

*Instituto Superior de Engenharia de Lisboa*

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

O Maestro

Verão 2010/2011

Ana Correia e Diogo Cardoso

Relatório de projecto realizado no âmbito de Projecto e Seminário no semestre de Verão 2010/2011 do curso de Licenciatura em Engenharia Informática e de Computadores sob orientação de Pedro Sampaio e Artur Ferreira



*Instituto Superior de Engenharia de Lisboa*

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

O Maestro

Projecto e Seminário 2010/2011

Relatório do Projecto

**Setembro 2011**

**Autores:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ana Sofia Duarte Correia , Nº 31831 (a31831@alunos.isel.pt)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Diogo Sérgio Esteves Cardoso, Nº 32466 (a32466@alunos.isel.pt)

**Orientadores:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Artur Jorge Ferreira (arturj@isel.pt)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Pedro Miguel Fernandes Sampaio (psampaio@cc.isel.pt)

# Resumo

Hoje em dia, existem inúmeros dispositivos, de tamanhos variados, nos quais podemos ouvir música. Essa disseminação da música leva a que as pessoas tenham vontade de aprender a produzir música , reduzindo-se normalmente à aprendizagem de um instrumento musical. A música como qualquer outra forma de arte, não é fácil de aprender, necessitando normalmente do acompanhamento de um professor, especialmente na fase inicial da aprendizagem. Seria então interessante incorporar um sistema dedicado que detecte as notas que estão a ser tocadas e mostra-las em forma de pauta. Assim a componente humana, que neste caso o professor, nem sempre era necessário para ajudar no treino da aprendizagem do instrumento que se pretende tocar. Este projecto consiste na captação e processamento do sinal produzido por um instrumento de forma a que seja possível mostrá-lo sob a forma de uma pauta. Após um estudo inicial, o algoritmo escolhido para detecção das frequências das notas musicais foi o de *Goertzel* pelas suas características na eficiência computacional e pela sua pouca complexidade aritmética.

A arquitectura do sistema dedicado é um ARM7 da placa de desenvolvimento YAAB2294 e para capturar o som das notas musicais foi utilizado o ADC a 10 bits interno da placa.

O resultado final do projecto atinge todos os objectivos propostos, de modo que a infra-estrutura realizada no âmbito deste projecto seja portável para qualquer arquitectura.

**Palavras Chave:** Sistema dedicado, *Analog to Digital Converter* (*ADC*), Algoritmo de *Goertzel*, notas música is, *Advanced Risc Machine* (*ARM7TDMI*).

# Agradecimentos

*Para os nossos pais e irmãos*

*Para os nossos amigos*

*pelo seu constante apoio.*

Conteúdo

[Resumo I](#_Toc303723097)

[Agradecimentos III](#_Toc303723098)

[Lista de Figuras VIII](#_Toc303723099)

[Lista de Tabelas IX](#_Toc303723100)

[1. Introdução 1](#_Toc303723101)

[1.1 Motivação 1](#_Toc303723102)

[1.2 Objectivos e Descrição 2](#_Toc303723103)

[1.3 Análise de Requisitos 3](#_Toc303723104)

[1.4 Especificações da placa desenvolvimento YAAB2294 5](#_Toc303723105)

[1.5 Soluções existentes 5](#_Toc303723106)

[1.6 Organização do documento 6](#_Toc303723107)

[2. Formulação do Problema 7](#_Toc303723108)

[2.1 Conceitos musicais 7](#_Toc303723109)

[2.1.1 Notas musicais 8](#_Toc303723110)

[2.1.2 Tempo musical 10](#_Toc303723111)

[2.2 Notação Musical 10](#_Toc303723112)

[2.2.1 Pauta 10](#_Toc303723113)

[2.2.2 Claves musicais 11](#_Toc303723114)

[2.2.3 Figuras musicais 12](#_Toc303723115)

[2.2.4 Alterações musicais 12](#_Toc303723116)

[2.2.5 Compasso 13](#_Toc303723117)

[2.3 Instrumentos 14](#_Toc303723118)

[2.3.1 Instrumento de Teste 14](#_Toc303723119)

[3. Detecção de frequência 17](#_Toc303723120)

[3.1 Discrete Fourier Transform(DFT) 17](#_Toc303723121)

[3.2 Fast Fourier Trasnform(FFT) 18](#_Toc303723122)

[3.3 O algoritmo de Goertzel 18](#_Toc303723123)

[3.3.1 Descrição 19](#_Toc303723124)

[3.3.2 Características 21](#_Toc303723125)

[3.4 Escolha do Algoritmo 22](#_Toc303723126)

[3.5 Implementação do algoritmo de *Goertzel* 23](#_Toc303723127)

[3.6 Tratamento da Resolução do Goertzel 24](#_Toc303723128)

[3.7 Filtragem do sinal 26](#_Toc303723129)

[3.7.1 Filtros *FIR* 27](#_Toc303723130)

[4. Implementação 31](#_Toc303723131)

[4.1 Linguagem de programação 31](#_Toc303723132)

[4.2 Algoritmo de Goertzel 32](#_Toc303723133)

[4.3. GoertzelController 32](#_Toc303723134)

[4.3.1 Configuração e Parametrização 33](#_Toc303723135)

[4.3.2 Samples Manager 34](#_Toc303723136)

[4.3.3 Goertzel Filters 34](#_Toc303723137)

[4.3.4 Results Controller 35](#_Toc303723138)

[4.3.5 Funcionamento e Características 37](#_Toc303723139)

[4.4 Calculo da duração das notas 39](#_Toc303723140)

[4.4.1 Funcionamento 40](#_Toc303723141)

[4.4.2 Limitação do módulo *GoertzelTimeController* 41](#_Toc303723142)

[4.5 Arquitectura PC (Windows) 43](#_Toc303723143)

[4.6 Arquitectura ARM7 43](#_Toc303723144)

[4.6.1 Aquisição de sinal 43](#_Toc303723145)

[4.6.2 Sistema Operativo 44](#_Toc303723146)

[4.6.3 *Port* da infra-estrutura para *ARM* 46](#_Toc303723147)

[4.6.4 Resumo de sistema 47](#_Toc303723148)

[5 Testes e Resultados 48](#_Toc303723149)

[5.1 Testes ao algoritmo de Goertzel 48](#_Toc303723150)

[5.1.1 Preparação 48](#_Toc303723151)

[5.1.2 Teste Funcional 49](#_Toc303723152)

[5.1.3 Teste Temporal 50](#_Toc303723153)

[5.2 Testes à infra-estrutura *GoerzelController* 51](#_Toc303723154)

[5.2.1 Preparação 51](#_Toc303723155)

[5.2.2 Teste teórico 51](#_Toc303723156)

[5.2.3 Teste temporal 52](#_Toc303723157)

[5.3 Testes temporais aos sistemas operativos 54](#_Toc303723158)

[5.4 Conclusão 55](#_Toc303723159)

[6. Conclusão 57](#_Toc303723160)

[6.1 Dificuldades e desafios 57](#_Toc303723161)

[6.2 Trabalho futuro 58](#_Toc303723162)

[Referências 60](#_Toc303723163)

Lista de Figuras

[Figura 1 - Funcionamento do Maestro. 2](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723164)

[Figura 2- Arquitectura de *Software* do Projecto 3](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723165)

[Figura 3 - Representação de uma oitava na escrita Europeia 9](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723166)

[Figura 4 - Representação das oitavas 9](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723167)

[Figura 5 - Metrónomo Analógico 10](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723168)

[Figura 6- Exemplo de uma pauta com clave de Sol. 11](#_Toc303723169)

[Figura 7 - Aumento do tempo de uma simínima. 13](#_Toc303723170)

[Figura 8- Representação de um compasso. 13](#_Toc303723171)

[Figura 9 - Piano 15](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723172)

[Figura 10 - Diagrama de blocos de um filtro de Goertzel. 19](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723173)

[Figura 11 - Máquina de Estados de um filtro de *Goertzel*. O estado "Calcular energia relativa" refere-se à equação (9) 23](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723174)

[Figura 12 - Diagrama de blocos de um filtro *FIR* 27](#_Toc303723175)

[Figura 12 - Resposta em frequência de um filtro passa-banda entre 2217 Hz e 4186 Hz. (a) - Filtro normal- (b) - Filtro com normalização de ganho. (c) - Filtro com janela de Hamming. (d) - Filtro com janela de Hamming e normalização de ganho. 29](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723176)

[Figura 13 - Diagrama de blocos do processamento de sinal. 32](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723177)

[Figura 14- Diagrama de blocos do módulo Goertzel Filters. 35](#_Toc303723178)

[Figura 15 - Flowchart do funcionamento do GoertzelController. 37](#_Toc303723179)

[Figura 16 - Flowchart to tratamento de ruído da infra-estrutura. 39](file:///D:\FAC\LEIC\PS\working-copy\docs\relfinal\rfv2.docx#_Toc303723180)

# Lista de Tabelas

[Tabela 1- Notas musicais e as suas frequências. 9](#_Toc303723181)

[Tabela 2- Claves musicais: símbolo e significado. 11](#_Toc303723182)

[Tabela 3- Representação dos símbolos dos tempos que uma nota musical poderá ter. 12](#_Toc303723183)

[Tabela 4 - Alterações ou acidentes musicais. 13](#_Toc303723184)

[Tabela 5 - Gama de frequências produzidas por alguns instrumentos musicais. 14](#_Toc303723185)

[Tabela 6 - Gama de frequências (D.C.A - Diferença Com a Anterior). 16](#_Toc303723186)

[Tabela 7 - Algumas frequências da Tabela 6. 25](#_Toc303723187)

[Tabela 8 - Valores de N e das frequências de amostragem para as frequências do piano. 26](#_Toc303723188)

[Tabela 9 - Exemplos de frequências com o mesmo coeficiente. 27](#_Toc303723189)

[Tabela 10 - Precisão temporal das gamas a detectar com um Fs = 8800. 41](#_Toc303723190)

[Tabela 11 - Precisão temporal das gamas a detectar com um Fs = 8800 e tempo absoluto de uma semibreve igual a 2 segundos (96 blocos). 42](#_Toc303723191)

[Tabela 12 - Memória utilizada pelo O Maestro. 47](#_Toc303723192)

[Tabela 13 - Resultados do teste teórico ao algoritmo de *Goertzel* com sinais compostos por múltiplas sinusoides 49](#_Toc303723193)

[Tabela 14 - Resultado do calculo do tempo de processamento do algoritmo de *Goertzel*. 50](#_Toc303723194)

[Tabela 15 - Resultados do teste teórico à infra-estrutura com múltiplas frequências. 52](#_Toc303723195)

[Tabela 16 - Tempos relativos de processamento da infra-estrutura. 53](#_Toc303723196)

[Tabela 17 - Resultado de testes temporais com várias frequências no sinal. 54](#_Toc303723197)

[Tabela 18 - Tempos de comutação. 54](#_Toc303723198)

# 1. Introdução

O presente documento constitui o relatório do projecto - O Maestro - elaborado por Ana Correia e Diogo Cardoso, no âmbito de Projecto e Seminário, na Licenciatura em Engenharia Informática e de Computadores, do Instituto Superior de Engenharia de Lisboa, no semestre de Verão 2010/2011.

## 1.1 Motivação

Hoje em dia, a música faz parte do quotidiano das pessoas de todas as classes sociais. Esta disseminação da música faz com que praticamente todas as pessoas tenham consigo um dispositivo de reprodução de música, o exemplo mais marcante é o dos leitores de MPEG1 – Layer3, vulgarmente designados por MP3, que existem praticamente em todos os dispositivos móveis, nomeadamente telemóveis. Este contacto diário com a música faz com que muitas pessoas iniciem um estudo sobre o mundo da música, levando-as a aprender a tocar determinado instrumento. Apesar de existirem diversos meios de estudo e aprendizagem, a interacção humana no âmbito do processo de aprendizagem é algo fulcral para os iniciados, uma vez que, estes simplesmente ainda não têm conhecimento suficiente para saber se o que estão a tocar está correcto ou não. Como tal, necessitam de interacção no processo de aprendizagem musical.

Seria então interessante que, para além do professor humano, existisse outra entidade neste contexto de aprendizagem musical; assim além dos alunos e professores propomos a criação de *O Maestro*. O sistema dedicado desenvolvido neste projecto, detecta as notas estão a ser tocadas, permitindo assim aos iniciados comparar as notas tocadas com o que realmente deveria ser tocado. Este sistema produzirá uma pauta musical a partir do som recolhido do instrumento que esta a ser tocado.

## 1.2 Objectivos e Descrição

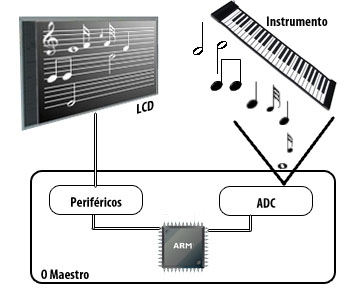
 A ilustra o diagrama de blocos dos elementos do projecto e a interacção entre os mesmos.

Figura - Funcionamento do Maestro.

