404804445417666,
0.419891325153975, 0.435053453341210, 0.450276587743297,
0.465546428819215, 0.480848633154982, 0.496168826936849,
0.511492619453045,
0.526805616611403, 0.542093434460154, 0.557341712699200,
0.572536128169173, 0.587662408305594, 0.602706344545525,
0.617653805674087,
0.632490751098331, 0.647203244035982, 0.661777464606671,
0.676199722813361, 0.690456471401761, 0.704534318585670,
0.718420040626272,
0.732100594253589, 0.745563128918412, 0.758794998863198,
0.771783775000611, 0.784517256588523, 0.796983482690544,
0.809170743411272,
0.821067590895751, 0.832662850082783, 0.843945629201985,
0.854905330004760, 0.865531657719542, 0.875814630721979,
0.885744589910972,
0.895312207781749, 0.904508497187474, 0.913324819781133,
0.921752894129794, 0.929784803493601, 0.937413003262199,
0.944630328041617,
0.951429998384927, 0.957805627160384, 0.963751225551047,
0.969261208680253, 0.974330400857655, 0.978954040440902,
0.983127784308390,
0.986847711938889, 0.990110329094200, 0.992912571101395,
0.995251805731562, 0.997125835672330, 0.998532900591870,
0.999471678792430,
0.999941288451841, 0.999941288451841, 0.999471678792430,
0.998532900591870, 0.997125835672330, 0.995251805731562,
0.992912571101395,
0.990110329094200, 0.986847711938889, 0.983127784308390,
0.978954040440902, 0.974330400857655, 0.969261208680253,
0.963751225551047,
0.957805627160384, 0.951429998384927, 0.944630328041617,
0.937413003262199, 0.929784803493601, 0.921752894129794,
0.913324819781133,
0.904508497187474, 0.895312207781749, 0.885744589910972,
0.875814630721979, 0.865531657719542, 0.854905330004760,
0.843945629201985,
0.832662850082783, 0.821067590895751, 0.809170743411272,
0.796983482690544, 0.784517256588523, 0.771783775000611,
0.758794998863198,
0.745563128918412, 0.732100594253589, 0.718420040626272,
0.704534318585670, 0.690456471401761, 0.676199722813361,
0.661777464606671,
0.647203244035982, 0.632490751098331, 0.617653805674087,
0.602706344545525, 0.587662408305594, 0.572536128169173,
0.557341712699200,
```

```
0.542093434460154, 0.526805616611403, 0.511492619453045,
0.496168826936849, 0.480848633154982, 0.465546428819215,
0.450276587743297,
0.435053453341210, 0.419891325153975, 0.404804445417666,
0.389806985685261, 0.374913033514877, 0.360136579236916,
0.345491502812526,
0.330991560795749, 0.316650373411576, 0.302481411762073,
0.288497985172571, 0.274713228689823, 0.261140090743869,
0.247791320985187,
0.234679458308573, 0.221816819074985, 0.209215485542417,
0.196887294516681, 0.184843826232741, 0.173096393477057,
0.161656030961151,
0.150533484956371, 0.139739203199606, 0.129283325079413,
0.119175672111791, 0.109425738714537, 0.100042683288865,
0.0910353196166412,
0.0824121085813410, 0.0741811502204907, 0.0663501761170618,
0.0589265421369659, 0.0519172215194722, 0.0453287983270376,
0.0391674612607014, 0.0334389978468553, 0.0281487890008486,
0.0233018039725347, 0.0189025956785078, 0.0149552964254134,
0.0114636140283504, 0.00843082832801023, 0.00585978810982574,
0.00375290842802251, 0.00211216833708677, 0.000939109032781049,
0.000234832404453433
};

//[] END OF FILE
```