*O Maestro,* é um sistema dedicado sobre a arquitectura *Advance Risk Machine* (*ARM7TDMI)* [[1](#ARM11)] que trata de obter notas música produzidas por determinado instrumento e apresentá-las sob a forma de uma pauta música. Para a recolha de amostras será utilizado o *Analog to Digital Converter* (*ADC)* associado ao microcontrolador. Para o *input* e *output* irá ser usado um *Liquid Crystal Display* (*LCD*) gráfico *touch screen* como ilustra a .

A componente de software deste projecto está dividida em três camadas, tal como se apresenta na :

1. *Hardware*, responsável por interagir directamente com os periféricos internos e externos do microcontrolador.
2. Abstracção ao *hardware*, é a ponte entre a camada aplicacional e o *hardware*.
3. Aplicacional, onde é efectuado o controlo do *input* e *output* do utilizador, gestão da aplicação e ainda é a camada onde o algoritmo de *Goertzel* será implementado.

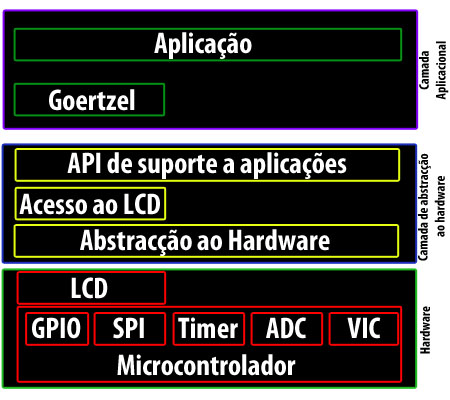
****

Figura - Arquitectura de *Software* do Projecto

O projecto foi desenhado de forma a que o código produzido seja portável entre arquitecturas, ou seja, toda a camada aplicacional foi implementada de forma a que não depende-se de qualquer característica exclusiva de uma determinada arquitectura. Em relação a dependências de hardware, a implementação do projecto assume apenas que existe alguma módulo de aquisição de sinal.

## 1.3 Análise de Requisitos

Após a análise dos requisitos do projecto, constatou-se que os problemas mais relevantes são a recolha e processamento das amostras de som. As frequências que se pretende captar e processar estão na banda de 27 Hz a 4186 Hz. Assim, é necessário, respeitando o teorema de *Nyquist* [[2](#1)]*,* no mínimo utilizar uma frequência de amostragem superior a 8372 Hz. Para aquisição do som foi utilizado o *ADC* interno do microcontrolador que funciona com 10 *bits* por amostra na gama dinâmica de amplitude de 0 a 3 V, com frequência de amostragem até 450 kHz, sendo adequado para a banda de frequência que se pretende processar.

Na realização do projecto utilizam-se os seguintes recursos:

* Microcontrolador baseado na arquitectura ARM7TDMI - LPC2294 da NXP [[3](#Kei11)].
* *LCD* R*G*B gráfico (320x240 *pixels*) com *touch screen*.
* Ferramentas open-source da GNU para desenvolvimento sobre a arquitectura ARM7TDMI.
* O periférico *ADC* do Microcontrolador LPC2294.

1.4 Soluções existentes

Existem várias soluções com objectivos semelhantes aos deste projecto. As soluções mais encontradas são aplicações de afinação de instrumentos produzidas normalmente para sistemas operativos móveis. O funcionamento destas reduz-se a pedir ao utilizador que toque uma nota música l, normalmente no centro da escala (*e.g.* 440Hz LÁ3), e posteriormente o som emitido pelo instrumento é avaliado e comparado com a nota pedida ao utilizador.

Existe ainda uma aplicação *Note Detector* realizada em *C#* e *C++* que tem o mesmo comportamento que o Maestro, esta trata de ouvir o som da placa gráfica e de identificar a nota que foi tocada através do algoritmo da *FFT*.

Apesar das diferentes aplicações com premissas semelhantes às deste projecto nenhuma cumpre todas os objectivos do Maestro, as aplicações de afinação apesar de funcionar praticamente em qualquer arquitectura estão normalmente comprometidas a um número fixo de notas que conseguem detectar, ou sabem que nota vão detectar no momento do processamento de sinal, a aplicação *Note Detector* está comprometida com a arquitectura *PC* e sistema operativo *Windows*, além disso usa um algoritmo com elevada complexidade aritmética para processar o sinal como irá ser descrito no capitulo 3*.*

## 

## 1.4 Especificações da placa desenvolvimento YAAB2294

A placa de desenvolvimento YAAB tem um microcontrolador LPC2294 que contem as seguintes características:

* Velocidade de CPU 60 MHz.
* 8 Mbytes de memória RAM.
* 8 Mbytes de memória ROM.
* Ethernet ENC28J60.
* ADC a 10 bits.
* *LCD* R*G*B gráfico (320x240 *pixels*) com *touch screen*
* *SD Card*

## 1.5 Soluções existentes

Existem várias soluções com objectivos semelhantes aos deste projecto. As soluções mais comuns são aplicações de afinação de instrumentos, produzidas normalmente para sistemas operativos móveis. O funcionamento destas reduz-se a pedir ao utilizador que toque uma nota musical, normalmente no centro da escala (*e.g.* 440Hz LÁ3), e posteriormente o som emitido pelo instrumento é avaliado e comparado com a nota pedida ao utilizador.

Existe ainda uma aplicação *Note Detector* realizada em *C#* e *C++* que tem o mesmo comportamento que a solução que se apresenta (*O Maestro*). Esta aplicação obtêm o som a partir da placa de som do PC e utiliza uma biblioteca criada em *C++* que implementa o algoritmo *Fast Fourier Transform*(*FFT*) para identificar as frequências que se encontram no sinal obtido.

Da aplicações semelhantes à proposta neste projecto nenhuma cumpre todas os objectivos do Maestro; as aplicações de afinação apesar de funcionarem numa grande gama de arquitecturas, mas estão normalmente comprometidas a um número fixo de notas que conseguem detectar, ou apenas detectam uma frequência predefinida para indicarem ao utilizador se o instrumento se encontra afinado, a aplicação *Note Detector* está comprometida com a arquitectura *PC* e sistema operativo *Windows*. Além disso usa o algoritmo utilizado para detectar as frequências tem elevada complexidade aritmética como irá ser descrito em na capitulo 3. Detecção de frequência.

## 1.6 Organização do documento

O restante documento está dividido em 6 capítulos.

No capitulo 2 consta a formulação do problema, onde é explicado os conceitos básicos música.

O capitulo 3 descreve os algoritmos de detecção de frequências estudados , as motivações para a escolha deste algoritmo Goertzel (2 )para o projecto e descrição detalhada do Goertzel.

O capitulo 4 contém toda a descrição da implementação do projecto, nas diferentes arquitecturas.

No capitulo 5 são apresentados os resultados dos testes efectuados para validar as implementações e opções tomadas.

Por fim, o capitulo 6 contém uma análise crítica sobre as opções tomadas, dificuldades e desafios encontrados bem como perspectivas de trabalho futuro sobre o projecto.

## 2. Formulação do Problema

No âmbito deste projecto é desenvolver uma infra-estrutura portável para qualquer arquitectura em que o seu objectivo é identificar notas musicais e representá-las na forma de uma pauta musical. O principal desafio foi encontrar um algoritmo de detecção de frequências, em que a sua complexidade tanto computacional como aritmética, seja portável para qualquer sistema. O sistema inicial que se deseja aplicar esta infra-estrutura é um sistema embebido que tem algumas limitações, tais como, velocidade do *CPU*, capacidade de memoria e entre outras. Deste modo a escolha do algoritmo para detecção de frequências teve-se que ter em conta os seguintes requisitos:

* Estar preparado para funcionar com arquitecturas que não tenham suporte para *floating* *point* (*FPU*).
* Ter latência baixa de processamento, de forma a apresentar resultados ao utilizador em tempo útil .
* Realizar simultaneamente a detecção de varias frequências.

Existem vários algoritmos de detecção de frequências, mas com base nos requisitos apresentados anteriormente e também com base no facto que deve ser portável, foram considerados dois algoritmos: *Goertzel* na versão optimizada e *Fast Fourier Transform*(*FFT*) na versão *Cooley-Turkey*.

Como o objectivo deste projecto é detectar notas musicais de um sinal de áudio e representar essas notas numa pauta musical, foi necessário elaborar um estudo nos conceitos fundamentais da música, tais como, notação musical, tempos musicais e notas musicais.

## 2.1 Conceitos musicais

A música é uma arte constituída por combinações de sons e silêncio seguindo uma composição ao longo do tempo. O som é a vibração de um corpo, constituído por 4 elementos: intensidade, altura, timbre e duração. A intensidade é a característica que identifica a força do som, isto é, quanto maior o tamanho da vibração produzida, mais forte fica o som. A altura é a velocidade de vibração do som. O timbre é a característica que identifica a personalidade deste. A duração é o intervalo de tempo que o som é produzido.

O som é composto uma ou várias notas musicais.

### 2.1.1 Notas musicais

Uma nota musical [[5](#And11)] é caracterizada por uma frequência(altura) e a sua duração, isto é, a nota é uma sinusóide que se propaga no ar com uma determinada frequência definida em Hertz, que se encontra no intervalo de 16 Hz a 8000 Hz e se repete durante um período de tempo.

A frequência da nota define-a como aguda, grave ou média. Por exemplo, caso a frequência se encontre no intervalo de 20 Hz a 100 Hz, a nota soa de forma grave, caso seja igual ou superior a 400 Hz, a nota soa de forma aguda. Terá uma frequência média no intervalo de 100 a 400 Hz.

Uma escala musical é representada pelas 7 notas musicais. Existem duas escritas diferentes para representar uma escala, a primeira é usada na Europa e as notas são representadas por Dó, Ré, Mi, Fá, Sol, Lá e Si e a segunda é usada nos países de Língua Inglesa, em que as notas são representadas por C, D, E, F, G, A e B.

Existem ainda outras notas que são alterações na frequência das 7 notas musicais apresentadas anteriormente, que se designa por acidentes musicais. Estas são representadas pelo nome da nota que irá ser alterada, mais o símbolo '*#*', que adiciona meio tom na altura ou o símbolo '*b*', que diminui meio tom na altura da nota. Assim, existem ao todo 12 notas musicais: Dó, Dó# ou Réb, Ré, Ré# ou Mib, Mi, Fá, Fá# ou Solb, Sol, Sol# ou Láb, Lá, Lá# ou Sib, Si. Como poderá ser observado não existe as notas Mi# , Fáb , Si# e Dób porque a frequência que estas notas iram conter, seria igual a outras notas que já existem, por exemplo, Mi# teria a mesma frequência que Fá. A Figura 3 mostra a representação desta 12 notas no piano.

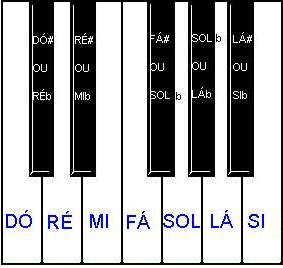


Figura - Representação de uma oitava na escrita Europeia

Ao conjunto destas 12 notas dá-se o nome de oitava. Existem ao todo 9 oitavas. A oitava da esquerda para a direita vai tendo o som mais agudo. A Figura 4 ilustra esse processo.



Figura - Representação das oitavas

A   
Tabela 1 demonstra a frequência de cada nota musical e a oitava a que essa se encontra.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Oitavas** | **C** | **C#** | **D** | **Eb** | **E** | **F** | **F#** | **G** | **G#** | **A** | **Bb** | **B** |
| **0** | 16,35 | 17,32 | 18,35 | 19,45 | 20,60 | 21,83 | 23,12 | 24,50 | 25,96 | 27,50 | 29,14 | 30,87 |
| **1** | 32,70 | 34,65 | 36,71 | 38,89 | 41,20 | 43,65 | 46,25 | 49 | 51,91 | 55 | 58,27 | 61,74 |
| **2** | 65,41 | 69,30 | 73,42 | 77,78 | 82,41 | 87,31 | 92,50 | 98,00 | 103,8 | 110 | 116,5 | 123,5 |
| **3** | 130,8 | 138,6 | 146,8 | 155,6 | 164,8 | 174,6 | 185 | 196 | 207,7 | 220 | 233,1 | 246,9 |
| **4** | 261,6 | 277,2 | 293,7 | 311,1 | 329,6 | 349,2 | 370 | 392 | 415,3 | 440 | 466,2 | 493,9 |
| **5** | 523,3 | 554,4 | 587,3 | 622,3 | 659,3 | 698,5 | 740,0 | 784 | 830,6 | 880 | 932,3 | 987,8 |
| **6** | 1047 | 1109 | 1175 | 1245 | 1319 | 1397 | 1480 | 1568 | 1661 | 1760 | 1865 | 1976 |
| **7** | 2093 | 2217 | 2349 | 2489 | 2637 | 2794 | 2960 | 3136 | 3322 | 3520 | 3729 | 3951 |
| **8** | 4186 | 4435 | 4699 | 4978 | 5274 | 5588 | 5920 | 6272 | 6645 | 7040 | 7459 | 7902 |

Tabela - Notas musicais e as suas frequências.

A relação entre a frequência de uma nota com a frequência da próxima nota em cada oitava é dada por 21/12. Por exemplo, a frequência da nota a seguir à nota Lá (440Hz) é:

PróximaNotaDoLá = 440 \* 21/12 ~ 446 Hz = Lá#

### 2.1.2 Tempo musical

 O tempo musical é o intervalo entre um som de uma nota da sua próxima nota ou da sua ausência (silêncio). Para marcar o tempo musical é utilizado o Metrónomo que se encontra na Figura 5. Este aparelho determina a velocidade da música, ou seja, é definido o número de batidas que cada representação de tempo () deve ser tocada, assim deste modo todos os instrumentos tocam no tempo correcto. A unidade utilizada para medida de tempo é BPM (Beat Per Minute) que marca o número de vezes que o pêndulo do Metrónomo oscilou.

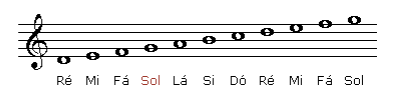
Figura - Metrónomo Analógico

## 2.2 Notação Musical

A notação musical é um sistema de escrita utilizado para representar uma peça musical, através de símbolos gráficos. Com este sistema é representado as notas musicais, os tempos de cada nota, o compasso da música e outros detalhes fundamentais, para que o músico tenha toda a informação necessária para tocar a peça musical.

### 2.2.1 Pauta

A pauta ou pentagrama é utilizada para escrever peças musicais. Esta é constituída por um conjunto de 4 espaços delimitados por 5 linhas equidistantes cuja função é a identificação das notas (sons). A cada espaço ou linha corresponderá apenas a uma nota. Além destas linhas principais existem as linhas suplementares que só são utilizadas se existirem notas cuja localização seja fora das 5 linhas principais(notas mais agudas ou mais graves). Essas linhas não são totalmente desenhadas, apenas o suficiente para se perceber qual a localização exacta da nota. O conjunto de linhas e espaços, por si só não tem qualquer significado, é necessário ter uma clave para indicar a localização da nota.

  
Figura - Exemplo de uma pauta com clave de Sol.

### 2.2.2 Claves musicais

As claves determinam o posicionamento das notas musicais, ou seja, cada clave contem uma nota de referência com uma posição fixa, de forma a indicar qual as posições das restantes notas. A Tabela 2 apresenta os símbolos das claves, e o respectivo significado.

|  |  |  |
| --- | --- | --- |
| Símbolo | Nome | Significado |
|  | Clave de Sol | Usada para instrumentos agudos |
|  | Clave de Fá | Usada para instrumentos graves |
|  | Clave de Dó | Usada para instrumentos de sons mediadores agudos |
|  | Clave de Percussão | Usada para instrumentos de percussão |

Tabela - Claves musicais: símbolo e significado.

A existência de vários tipos de claves facilita a leitura musical de vários instrumentos. Por exemplo, o violino usa a clave de sol, o violoncelo usa a clave se Fá. O piano é um instrumento que para escrever todas as notas possíveis é necessário utilizar tanto a clave de Fá como a clave Sol.

### 2.2.3 Figuras musicais

As figuras musicais tem como objectivo representar a duração de tempo que a nota deve ser tocada. A apresenta os símbolos deste tempos e a relação entre os mesmos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nota | Pausa | Código | Nome | Valor Proporcional |
|  |  | 1 | Semibreve | 1 |
|  |  | 2 | Mínima | 1/2 |
|  |  | 4 | Semínima | 1/4 |
|  |  | 8 | Colcheia | 1/8 |
|  |  | 16 | Semicolcheia | 1/16 |
|  |  | 32 | Fusa | 1/32 |
|  |  | 64 | Semifusa | 1/64 |

Tabela 3- Representação dos símbolos dos tempos que uma nota musical poderá ter.

### 

### 2.2.4 Alterações musicais

Uma nota musical pode sofrer alterações na sua frequência original, assim é necessário indicar ao músico essa alteração da nota. A Tabela 4 ilustra os símbolos de alteração e as suas consequências.

|  |  |  |
| --- | --- | --- |
| **Símbolo** | **Nome** | **Objectivo** |
|  | Sustenido | Eleva a nota meio tom |
|  | Dobro Sustenido | Eleva a nota um tom |
|  | Bemol | Desce a nota meio tom |
|  | Dobro Bemol | Desce a nota um tom |
|  | Bequadro | Anula o efeito das alterações |

Tabela - Alterações ou acidentes musicais.

Também existe o símbolo 'ponto' que adiciona metade da duração original à nota. Por exemplo, caso se encontre uma semínima pontuada(), é adicionado à duração original o tempo de uma colcheia como ilustrado na Figura 7.

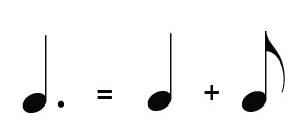


Figura - Aumento do tempo de uma simínima.

### 2.2.5 Compasso

Um compasso é a forma de dividir quantitativamente em grupo de sons de uma composição musical, com base de som e silêncio. Estes podem ser de diversos tipos de divisão, e são representados por fracções matemáticas, tais como, 2/4, 3/4 e outros. O número superior indica a quantidade de notas que deve tocar e o número inferior indica a unidade de tempo que deve ser utilizada. A Figura 8 ilustra o exemplo de um compasso.   
 

Figura - Representação de um compasso.

## 2.3 Instrumentos

Todas as frequências das notas musicais são únicas, ou seja, são iguais para todos os instrumentos. A distinção entre instrumentos é realizada através do respectivo timbre.

Nem todos os instrumentos reproduzem todas as notas apresentadas na   
Tabela 1, porque os seus intervalos de frequências são diferentes. A Tabela 5 mostra alguns dos intervalos dos instrumentos mais conhecidos.

|  |  |
| --- | --- |
| **Instrumento musical** | **Largura de banda(Hz)** |
| Piano | 27,50 - 4.186,00 |
| Tuba | 43,65 - 349,23 |
| Violino | 196,00 - 3.136,00 |
| Viola | 130,81 -1.174,00 |
| Violoncelo | 41,20 - 246,94 |
| Clarinete | 164,81 - 1.567,00 |
| Flauta | 261,63 - 3.349,30 |
| Guitarra | 82,41 - 880,00 |
| Trombone | 82,41 - 493,88 |
| Trompete | 164,81 - 987,77 |

Tabela - Gama de frequências produzidas por alguns instrumentos musicais.

### 2.3.1 Instrumento de Teste

Para realização de testes ao projecto, era necessário que um instrumento que abrangesse as seguintes características:

* Largura de banda elevada, para ser possível aplicar o algoritmo para frequências altas e baixas.
* Produzir frequências próximas, para testar a precisão e resolução do algoritmo.

Em consideração as características anteriormente ditas, foi escolhido o piano, como mostra a Figura 9, como instrumento de teste. O piano é um instrumento da categoria de cordas. Ele contem um teclado de 88 teclas brancas e pretas, que simbolizam as notas musicais. Para produzir o som, o piano contem um conjunto de cordas esticadas e presas a uma estrutura de madeira e batentes, que normalmente são designados por martelos.



Figura - Piano

O piano abrange uma vasta gama de frequências e estas são muito próximas das suas anteriores, como mostra a   
Tabela 6, assim as frequências a captar e processar estão na banda de 27 Hz a 4186 Hz, que representam 8 oitavas. Portanto é necessário no mínimo utilizar uma frequência de amostragem superior a 8372 Hz, respeitando o ritmo de *Nyquist*.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Frequências | D.C.A | Frequências | D.C.A | Frequências | D.C.A | Frequências | D.C.A |
| 27,5000 | - - - | **97,9989** | 5,5003 | **349,2280** | 19,6000 | **1244,5100** | 69,8500 |
| 29,1352 | 1,6352 | **103,8260** | 5,8271 | **369,9940** | 20,7660 | **1318,5100** | 74,0000 |
| 30,8677 | 1,7325 | **110,0000** | 6,1740 | **391,9950** | 22,0010 | **1396,9100** | 78,4000 |
| 32,7032 | 1,8355 | **116,5410** | 6,5410 | **415,3050** | 23,3100 | **1479,9800** | 83,0700 |
| 34,6478 | 1,9446 | **123,4710** | 6,9300 | **440,0000** | 24,6950 | **1567,9800** | 88,0000 |
| 36,7081 | 2,0603 | **130,8130** | 7,3420 | **466,1640** | 26,1640 | **1661,2200** | 93,2400 |
| 38,8909 | 2,1828 | **138,5910** | 7,7780 | **493,8830** | 27,7190 | **1760,0000** | 98,7800 |
| 41,2034 | 2,3125 | **146,8320** | 8,2410 | **523,2510** | 29,3680 | **1864,6600** | 104,6600 |
| 43,6535 | 2,4501 | **155,5630** | 8,7310 | **554,3650** | 31,1140 | **1975,5300** | 110,8700 |
| 46,2493 | 2,5958 | **164,8140** | 9,2510 | **587,3300** | 32,9650 | **2093,0000** | 117,4700 |
| 48,9994 | 2,7501 | **174,6140** | 9,8000 | **622,2540** | 34,9240 | **2217,4600** | 124,4600 |
| 51,9131 | 2,9137 | **184,9970** | 10,3830 | **659,2550** | 37,0010 | **2349,3200** | 131,8600 |
| 55,0000 | 3,0869 | **195,9980** | 11,0010 | **698,4560** | 39,2010 | **2489,0200** | 139,7000 |
| 58,2705 | 3,2705 | **207,6520** | 11,6540 | **739,9890** | 41,5330 | **2637,0200** | 148,0000 |
| 61,7354 | 3,4649 | **220,0000** | 12,3480 | **783,9910** | 44,0020 | **2793,8300** | 156,8100 |
| 65,4064 | 3,6710 | **233,0820** | 13,0820 | **830,6090** | 46,6180 | **2959,9600** | 166,1300 |
| 69,2957 | 3,8893 | **246,9420** | 13,8600 | **880,0000** | 49,3910 | **3135,9600** | 176,0000 |
| 73,4162 | 4,1205 | **261,6260** | 14,6840 | **932,3280** | 52,3280 | **3322,4400** | 186,4800 |
| 77,7817 | 4,3655 | **277,1830** | 15,5570 | **987,7670** | 55,4390 | **3520,0000** | 197,5600 |
| 82,4069 | 4,6252 | **293,6650** | 16,4820 | **1046,5000** | 58,7330 | **3729,3100** | 209,3100 |
| 87,3071 | 4,9002 | **311,1270** | 17,4620 | **1108,7300** | 62,2300 | **3951,0700** | 221,7600 |
| 92,4986 | 5,1915 | **329,6280** | 18,5010 | **1174,6600** | 65,9300 | **4186,0100** | 234,9400 |

Tabela - Gama de frequências (D.C.A - Diferença Com a Anterior).

# 3. Detecção de frequência

Neste capitulo aborda-se os algoritmos de detecção de frequências estudados para o âmbito deste projecto, nomeadamente FFT e Goertzel. Esta secção é discutido as diferenças destes dois algoritmos e indica qual foi o algoritmo escolhido para ser implementado neste projecto.

## 3.1 Discrete Fourier Transform(DFT)

A *DFT* é o calculo da transformada de *Fourier* no domínio da frequência. Este calculo é muito utilizado no mundo da computação, por causa do custo elevado do calculo da transforma de *Fourier*.

Seja um sinal discreto *x* com uma dimensão *N* e em que *xm* é uma amostras desse sinal, o calculo da *DFT* é efectuada pela (1.

|  |  |  |
| --- | --- | --- |
|  | C:\Users\Sorcha\Pictures\DFT.jpg | (1) |

Este calculo é muito dispendioso a nível de poder computacional porque necessita de N2 de multiplicações complexas e N(N-1) somas, desta forma a complexidade computacional é O(N2).

## 3.2 Fast Fourier Trasnform(FFT)

A *FFT* é um algoritmo utilizado para calcular a transformada discreta de Fourier (*DFT*) e a sua inversa. O modo de funcionamento deste algoritmo é repartir uma *DFT* que contém N amostras, por duas *DFT* de comprimento N/2 como mostra a (2

|  |  |  |
| --- | --- | --- |
|  | \begin{matrix} X_k & = & \sum \limits_{m=0}^{N/2-1} x_{2m}     e^{-\frac{2\pi i}{N} (2m)k}   +   \sum \limits_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N} (2m+1)k}.   \end{matrix}  onde Xk é o resultado da *DFT*, N é o número de amostras, x2 é o sinal adquirido. | (2) |

.

O modo de funcionamento do algoritmo *FFT* é igual a uma expansão de uma arvore binária de um vector, ou seja, o algoritmo divide a *DFT* sempre por duas , até que a divisão do N seja igual a zero, de modo que no final apenas tenha que somar os resultados de cada *DFT* que cada filho da arvore contem .

Assim para calcular uma *DFT* utilizando este algoritmo é de multiplicações complexas e somas. Deste modo a complexidade computacional da *FFT* é de

## 3.3 O algoritmo de Goertzel

O algoritmo de *Goertzel* foi criado por Gerald Goertzel em 1958. Este algoritmo calcula um coeficiente da transformada discreta de Fourier (*DFT* – *Discrete Fourier Transform*) através de um filtro recursivo [[8](#Rob01)]. Existem várias versões do algoritmo; neste documento trata-se uma versão optimizada que não tira partido de operações complexas para a detecção de frequências.

A ilustra o diagrama de blocos do filtro de *Goertzel*.

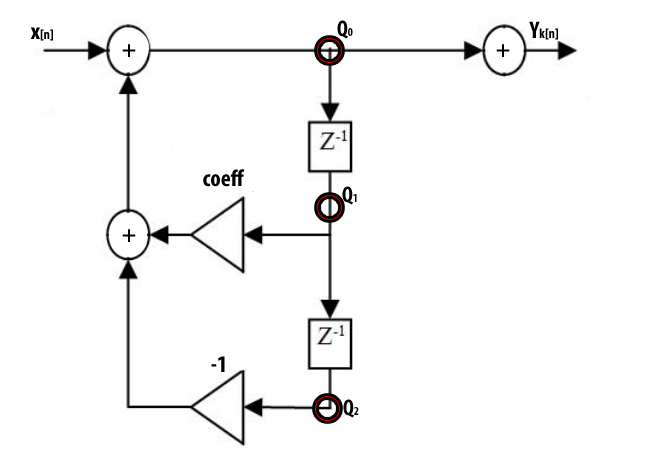


Figura - Diagrama de blocos de um filtro de Goertzel.

### 3.3.1 Descrição

O algoritmo de *Goertzel* [[9](#Gen1)] [[10](#MarcadorPosição1)] detecta a presença de uma dada frequência através de amostragem do espectro do sinal nessa frequência. Calculado o valor do módulo do espectro de amplitude numa dada frequência e comparando-o com a energia total das amostras, é possível verificar quanto é que a frequência contribui para a energia do sinal. Quanto menor a diferença entre a energia do sinal e a energia da frequência, maior é a contribuição da frequência para o sinal. Assim, definindo um limite nesta diferença é possível avaliar se uma frequência se encontra ou não presente no sinal [[11](#Gen111)].

O algoritmo utiliza os seguintes parâmetros:

* Um coeficiente .
* Uma constante *k* que representa a frequência que se pretende detectar.
* O valor da frequência de amostragem *Fs.*
* O valor da frequência que se pretende detectar, *Fn.*
* O número de amostras do sinal que irão ser processadas, *N*.

O valor do coeficiente e da constante *k* são calculados pelas seguintes expressões:

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

A constante *k* tem o valor inteiro mais próximo resultante do arredondamento do resultado da equação (4).

A partir da pode-se deduzir as seguinte equações:

|  |  |  |
| --- | --- | --- |
|  | ;  *onde y(-1) e y(-2) = 0* | (5) |
|  |  | (6) |

A equação (5) representa a relação entre as amostras de entrada x(n) e o resultado do filtro y(n), enquanto que a equação (6) representa a evolução dos valores das unidades de atraso intermédias à medida que as N amostras "circulam" pelo filtro. O filtro guarda apenas os últimos dois estados intermédios para os usar posteriormente na geração de um novo.

Após o processamento de todos os elementos das *N* amostras o algoritmo de *Goertzel* retorna um valor de energia relativa através da equação .

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Na realidade o algoritmo de *Goertzel* não retorna a energia total do espectro bilateral da frequência, isto é, este só retorna o valor da energia da componente positiva do espectro. Sendo assim é necessário multiplicar por dois para obter a energia total da frequência no espectro (8).

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

Para saber se uma dada frequência está presente no sinal é necessário comparar a energia total do sinal com a energia relativa da frequência assim é necessário calcular essa energia relativa com a equação (9):

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

|  |  |  |
| --- | --- | --- |
|  |  |  |

### 3.3.2 Características

O algoritmo optimizado de *Goertzel* não usa operações complexas, e como consequência a sua complexidade aritmética é reduzida necessitando apenas de multiplicações e adições, sendo *N* o número de amostras do sinal na entrada do filtro.

Em memória, em cada instante o algoritmo apenas necessita de ter a amostra actual e os valores intermédios , e , podendo ter em memória não volátil os valores de k e coeficientes.

Outra característica do *Goertzel* é este ser paralelizável uma vez que cada filtro é independente de outros que possam existir, podendo assim detectar várias frequências simultaneamente. Através de um banco de filtros de *Goertzel*, é possível detectar simultaneamente a presença de várias frequências.

A resolução em frequência do algoritmo é dada pela equação (10).

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

A resolução em frequência é o intervalo entre duas frequências detectáveis, ou seja, se tivermos duas frequências *a* e *b* para detectar, a diferença entre estas deve ser maior do que o valor da resolução (11).

|  |  |  |
| --- | --- | --- |
|  | para *b > a* | (11) |
|  |  |  |

Qualquer frequência entre o intervalo ] *a , b* [ que se pretenda detectar irá ser falsamente detectada sempre que *a* ou *b* estejam presentes no sinal.

## 3.4 Escolha do Algoritmo

Quando é necessário resolver um problema que envolva detecção da presença de uma frequência no espectro de determinado sinal, uma das abordagens a considerar é a *Fast Fourier Transform* (*FFT*), devido à sua implementação eficiente e precisão de detecção. A *FFT* diferencia-se do *Goertzel* pelo facto de conseguir detectar várias frequências de uma só vez, porque produz um vector com *N* coeficientes espectrais, enquanto que para cada filtro de *Goertzel* apenas é possível detectar a presença de uma frequência. Essa diferença reflecte-se nas complexidades computacionais e aritméticas destes dois algoritmos, como foi demonstrado nas subsecções anteriores, ou seja, a FTT utiliza multiplicações e somas com números complexos. Mas o Goertzel também pode detectar várias frequências ao mesmo tempo, utilizando a característica de ser paralizável. Os sistemas de alvos que se pretende abranger não contem FPU e apesar de existir técnicas matemáticas(e.g virgula fixa) para retirar as multiplicações complexas, o algoritmo FFT não teria a mesma precisão. Este algoritmo também usa uma quantidade substancial de memória e necessita de uma grande capacidade computacional devido à sua elevada complexidade aritmética. A principal vantagem da *FFT* é a rapidez do cálculo de uma *DFT*, mas não seria aproveitado nos sistemas alvo por causa das suas limitações de recursos computacionais, como velocidade do CPU e a capacidade de memória.

O algoritmo de *Goertzel* em comparação com a *FFT*, é um algoritmo mais leve em relação ao poder de computação necessário, não necessita de uma grande quantidade de memória e é portável para todos tipos de arquitectura (com ou sem *FPU*). A principal vantagem é o facto de ser possível ter diferentes filtros de *Goertzel* em execução com diferentes configurações (*N* e *Fs*), podendo assim optimizar ainda mais o processamento das frequências.

Tanto a *FFT* como o *Goertzel*, descartam o timbre do instrumento e o valor da amplitude deve ser apenas diferente de zero, deste modo, os dois algoritmos podem ser utilizados para detectar frequências de qualquer instrumento.

Concluindo, todos os factores referidos anteriormente tornam o algoritmo de *Goertzel* bastante eficiente, escalável e implementado com pouca memória, tornando-o portável a qualquer tipo de arquitectura.

### 3.5 Implementação do algoritmo de *Goertzel*

 A representa o *flowchart* da implementação do algoritmo de *Goertzel*.

Figura - Máquina de Estados de um filtro de *Goertzel*. O estado "Calcular energia relativa" refere-se à equação

A ilustra o funcionamento do algoritmo como referido anteriormente na equação e . Este algoritmo utiliza uma equação recorrente e necessita de apenas três variáveis locais (*Q0, Q1 e Q2*) para calcular o módulo do espectro de amplitude da frequência que se deseja detectar.

Durante a implementação do algoritmo teve-se de ter em conta a representação numérica das amostras, uma vez que estas deveriam ser o mais próximo possível dos cálculos teóricos. Com este factor em mente foram realizadas duas implementações, uma com valores inteiros e outra com valores decimais (*floating-point*). (descritas na secção *4.6.3* *Port* da infra-estrutura para *ARM*)

## 3.6 Tratamento da Resolução do Goertzel

A resolução do algoritmo de *Goertzel* é dada pela equação (10) descrita anteriormente neste capitulo (secção ). Após a confrontação entre a gama de frequências que se pretende detectar (  
Tabela 6), a mínima frequência de amostragem a poder usar e os requisitos de memória conclui-se que iriam existir problemas na fase experimental do algoritmo de *Goertzel*.

Por exemplo, para os valores de e temos . Isso significa que caso se queira detectar uma frequência com o valor de 440 Hz e que esta se encontre numa dada amostra, o algoritmo de *Goertzel* irá falsamente indicar que as frequências dentro do intervalo se encontram presentes no sinal, introduzindo assim um erro significativo ao processamento das amostras.

A solução ideal seria que o valor de fosse inferior a qualquer diferença entre frequências que se pretende detectar. Na Tabela 7 encontram-se exemplos de algumas frequências que se pretende detectar e a diferença entre as mesmas. Para baixas frequências, a necessidade de ter resolução detalhada leva a que tenha que ser utilizado um número elevado de pontos *N*.

|  |  |
| --- | --- |
| Frequência | Diferença com a anterior |
| 27,5000 | - - - |
| 29,1352 | 1,6352 |
| 30,8677 | 1,7325 |
| 32,7032 | 1,8355 |
| 34,6478 | 1,9446 |
| ... | ... |
| 3520,0000 | 197,5600 |
| 3729,3100 | 209,3100 |
| 3951,0700 | 221,7600 |
| 4186,0100 | 234,9400 |

Tabela - Algumas frequências da   
Tabela 6.

Como ilustrado na  
  
  
  
Tabela 7, as diferenças entre as frequências são crescentes e enquanto que a resolução anteriormente calculada era adequada para as frequências superiores a 3000 Hz não o era para as frequências inferiores a 740 Hz. Assim, foi necessário fazer ajustes de modo a que a resolução nunca seja superior à diferença entre duas frequências consecutivas, de forma a conseguir detectar todas as frequências de interesse.

A solução mais intuitiva seria aumentar o divisor da equação (10), o *N*, para um valor mais próximo de *Fs*, por exemplo com um , o valor de seria 1 sendo inferior a todas as diferenças de frequências. O problema desta solução é que se aumentava consideravelmente o tempo de processamento do algoritmo aumentando igualmente a latência e diminuindo o tempo de resposta aos consumidores do processamento de sinal.

A segunda solução seria diminuir o valor de *Fs*, diminuindo assim também o valor de . A consequência desta solução seria que ao diminuir a frequência de amostragem estaria-se a diminuir o intervalo de frequências possíveis de serem detectadas, pelo teorema de *Nyquist*.

No final a solução adoptada foi um misto das duas anteriores, a frequência de amostragem fica constante para que seja possível processar a gama de frequências que se pretende, realizando decimação do sinal adquirido por *software*. Por exemplo para as primeiras frequências da o seu processamento será realizado com e . Considerando um array de *N* posições onde são guardadas as amostras com uma frequência de amostragem de 8800 Hz, para que os dados sejam processados com um *Fs* de 275 Hz bastará que a indexação a esse *array* seja realizada com índices múltiplos de 32 uma vez que, .

|  |  |  |
| --- | --- | --- |
| ***Gama(Hz)*** | ***Fs (Hz)*** | ***N*** |
| **25,7 - 61,7354** | 275 | 200 |
| **65,4064 - 146,832** | 550 | 200 |
| **155,563 - 349,228** | 1100 | 200 |
| **369,994 - 830,609** | 2200 | 200 |
| **880 - 1975,53** | 8800 | 200 |
| **2093 - 4186,01** | 8800 | 100 |

Tabela - Valores de N e das frequências de amostragem para as frequências do piano.

Com esta solução construiu-se uma aplicação utilitária que tem como funcionalidade calcular os valores de *Fs* e *N* óptimos para processar uma gama de frequências. Na Tabela 8 encontra-se o resultado da execução da aplicação referida anteriormente.  
   
 Com este tratamento foi possível reduzir a resolução do algoritmo de tal forma a que todas as notas sejam correctamente identificadas sem falsas detecções pela resolução. Por exemplo a resolução para a primeira gama da ficou sendo um valor adquado de acordo com os valores apresentados na .

## 

## 3.7 Filtragem do sinal

O facto deste algoritmo apenas se basear no valor do coeficiente para detectar a presença de uma frequência num dado sinal, inviabiliza que existam frequências com o coeficiente igual. Ao tratar o problema da resolução em frequência do algoritmo, agravou-se este problema ainda mais, uma vez que a probabilidade de existirem duas ou mais frequências com o mesmo coeficiente é alta, já que as frequências estão divididas em blocos com frequências de amostragem diferentes e valor de *N* diferentes. A Tabela 9 mostra alguns exemplos deste problema:

|  |  |
| --- | --- |
| Frequências(Hz) | Coeficiente |
| 110; 220; 440; 1760 | 0,61803 |
| 466,164; 116,54; 233,082 | 0,4743 |
| 293,665; 587,33; 2349,32 | -0,21282 |

Tabela - Exemplos de frequências com o mesmo coeficiente.

Uma vez que o problema está com os coeficientes a solução mais directa seria reajustar estes coeficientes até que todos os valores fossem diferentes. O problema desta solução é a complexidade de calcular coeficientes diferentes para todas as 88 frequências quando estas não partilham valores de frequências de amostragem nem de *N*. A solução terá de ser algo exterior ao algoritmo e a sua configuração, portanto optou-se por realizar filtragem de sinal antes da aplicação do algoritmo de Goertzel.

Esta filtragem irá ser realizada para cada gama de frequências () de tal maneira a que as amostras passadas ao algoritmo de *Goertzel* estejam filtradas antes deste efectuar a verificação, evitando assim as falsas detecções.

## 3.7.1 Filtros *FIR*



Figura - Diagrama de blocos de um filtro *FIR*

Para filtrar as amostras foram utilizados filtros do tipo FIR (*Finite Impulse Response*), cujo funcionamento está ilustrado na .

|  |  |
| --- | --- |
|  | () |

A equação corresponde à equação às diferenças do filtro FIR. Em y[n] temos as amostras filtradas. O número de amostras atrasadas presentes no filtro é o mesmo que o número total de coeficientes .

Os coeficientes são calculados a partir da resposta impulsional de um filtro passa-banda, que por sua vez é calculado com a diferença da resposta impulsional de dois filtros passa-baixo como demonstra a equação .

|  |  |
| --- | --- |
|  | () |

Os valores de são dados pela equação:

|  |  |
| --- | --- |
|  | () |

A componente representa a frequência digital e é calculada com a seguinte expressão:

|  |  |
| --- | --- |
|  | () |

Foi ainda aplicada a janela de *Hamming* [[12](#III111)] sobre a resposta impulsional do filtro de maneira a minimizar o ganho das frequências próximas das frequências de corte e simultaneamente tornar o ganho aproximadamente constante na banda de psaagem. Foi ainda realizada uma normalização do ganho do filtro de tal forma a que as frequências contidas na banda de passagem tenham ganho unitário.

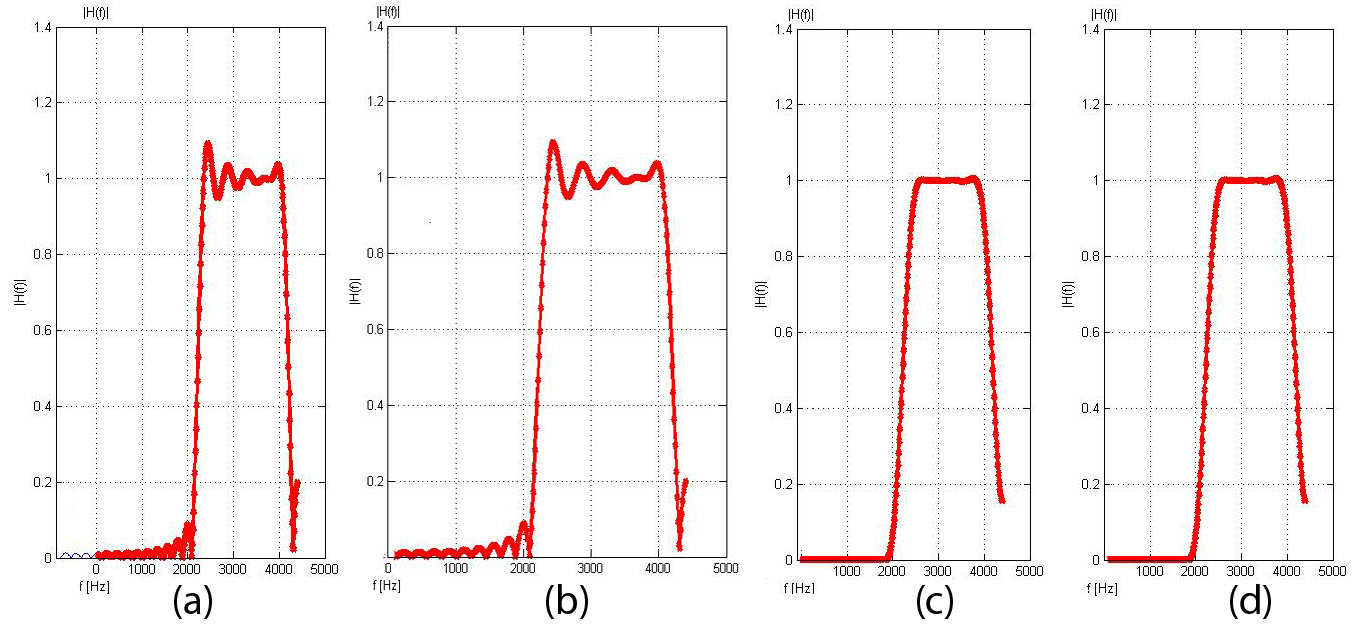


Figura - Resposta em frequência de um filtro passa-banda entre 2217 Hz e 4186 Hz.  
(a) - Filtro normal-  
(b) - Filtro com normalização de ganho.  
(c) - Filtro com janela de Hamming.  
(d) - Filtro com janela de Hamming e normalização de ganho.

Por fim a aplicação descrita anteriormente em foi actualizada de tal forma a que todos os coeficientes dos filtros sejam automaticamente gerados.

# 4. Implementação

Este capitulo trata de explicar as soluções e opções adoptadas na implementação do projecto, bem como dos requisitos a nível de *hardware* e software necessários para correr a aplicação final.

Uma decisão que teve de ser tomada *à priori* da implementação foi os requisitos mínimos do software, ou seja, sobre que tipo de infra-estrutura iria estar dependente. Os requisitos são:

* Uma biblioteca/infra-estrutura que saiba realizar operações com *doubles/floats*.
* Uma biblioteca/infra-estrutura que saiba criar e gerir múltiplos fios de execução (tarefas).
* Suporte para aquisição de sinal.

## 4.1 Linguagem de programação

Um dos aspectos fulcrais para a realização deste projecto foi a boa escolha da linguagem programática, as opções eram à partida limitadas uma vez que como descrito na secção era essencial que o código produzido no âmbito do projecto corresse em qualquer dispositivo com baixas ou altas características a nível de hardware. A escolha ficou reduzida às linguagens *C* e *C++*, ambas reconhecidas pela sua eficiência e propagação no mercado. A linguagem *C* é conhecida por ser uma das mais eficientes uma vez que é a linguagem imperativa "mais próxima" do hardware e pelos seus baixos custos a nível de memória. A linguagem *C++* é uma extensão do *C* introduzindo o paradigma orientado a objectos. Uma das maiores vantagens/desvantagens da linguagem *C++* é o facto de ser extremamente flexível levando a que seja propicia a erros do programador. Apesar disso a linguagem *C++* é muito menos propicia a erros do que a linguagem *C*, por ser uma linguagem tipificada, o código produzido em *C++* é mais genérico (*template metaprograming* [[13](#Bja97)]) e mais legível levando assim a esta ser a linguagem adoptada na elaboração do projecto.

## 4.2 Algoritmo de Goertzel

No capitulo 3 descreveu-se como funciona o algoritmo e que técnicas foram abordadas de forma a que o algoritmo seja utilizado no âmbito deste projecto. Nesta secção do capitulo explicam-se os detalhes de implementação de todos os mecanismos necessários para utilizar o algoritmo.

Uma das características mais interessantes do algoritmo de *Goertzel* é o facto deste ser paralelizável. Esta característica é fulcral para sistemas em que o tempo de resposta deve ser o mais curto possível, como é o caso deste projecto, assim foi criada toda uma infra-estrutura que tira partido e controla todo o processo de processamento de amostras.

### 4.3. GoertzelController

A infra-estrutura *GoertzelController* é responsável por controlar:

* Os filtros Goertzel.
* O buffer onde as amostras são guardadas.
* Quando é que as amostras devem ser entregues aos filtros.
* Quando e como os resultados dos filtros devem ser apresentados.
* Toda a sincronização necessária para aceder a variáveis partilhadas.

A representa o *pipeline* de processamento de sinal utilizando o algoritmo de *Goertzel*.



Figura - Diagrama de blocos do processamento de sinal.

Existem três módulos intrínsecos na infra-estrutura. O primeiro passa por salvaguardar as amostras num buffer interno (*Samples Manager*) onde serão posteriormente enviadas para os filtro *Goertzel,* o segundo é a detecção das frequências presentes na amostra (*Goertzel Filters*) e o último é o armazenamento e complemento dos resultados retornados pelos filtros com *metadata* gerada pelo controlador (*Results Controller*).

### 4.3.1 Configuração e Parametrização

Os módulos do *GoertzelController* funcionam com valores gerados/calculados previamente à sua execução, (coeficientes dos filtros, gama de frequências, etc) para tal foi necessário criar um estrutura que defina esses valores de forma a que fosse possível implementar código genérico ao nível de funcionamento. As estruturas são as seguintes:

* ***GoertzelFrequency*** - representa uma frequência, contem os valores da frequência e do coeficiente de *Goertzel*.
* ***GoertzelFrequenciesBlock*** - representa uma gama de frequências, nesta está presente a frequência de amostragem da gama, o seu valor de *N*, o salto necessário para realizar a divisão da frequência de amostragem por software(), um *array* com os valores do filtro a utilizar sobre esta gama e finalmente o array de frequências que pertencem a esta gama.
* ***GoertzelResult*** - representa o resultado produzido pelo filtro de *Goertzel*, contem uma referencia para a frequência encontrada (*GoertzelFrequency*) e uma percentagem com a diferença entre a energia total do sinal e a energia relativa calculada pelo algoritmo de *Goertzel*.
* ***GoertzelResultCollection*** - representa todos os resultados que a infra-estrutura produziu num dado momento, contem um *array* de resultados (*GoertzelResult*), quantos resultados estão presentes no *array* e quantos **blocos** foram utilizados para produzir os resultados.

Um **bloco** representa uma sequencia de amostras com o tamanho do máximo *N* presente ao longo de todas as gamas.

O *GoertzelController* precisa apenas de um *array* com as gamas de frequências que deve processar (*GoertzelFrequenciesBlock*) sendo a única parametrização necessária para o funcionamento da infra-estrutura.

### 4.3.2 Samples Manager

Este modulo da infra-estrutura tem como principal funcionalidade de salvaguardar as amostras que vão sendo fornecidas ao *GoertzelController*, para além de ir calculando a energia das amostras à medida que vão sendo fornecidas, evitando assim que seja necessário percorrer todas as amostras para calcular a sua energia.

Este módulo tem ainda como responsabilidade saber quando dar amostras aos filtros de *Goertzel*, para tal é necessário ter em *buffer* pelo menos *N* amostras, sendo o valor de *N* o máximo *N* de todas as gamas de frequências previamente calculados. Por fim o módulo ainda permite descartar blocos quando o controlador achar necessário fazê-lo, avisando igualmente os filtros que foram blocos descartados.

### 4.3.3 Goertzel Filters

A infra-estrutura tira partido da paralelização do algoritmo através da criação de diferentes fios de execução (tarefas) para executar cada um dos filtros, como ilustra a Figura 15. 

Figura - Diagrama de blocos do módulo Goertzel Filters.

O segundo módulo do controladoré responsável por filtrar o sinal para a sua gama de frequências e de produzir os resultados. Este filtro usa a mesma estratégia utilizada no módulo *Samples Manager* de ir fazendo trabalho à medida que seja possível*.* Uma particularidade destes filtros é que estes só invocam o *Goertzel* caso exista evidencias que alguma das frequências da gama que está a processar se encontra nas amostras, para tal este vai guardando a energia do sinal filtrado à medida que filtra as amostras, assim quando a condição para chamar o *Goertzel* (ter as *N* amostras) esteja cumprida é comparado o valor da energia do sinal (calculado pelo *Samples Manager*) com a energia do sinal filtrado (calculado no filtro), assim o *Goertzel* é apenas invocado quando a energia do sinal filtrado esteja sobre um limiar configurável. Estes filtros estão ainda preparados para descartar todo o trabalho (filtragem, calculo da energia) caso recebem essa ordem do módulo *Samples Manager.*

### 4.3.4 Results Controller

O *Results Controller* é o módulo responsável por gerir os resultados, esta gestão é necessária uma vez que uma das particularidades desta infra-estrutura é que os filtros podem produzir resultados em alturas diferentes, por exemplo enquanto que a frequência de 4186.01Hz necessita apenas de 179 amostras para ser possível saber se está presente no sinal ou não, a frequência 55Hz necessita de 2704 amostras, assim foi utilizada uma técnica de *double buffering* [[14](#Wik11)]de forma a que os filtros tenham sempre algum local onde preservar os seus resultados. Este módulo gera ainda *metadata* para ser utilizada pela aplicação que utilize esta infra-estrutura, nomeadamente quantas frequências foram encontradas e quantos blocos de amostras foram necessários para encontrar todas as frequências encontradas, com esta informação é possível por exemplo que uma aplicação saiba quanto tempo é que uma dada frequência esteve activa nas amostras passadas ao controlador.

### 4.3.5 Funcionamento e Características

A Figura 16 representa o *flowchart* do controlador:

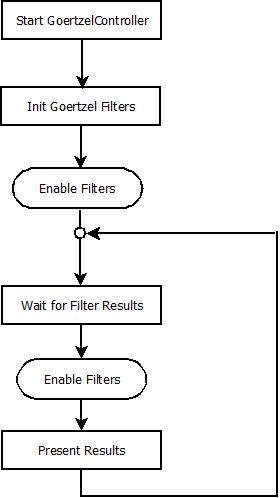


Figura - Flowchart do funcionamento do GoertzelController.

Como já foi referido esta infra-estrutura não funciona apenas com um fio de execução (), mas sim com vários por isso é natural que esta apresente os resultados de forma assíncrona, assim para que uma aplicação obtenha resultados desta infra-estrutura é necessário registar um *callback* com um protótipo definido, representado na Listagem 1.

typedef void (\*GoertzelControllerCallback)(GoertzelResultCollection \* results);

Listagem - Protótipo do callback do GoertzelController.

Na Figura 16 essa característica também é visível uma vez que o controlador depois de esperar pelos resultados do *ResultsController* reactiva os filtros e antes de chamar o *callback* definido (*Present Results*).

A interface publica do controlador permite ainda acesso ao módulo *Samples Manager* de forma a que a infra-estrutura não esteja comprometida com a aquisição do sinal. Este acesso reduz-se a dois modos:

* *Single mode*, a aplicação irá fornecer ao controlador amostras singulares ao longo do tempo para este processar, neste modo o controlador toma conta de gerir toda a informação sobre a amostra e o bloco onde se encontra.
* *Burst mode*, a aplicação irá utilizar um ponteiro fornecido pelo controlador para guardar amostras, a aplicação neste caso é responsável por calcular a energia do sinal bem como toda a lógica necessária para evitar *buffer overflow* [[15](#Wik111)]. Após as amostras desse bloco estarem preenchidas o controlador deve ser avisado de forma a libertar acesso ao bloco para os filtros.

Independentemente do modo de utilização, a aplicação terá sempre de contactar o controlador à medida ou quando acabar de preencher o bloco, assim existe um ponto após a aquisição de sinal onde é possível manter sempre controlo no que diz respeito ao processamento de sinal.

Esta infra-estrutura foi desenhada para ser portável entre arquitecturas de alta ou baixa gama de processamento, assim não existe nenhum compromisso com memória dinâmica nem funções especificas com o sistema, apesar disso existe algum compromisso entre arquitecturas uma vez que para esta infra-estrutura funcionar plenamente é necessário ser possível de criar múltiplos fios de execução (Tarefas).

Outra característica da infra-estrutura é o facto de no inicio da aplicação ser calculado o valor da energia do ambiente, ou seja, por um tempo configurável nenhuma amostra é avaliada pelos filtros mas sim pelo próprio controlador de forma a detectar o ruído presente no ambiente. Assim é possível avaliar, sempre que um bloco de sinal é adquirido, se existe alguma nota presente no sinal, através da comparação entre a energia do sinal recolhida no arranque da infra-estrutura com a energia do bloco adquirido. A Figura 16 representa essa característica da infra-estrutura, apesar de estar desenhada para o modo *Single* o comportamento no modo *Burst* é idêntico.

## C:\Users\DVD\Desktop\SamplesModeFlowchart.jpeg4.4 Calculo da duração das notas

Figura - Flowchart to tratamento de ruído da infra-estrutura.

## 

Como referido na secção 2.1.2 o desenho das notas musicais está relacionado com a duração da sua frequência no tempo. Portanto foi necessário construir algo que através dos resultados fornecidos pelo controlador calculasse a duração de uma dada frequência. Além disso para desenhar uma pauta musical é necessário saber ainda, a ordem a que estas foram tocadas.

O controlador fornece nos resultados *metadata* para que seja possível calcular a duração do tempo, nomeadamente quantos blocos foram utilizados para processar os resultados que num dado instante produziu.

O número de blocos por si só não é suficiente para calcular a duração de uma frequência, é necessário saber qual a frequência de amostragem que está a ser usada na aquisição do sinal, bem como o número de amostras que representa um bloco. Com esta informação foi possível construir um módulo que calculasse o tempo relativo de uma nota.

O módulo *GoertzelTimeController* é responsável por:

* Determinar a duração de uma frequência no tempo, através dos resultados produzidos pelo *GoertzelController*.
* Determinar a ordem temporal a que as frequências foram tocadas, sobre os mesmos resultados.
* Salvaguardar os resultados até que seja oportuno libertá-los para o utilizador.
* Garantir que os acessos à instancia possam ser realizadas por várias tarefas simultaneamente.

O objectivo deste módulo é analisar os resultados do *GoertzelController* de forma a produzir *metadata* adicional para o módulo que ficará encarregue de desenhar a pauta musical.

### 4.4.1 Funcionamento

O módulo *GoertzelTimeController* está directamente associado ao *GoertzelController*. Todos os resultados devem passar por este módulo de forma a determinar o tempo e ordem das notas musicais no tempo.

Este utiliza o valor da frequência de amostragem e do número de amostras por bloco para saber quantos blocos são adquiridos por segundo, por exemplo, se e , o número de blocos por segundo é , assim apenas com o número de blocos processados é possível saber o tempo relativo que uma nota foi tocada.

A apresentação dos resultados depende de diversos factores discutidos na secção 2.1.2, este módulo permite configurar quantos blocos são necessário processar até que sejam apresentados resultados, assim é possível sincronizar o desenho da pauta com os tempos de compassos ou ritmo da musica.

Este módulo é ainda responsável por gerar informação que reflicta a ordem a que as frequências foram detectadas pelo *GoertzelController.* Esta é obtida através da ordem dos resultados recolhidos ao *GoertzelController* em conjunção com uma máquina de estados interna (ao módulo) de forma a saber se a frequência já esteve presente em resultados anteriores ou não.

### 4.4.2 Limitação do módulo *GoertzelTimeController*

Este módulo tem uma limitação funcional, devido à sua dependência directa com o *GoertzelController*, esta encontra-se ilustrada na Tabela 10:

|  |  |  |
| --- | --- | --- |
| Gama (Hz) | Numero de blocos necessários para detectar presença da frequência (blocos) | Duração mínima necessária para detectar frequência (milissegundos) |
| **55 - 130.813** | 16 | 333 |
| **138.591- 329.628** | 8 | 166 |
| **349.228- 830.609** | 4 | 83 |
| **880- 2093** | 1 | 20 |
| **2217.46- 4186.01** | 1 | 20 |

Tabela - Precisão temporal das gamas a detectar com um Fs = 8800.

Na Tabela 10 está ilustrada a dependência entre o número de blocos necessários para detectar uma frequência de uma dada gama com a duração temporal mínima. Um bloco no domínio do tempo representa 20 milissegundos (para Fs = 8800), assim quanto maior for o número de blocos necessários para detectar uma certa frequência maior terá de ser a duração temporal da frequência, uma vez que, o numero de blocos e a duração mínima são directamente proporcionais.

Como ilustrado na o tempo relativo das notas musicais é representado por símbolos cujos valores são proporcionais entre si, por exemplo, admitindo que o tempo absoluto de uma semibreve é de um segundo e que se pretende detectar a duração da frequência 4186.01 Hz (*DÓ8*) num sinal adquirido durante um segundo (8800 amostras = 48 blocos), o resultado obtido estaria entre uma semibreve (1/1) e de uma fusa (1/32 da semibreve) uma vez que a precisão temporal do frequência é de 1/48.

Na ilustra que existe outro símbolo representativo da duração temporal, a semifusa (1/64 da semibreve = 15 milissegundos), com o exemplo descrito anteriormente não seria possível detectar este domínio temporal.

A limitação agrava-se quando o sinal contém mais que uma frequência. Se for adquirido um sinal durante um segundo onde estejam presentes duas frequências X e Y, a precisão temporal será a maior duração mínima entre X e Y, por exemplo, sendo X a frequência 55 Hz e Y a frequência 4186.01 Hz a precisão temporal máxima seria de 333 milissegundos, ou seja, entre a semibreve e a mínima.

Esta limitação pode ser ofuscada através do aumento do tempo absoluto da semibreve, uma vez que, para cada multiplicação que se faça sobre o tempo desta "divide-se" a duração temporal mínima das gamas, por exemplo, se ao invés de um segundo a semibreve tivesse o tempo absoluto de dois segundos a precisão temporal seria como ilustrado na Tabela 11:

|  |  |
| --- | --- |
| Gama (Hz) | Precisão temporal máxima (milissegundos) |
| **55 - 130.813** | 166 |
| **138.591- 329.628** | 83 |
| **349.228- 830.609** | 41 |
| **880- 2093** | 10 |
| **2217.46- 4186.01** | 10 |

Tabela - Precisão temporal das gamas a detectar com um Fs = 8800 e tempo absoluto de uma semibreve igual a 2 segundos (96 blocos).

Um bloco continua a representar 20 milissegundos no domínio do tempo, mas a precisão temporal (mínima duração relativa possível de determinar) aumenta proporcionalmente ao aumento do tempo absoluto da semibreve, uma vez que para este exemplo, a representação de uma semibreve deixa de ser de 48 blocos para 96 levando assim a que seja possível representar outras relações temporais das notas musicais.

## 

## 4.5 Arquitectura PC (Windows)

Toda a implementação da infra-estrutura *GoertzelController* foi realizada sobre a arquitectura de PC de modo a testar a funcionalidade do algoritmo *Goertzel* bem como para calcular o tempo de processamento.

Para fazer a aquisição de sinal foi utilizado a biblioteca de multimédia do *Windows*, onde é possível configurar a placa de som do sistema para colocar amostras num *array*. Posteriormente à recepção das amostras da placa de som, estas são passadas ao controlador através do modo *Single*.

Uma vez que a implementação é executada sob um sistema operativo, a implementação estática da infra-estrutura *GortezelController* é algo desvantajosa uma vez que não utiliza todos os recursos que poderia utilizar (*e.g* memória dinâmica) , mas por outro lado como actualmente praticamente todos os PCs têm múltiplos cores a escalabilidade da infra-estrutura é uma grande vantagem no que diz respeito a tempos de processamento neste tipo de arquitecturas.

## 4.6 Arquitectura ARM7

Neste relatório foram referidas as especificações do *yaab2294* (), kit de desenvolvimento que foi usado para a elaboração deste projecto, as especificações deste hardware estão muito próximas das usadas no mercado (para este tipo de aplicações) tendo sido essa uma das razões para a escolha do hardware.

### 4.6.1 Aquisição de sinal

A aquisição de sinal foi realizada com o *ADC* a 8800 Hz, este gera uma interrupção sempre que contém uma amostra pronta, o tratamento desta trata de guardar o valor da amostra adquirida num *array* fornecido pela infra-estrutura *GoertzelController* e de calcular a contribuição da amostra para a energia total do bloco.

### 4.6.2 Sistema Operativo

Uma das grandes diferenças entre executar a infra-estrutura implementada num *PC* e num *ARM* é o simples facto do *PC* ter um sistema operativo em execução a virtualizar o hardware e como a infra-estrutura foi desenhada para correr sob alguma abstracção (nomeadamente de tarefas) foi necessário encontrar um sistema operativo para portar a infra-estrutura para o *ARM*. Existem muitos sistemas operativos *open-source* para a arquitectura *ARM*. Para a realização deste projecto foram abordados três:

* ***eCos*** - um dos maiores sistemas operativos no mercado, conhecido por estar disponível para várias arquitecturas e *targets* diferentes.
* ***FreeRTOS*** - um sistema bastante mais pequeno que o *eCos*, mas igualmente com alguma presença no mercado.
* ***TNKernel***- o sistema mais pequeno de todos, praticamente só trás consigo mecanismos de sincronização e de escalonamento de tarefas.

O único que seria necessário fazer o *port* seria o *eCos*, uma vez que para os outros dois existem já versões para o processador que se usou no projecto. Na escolha do sistema operativo foi tido em conta a latência do tratamento de interrupções, uma vez que era importante que fosse o mais rápido possível para não perder amostras, o tempo de comutação de uma tarefa, já que a infra-estrutura funciona com múltiplas tarefas em simultâneo é importante que este tempo seja baixo, por fim a implementação dos sincronizadores, uma vez que é importante saber se os sincronizadores desligam as interrupções de forma a obter exclusão para secções criticas, pela mesma razão referida em cima.

Com os critérios definidos, foram avaliados os três sistemas escolhidos, o *eCos* foi excluído quase de imediato devido aos seus elevados tempos de comutação, tanto o *TNKernel* e *FreeRTOS* desligam as interrupções em chamadas a secções criticas inviabilizando-os. A solução ideal seria um sistema operativo que apenas desliga-se as interrupções quando fosse mesmo necessário (e.g. *context switch*). Optou-se então por fazer um sistema operativo que cumprisse todos os requisitos necessários.

#### 4.6.2.1. Micro Operating System (mos)

O *mos* foi criado no âmbito deste projecto de forma a cumprir todos os requisitos necessários para correr a infra-estrutura, este é baseado numa versão em C++ do núcleo *uthreads* leccionado nas aulas de Programação Concorrente. Em comparação com os sistemas operativos referidos neste capitulo (*FreeRTOS* e *TNKernel*), o *mos* oferece alguma virtualização de periféricos básicos no mundo dos embebidos (*e.g. SPI*, *UART*, *TIMER*, etc). Apesar de ter sido criado para cumprir os requisitos da infra-estrutura de processamento de sinal, o *mos* é completamente independente e adjacente a esse facto.

O mos está dividido em três partes:

* ***Kernel*** - o núcleo do sistema operativo, trata de todo o sistema multi-tarefa bem como o tratamento de interrupções.
* ***Port*** - todo o código comprometido com o hardware, este módulo está dividido em arquitectura, onde contém todo o código comum à arquitectura e *target* onde se encontra as *drivers e startups* especificas de um processador/*board*.
* ***System*** - a virtualização do hardware, aqui são definidas contractos que cada *Port* deverá cumprir e/ou implementar para o funcionamento do sistema.

Todos os requisitos da infra-estrutura referidos anteriormente foram cumpridos no kernel do sistema:

* *Tempo de comutação de tarefas reduzido*, este requisito foi implementado sob o *context switch* do sistema, este apenas salvaguarda a informação necessária para que a tarefa possa ser reposta futuramente.
* *Tempo de latência das interrupções baixa*, para diminuir o tempo de latência foi implementado um esquema semelhante ao do *eCos* cuja principal característica é delegar o processamento (que não seja crucial) da interrupção para um tarefa definida pelo sistema, que correrá já com as interrupções ligadas. Outra solução teria sido fazer o tratamento das interrupções dentro do contexto da tarefa que estava a executar previamente ao pedido de interrupção, mas esta solução trás é mais propicia a erros nomeadamente a *buffer overflow (ADD\_REFERENCE).*
* *Sincronização sem desligar as interrupções,* para resolver este requisito foi criado um *lock* de sistema, este *lock* é um simples contador que quando está diferente de zero significa que está adquirido por alguma tarefa. O contador não é mais que uma "*flag*" reentrante que informa o sistema de tratamento de interrupções que não deve alterar nenhuma variável do sistema (*e.g* não deve trocar de tarefas), levando a que o sistema esteja sempre sensível a interrupções.

### 4.6.3 *Port* da infra-estrutura para *ARM*

Após a implementação do sistema operativo *mos* e da aquisição de sinal o *port* da infra-estrutura reduziu-se a alterar as criações das tarefas para usar a API do *mos* à do *Windows*.

Uma vez que a infra-estrutura utiliza aritmética com *floating-point* foi utilizada a biblioteca da *toolchain* que sabe realizar tais operações por software. O problema da utilização desta biblioteca é que acrescentou um peso considerável ao processamento de sinal, uma vez que as operações realizadas nos filtros e no *Goertzel* são multiplicações e adições de *doubles.*

Após a implementação foram realizados testes à infra-estrutura e verificou-se que a infra-estrutura demorava cerca de 36 segundos para processar um bloco de 1 segundo (8800 amostras). Este facto iria contra um dos objectivos do trabalho, latência mínima entre a acção de tocar o instrumento e a reflexão dessa na pauta.

Foi realizada então uma análise temporal sobre todos os processos que constituem a infra-estrutura *GoertzelController*. Nesta chegou-se à conclusão que o processo responsável pela filtragem do sinal, utilizada em todos os filtros de *Goertzel*, estava a consumir grande parte do tempo de processamento medido anteriormente.

O processo de filtragem é uma implementação directa dos filtros FIR apresentados anteriormente na secção 3.7.1. Estes filtros usam multiplicações e somas de valores decimais (*double*) para realizar a filtragem. Como referido anteriormente o *target* utilizado não contém *FPU* logo todas as operações com valores decimais são realizadas por software, originando *bottleneck*. A solução encontrada foi abandonar a precisão *floating-point* e usar inteiros a 64bits. Para não perder toda a precisão dos valores decimais estes foram multiplicados por uma constante.

De modo a optimizar ainda mais o processo de filtragem, a implementação tirou partido de uma característica do tipo de filtros utilizado. Essa característica é o facto dos valores dos coeficientes serem simétricos sobre o seu centro. Assim é possível realizar multiplicações por cada iteração do filtro.

### 4.6.4 Resumo de sistema

Para realizar a implementação do sistema operativo e da infra-estrutura apenas foram utilizados os seguintes periféricos:

* *Timer* - relógio do sistema operativo.
* *ADC* - para aquisição de sinal.
* *LCD* - como interface ao utilizador.

A Tabela 12 descreve os requisitos de memória da implementação do *Maestro*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Data (byte) | BSS(byte) | Text (byte) | Total(byte) |
| *mos* | 0 | 49612 | 26604 | 76216 |
| Infra-estruturas | 44 | 584816 | 23464 | 608324 |
| Total | 44 | 634428 | 50068 | 684540 |

Tabela - Memória utilizada pelo O Maestro.

A quantidade de *bss* utilizada pelo sistema operativo refere-se nomeadamente ao *stack* das tarefas definidas pelo o sistema. Na infra-estrutura além das tarefas criadas pelo controlador para os filtros, existe ainda o *buffer* onde são guardadas as amostras.

Com os valores apresentados na Tabela 12 é possível concluir os requisitos de memória do Maestro, 60 *kb* de *flash* e 650 *kb* de *ram*.

# 5 Testes e Resultados

No âmbito do projecto foram realizados vários testes sob o código produzido, esses consistiram nomeadamente em dois tipos de testes:

* **Testes Temporais** - testes verificar o tempo que demora a dada funcionalidade a executar.
* **Testes Funcionais -** testes realizados com valores teóricos, utilizados nomeadamente para aprovar a implementação do algoritmo e da infra-estrutura.

Neste capitulo irá estar presente vários testes dos tipos referidos anteriormente em diferente *targets*, nomeadamente sob a arquitectura *PC* e *ARM*. Os testes ainda estão divididos entre testes ao algoritmo de *Goertzel*, à infra-estrutura realizada, de forma a validar ambos e ao sistema operativo realizado no âmbito do projecto.

## 5.1 Testes ao algoritmo de Goertzel

Após a implementação do algoritmo de *Goertzel* foi necessário testá-lo em factores como o funcionamento básico do algoritmo, nomeadamente, como é que programaticamente se identifica a existência de uma frequência presente num determinado sinal, bem como quanto tempo demora a processar N amostras. Estes testes foram realizados com a versão com decimais do algoritmo e é de salientar que não realizada qualquer tipo de filtragem do sinal, nem foram efectuados testes funcionais uma vez que o algoritmo nunca irá funcionar como meio isolado.

### 5.1.1 Preparação

Para estes testes foi realizado um módulo de criação de sinusóides, no qual estas são criadas com os seguintes requisitos:

* Frequência de Amostragem(*Fs*) = 8800 kHz
* Amplitude(*A*) = 1000
* Frequência (*fo*) é passada como parâmetro

O cálculo das amostras da sinusóide é efectuado de acordo com a equação )

|  |  |  |
| --- | --- | --- |
|  |  | () |

### 

### 5.1.2 Teste Funcional

Os testes iniciais consistiram em a criar várias sinusóides com diferentes frequências, passá-las a um filtro de *Goertzel* e verificar se as frequências estavam presentes.

Após se verificar que o algoritmo estava devidamente implementado, uma vez que produziu resultados válidos para os testes iniciais, gerou-se um sinal composto por sinusóides de várias frequências (descritas na Tabela 13) e entregou-se ao algoritmo de *Goertzel* de forma a obter resultados. A mostra os resultados obtidos neste teste.

|  |  |  |
| --- | --- | --- |
| Frequências(Hz) | Percentagem do sinal (%) | Percentagem Válida (>10%) |
| 55 | 21.8 | Sim |
| 110 | 22 | Sim |
| 440 | 15.1 | Sim |
| 880 | 15.6 | Sim |
| 1760 | 15.9 | Sim |
| 3520 | 16 | Sim |

Tabela - Resultados do teste teórico ao algoritmo de *Goertzel*   
com sinais compostos por múltiplas sinusoides

A **Percentagem do Sinal** representa a contribuição da frequência para o sinal.

Como se pode verificar nos resultados obtidos todas as frequências foram detectadas porem nota-se que existe alguma divisão da energia do sinal pelas diferentes frequências, mas durante o teste detectaram-se problemas com a resolução do algoritmo (tal como apresentado no subcapítulo ), uma vez que ao testar o algoritmo com frequências com um intervalo curto(*e.g.* primeira oitava do piano) este retornava a indicação da presença de frequências que na realidade não estavam presentes.

### 5.1.3 Teste Temporal

Um dos aspectos mais importantes na escolha de um algoritmo de processamento de sinal é o seu tempo de processamento. Para averiguar esses tempos foram realizados alguns testes que determinam quanto tempo demoraria o *Goertzel* a calcular se alguma das 88 frequências está presente numa dada amostra; apresentam-se os respectivos resultados na .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Tempo Total(ns) | Tempo Relativo(ns) | Arquitectura | Processador-MHz |
| 200 | 6022000000 | 602200 | x86 | I5-2400 |
| 200 | 3260000000 | 326000 | x64 | I7-1600 |
| 200 | 2792000000 | 279200 | x64 | I5-2800 |
| 200 | 10545000000 | 1054500 | ARM7 | LPC2294-60 |
| 2000 | 54117000000 | 5411700 | x86 | I5-2400 |
| 2000 | 30967000000 | 3096700 | x64 | I7-1600 |
| 2000 | 24585000000 | 2458500 | x64 | I5-2800 |
| 2000 | 122060000000 | 12206000 | ARM7 | LPC2294-60 |

Tabela - Resultado do calculo do tempo de processamento do algoritmo de *Goertzel*.

O **tempo total** representa o tempo de execução do teste; o **tempo relativo** representa quanto tempo demora o algoritmo de Goertzel a iterar cada uma das 88 frequências de tal forma a testar se esta está presente ou não; a frequência de amostragem utilizada para estes testes foi de 8800 Hz.

Os testes consistiram maioritariamente em dar várias amostras à implementação do algoritmo e esperar pelos resultados da detecção de frequência.

Os testes foram executados em várias arquitecturas com diferentes processadores de tal forma a que fosse possível visualizar como é que o algoritmo se comportava com diferentes capacidades de processamento, como é visível na Tabela 14 .

## 5.2 Testes à infra-estrutura *GoerzelController*

Os testes à infra-estrutura foram realizados igualmente em duas arquitecturas diferentes de forma a aprovar a sua implementação, os testes apresentados neste subcapítulo sobre a arquitectura PC correm sobre o sistema operativo *Windows* e sobre a arquitectura ARM correram sobre o *mos.*

### 5.2.1 Preparação

Ao contrário dos testes realizados ao algoritmo de *Goertzel* os testes à infra-estrutura são muito mais amplos, uma vez que, com a infra-estrutura vêm resolvido todos os problemas detectados na fase processamento de sinal sendo necessário testar as soluções adoptadas.

A preparação segue os mesmos princípios descritos na preparação para o algoritmo de *Goertzel* mas como foram realizados testes funcionais foi necessário preparar a placa de som no *Windows* e o *ADC* no *mos* de forma a que fosse possível obter amostras de som reais.

### 5.2.2 Teste Funcional

As frequências utilizadas para o teste teórico da infra-estrutura foram as mesmas usadas no teste teórico do algoritmo de Goertzel os resultados estão apresentados na Tabela 15.

Tabela - Resultados do teste teórico à infra-estrutura com múltiplas frequências.

|  |  |  |
| --- | --- | --- |
| Frequências(Hz) | Percentagem do sinal (%) | Percentagem Válida (>10%) |
| 55 | 50 | Sim |
| 110 | 49 | Sim |
| 440 | 67 | Sim |
| 880 | 21 | Sim |
| 1760 | 72 | Sim |
| 3520 | 79 | Sim |

As diferenças nos resultados são visíveis, todas as percentagens das frequências aumentaram consideravelmente, isto deve-se ao facto de haver uma filtragem do sinal antes de ser processado pelo algoritmo de Goertzel, assim a energia da contribuição da frequência para o sinal é comparada com a energia das amostras filtradas e como estas estão filtradas para cada gama, todas as outras frequências que não estejam presentes nessa gama, não estarão "visíveis", não contribuindo assim para o valor da energia filtrada, levando a que a diferença entre a energia total(energia filtrada) e a energia de contribuição da frequência seja menor.

Algo que se pode observar na Tabela 15 é que a frequência 880Hz tem uma percentagem consideravelmente mais baixa em comparação com todas as outras, isso deve-se ao facto de ser uma frequência de corte na gama onde se encontra.

Outra diferença foi o facto de em frequências com intervalo curto não existiu qualquer resultado falso, isto prova que todo o tratamento realizado para resolver a resolução do Goertzel foi propicio.

### 5.2.3 Teste Temporal

Ao contrário do teste temporal realizado para o algoritmo de *Goertzel*, o teste da infra-estrutura não pode ser apenas entregar amostras à infra-estruturas e medir o tempo que esta demora a processar. Isto deve-se necessariamente à forma como a infra-estrutura trabalha as amostras.

Foi necessário então fazer vários testes modulados ao funcionamento da infra-estrutura, para testar e comparar quanto tempo demora o processamento de uma ou várias notas, tendo em atenção se estas se encontram na mesma gama ou não.  
A Tabela 16 mostra os resultados de testes singulares (amostras que só contenham uma frequência).

Tabela - Tempos relativos de processamento da infra-estrutura.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Gama(Hz)*** | Tempo processamento(ns) PC(aprox.) | Tempo processamento(ns) ARM(aprox.) | Frequência utilizada (Hz) |
| **27,5 - 61,7354** | 795000 | 399620000 | 55 |
| **65,4064 - 146,832** | 670000 | 399400000 | 103.826 |
| **155,563 - 349,228** | 655000 | 398870000 | 329.628 |
| **369,994 - 830,609** | 193500 | 389943000 | 587.33 |
| **880 - 1975,53** | 148200 | 89480000 | 1567.98 |
| **2093 - 4186,01** | 127900 | 58170000 | 4186.01 |

Na Tabela 16 nota-se imediatamente a discrepância de valores entre as primeiras gamas e as ultimas, isso deve-se à divisão por software da frequência de amostragem (). É interessante comparar os valores entre a Tabela 14 e a Tabela 16 já que a infra-estrutura conseguiu (para gamas altas) ser mais eficiente do que a implementação do algoritmo isso deve-se ao facto da infra-estrutura não processar sempre as amostras, além disso uma vez que esta sabe descartar silêncio () o processamento em "*long running*" torna-se mais eficiente tanto a nível de memória como a nível de velocidade de execução.

A segunda fase dos testes temporais consistiu em construir um sinal com várias frequências de diferentes gamas e medir o seu tempo, os resultados encontram-se na Tabela 17.

|  |  |  |
| --- | --- | --- |
| **Frequências (Hz)** | **Tempo processamento(ns) PC(aprox.)** | **Tempo processamento(ns) ARM(aprox.)** |
| **55**, **329.628**, **4186.01** | 792000 | 411320000 |
| **329.628, 783.991**, **1244.51**, **4186.01** | 343000 | 455570000 |
| **65.4064, 329.628, 783.991**, **1244.51**, **4186.01** | 792230 | 456230000 |

Tabela - Resultado de testes temporais com várias frequências no sinal.

A Tabela 17 demonstra que a escolha de tornar a infra-estrutura *GoertzelController* multi-tarefa foi correcta, uma vez que como se pode observar os valores de processamento de várias frequências são praticamente o mesmo que levaria a processar a nota com gama mais baixa presente. Na arquitectura ARM o escalonamento do processamento em várias tarefas foi igualmente vantajoso uma vez que o tempo de processamento não cresce linearmente com o aumento do número de frequências no sinal.

## 5.3 Testes temporais aos sistemas operativos

A Tabela 18 mostra os resultados dos testes temporais realizados aos diferentes sistemas operativos. O teste realizado consistiu em trocar sistematicamente duas tarefas uma com a outra durante um número de iterações, o tempo foi medido no inicio e no fim de todas as iterações.

|  |  |
| --- | --- |
| Sistema Operativo | Tempo de comutação(µs) |
| FreeRTOS | 5 |
| TNKernel | 6 |
| eCos | >200 |
| mos | 2 |

Tabela - Tempos de comutação.

Com base na Tabela 18 pode-se concluir que o sistema operativo desenvolvido no âmbito deste projecto tem um tempo de comutação menor que os outros sistemas estudados. Apesar de, em comparação com o *FreeRTOS* e *TNKernel*, a diferença do tempo de comutação ser mínima, se for considerado que as tarefas utilizadas para realizar o processamento dos filtros de *Goertzel* frequentemente entram em mecanismos de sincronização, levando possivelmente a uma comutação, quanto menor for o tempo de comutação maior será o tempo de resposta do sistema.

## 5.4 Conclusão dos testes

Os testes realizados sobre as infra-estruturas produzidas serviram para validar as opções tomadas ao longo da implementação. De salientar que todos os resultados apresentados neste capitulo de tempos de execução sobre a arquitectura *ARM*, foram produzidos com a versão optimizada descrita na secção *4.6.3* *Port* da infra-estrutura para *ARM*.

Com os resultados apresentados anteriormente neste capitulo é possível concluir que é viável utilizar as infra-estruturas produzidas no âmbito do projecto num *target* com baixas especificações, como é o caso do *yaab2294*.

Os testes realizados sobre a arquitectura de PC não foram refeitos com as optimizações utilizadas na arquitectura *ARM*, porque uma vez que o PC tem *FPU* para realizar aritmética com *floating-point*, o ganho em tempo de execução não seria relevante.

# 6. Conclusão

Neste trabalho tratou-se de estudar e implementar uma sistema de detecção de frequências para criar uma aplicação que interpretasse o som e o representa-se na forma de uma pauta musical. Para tal foi utilizado o algoritmo de *Goertzel* pela sua simplicidade e baixa complexidade aritmética. No processo foi criado, uma infra-estrutura que controla o funcionamento do algoritmo e tira partido das suas características e um módulo que determina a duração da frequência do sinal. Como auxiliar foi criada uma aplicação que conhece o funcionamento do algoritmo de *Goertzel* e da infra-estrutura *GoertzelController* de forma a automatizar a configuração do sistema. A implementação foi testada em duas arquitecturas diferentes com especificações diferentes para validar as soluções. Para a arquitectura *ARM* foi desenvolvido o sistema operativo *mos,* após invalidar outras soluções presentes no mercado, para resolver a dependência de suporte a multi-tarefa infra-estruturas, bem como para cimentar conhecimentos adquiridos na área de sistemas operativos.

Os objectivos do projecto foram cumpridos. Aliás da realização do projecto nasceu software reutilizável no futuro.

Da implementação deste projecto resultou duas componentes singulares e interessantes, em primeiro a infra-estrutura produzida para o algoritmo de Goertzel que permite de uma maneira simples (programaticamente) de detectar frequências num sinal em qualquer *target*, e o *Micro Operating System,* um sistema operativo construído praticamente de raiz que não é mais que um aglomerado de conceitos utilizados em grandes sistemas operativos (*eCos*, *Windows*, *FreeRTOS*) de uma forma compacta e funcional.

## 6.1 Dificuldades e desafios

A maior dificuldade encontrada foi entender todos os conceitos necessários para realizar o processamento de sinal, uma vez que os autores não tinham os conhecimentos básicos nesta área. Devido a esta dificuldade nenhum dos prazos que o grupo se comprometeu a cumprir foram atingidos, já que a fase inicial do projecto foi passada a estudar todos os conceitos básicos de processamento de sinal bem com a interiorizar o modo de funcionamento do algoritmo de *Goertzel*. Um dos maiores desafios encontrados na realização deste projecto foi a aprendizagem da linguagem *C++*, apesar de ter sido a escolha dos autores (secção ). A linguagem *C++* apesar de ser uma linguagem de médio nível é muito flexível e propicia a erros do programador levando a que tenha sido passado tempo considerável em resolver erros que apenas existiam devido à própria especificação da linguagem, por exemplo em *C++* os tipos *char*, *unsigned char* e *signed char* são considerados todos diferentes. Outra das dificuldades encontradas foi o ambiente programático, apesar de grande parte de desenvolvimento tenha sido realizado no *Visual Studio*, o sistema operativo e a aplicação foram todos construídas de raiz no *Eclipse CDT.* Apesar do *Eclipse* ser um bom *IDE* para programação de linguagens de alto nível, tem alguns erros no *IntelliSense* de C++ e as suas ferramentas de *debug* para *targets* que não sejam *PC/Mac,* encontram-se repletas de bugs e erros levando a que o tempo de fazer *debug* a uma funcionalidade relativamente simples demora-se três ou quatro vezes mais caso o *target* fosse diferente. Outro desafio foi a criação do sistema operativo, este levou ao máximo os conhecimentos dos autores no que diz respeito a optimizações e programação de baixo nível, bem como a aplicar conceitos ainda não vistos a este nível, nomeadamente o conceito de *Parker* do *SlimThreading* aplicado hoje em dia sobre a máquina virtual *Mono.*

## 6.2 Trabalho futuro

O trabalho mais imediato a ser realizado é o *port* deste projecto para a arquitectura *AVR32,* uma vez que este projecto já foi vencedor do prémio *Application Challenge* da empresa *Bithium* (um prémio de conceito) e os autores fazem questão em participar na segunda fase do concurso denominada *Design Challenge*.

Seria interessante ainda remodelar a aplicação produzida no âmbito deste projecto, que gera os valores dos coeficientes do algoritmo de Goertzel e dos respectivos filtros, de forma a transforma-la num serviço disponível pela *web*.

Uma das características deste projecto é o facto de ser possível montar qualquer tipo de aplicação sobre este, por exemplo, seria interessante gravar as pautas apresentadas, ou ser possível carregar uma pauta para o Maestro e confirmar está a ser tocada correctamente pelo utilizador.

Por fim seria ainda interessante construir um sistema dedicado para o Maestro, uma vez que neste momento o *hardware* utilizado para produzir este projecto foi mínimo em comparação ao que está disponível no *yaab2294*.

# Referências

x

|  |  |
| --- | --- |
| [1] | ARM. The Architecture for the Digital World. [Online]. <http://www.arm.com> |
| [2] | R. Schafer A. Oppenheim, *Discrete-Time Signal Processing 2nd edition*.: Prentice Hall , 1999. |
| [3] | Keil. LPC2294 User Manual. [Online]. <http://www.keil.com/dd/docs/datashts/philips/user_manual_lpc2119_2129_2194_2292_2294.pdf> |
| [4] | Thomas Miller. TMIV ann the dude. [Online]. <http://tmivnthedude.blogspot.com/2007/06/experimentation-with-c.html> |
| [5] | André Cardoso Rocha e RicardoSérgio Caetano Ferreira. Ler Partituras. [Online]. <http://amais.esoterica.pt/documentacao/ler_partituras.pdf> |
| [6] | Curt Sachs, *Rhythm and Tempo: A Study in Music History*. New York: Norton, 1953. |
| [7] | Chuan C. Chang, *Fundamentals of Piano Practice*., 2009. |
| [8] | Andrew G. Dempster, Izzet Kale Robert Beck, "Finite-Precision Goertzel Filters Used for Signal," vol. VOL. 48, no. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING, 2001. |
| [9] | Kevin Banks. The Goertzel Algorithm. [Online]. <http://www.eetimes.com/design/embedded/4024443/The-Goertzel-Algorithm> |
| [10] | R. Schafer A. Oppenheim, *Discrete-Time Signal Processing 2nd edition*.: Prentice Hall, 1999. |
| [11] | Gene Small. Detecting CTCSS tones with Goertzel's algorithm. [Online]. <http://www.eetimes.com/design/embedded/4025660/Detecting-CTCSS-tones-> |
| [12] | Julius O. Smith III. Spectral Audio Signal Processing. [Online]. <https://ccrma.stanford.edu/~jos/sasp/Hamming_Window.html> |
| [13] | Bjarne Stroustrup, *The C++ Programming Language*, 3rd ed.: Addison-Wesley , 1997. |
| [14] | Li Wang, Xiaobo Yan and Xuejun Yang Yu Deng. (2011, Sep.) A Double-Buffering Strategy for the SRF management in the Imagine Stream. [Online]. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4708966> |
| [15] | Tim Leek, Richard Lippmann Michael Zhivich. Dynamic Buffer Overﬂow Detection. [Online]. <http://www.cs.umd.edu/~pugh/BugWorkshop05/papers/61-zhivich.pdf> |
| [16] | Anthony J. Massa, *Embedded Software Development with eCos*.: Techne Group, 2002. [Online]. <http://ecos.sourceware.org/> |
| [17] | The FreeRTOS Project. [Online]. <http://www.freertos.org/> |
| [18] | Yuri Tiomkin. (2011, Aug.) TNKernel real-time system. [Online]. <http://www.tnkernel.com/> |
| [19] | Richard Barry, *Using the FreeRTOS Real Time Kernel - Standard Edition*., 2010. |
| [20] | Julius O. Smith III. (2011, Junho) Center for Computer Research in Music and Acoustics. [Online]. <https://ccrma.stanford.edu/~jos/sasp/Hamming_Window.html> |
| [21] | R. E. Morrow E. 0. Brigham. IEEE Explore. [Online]. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5217220&tag=1> |
| [22] | G. D. Bergland. A guided tour of the fast Fourier transform. [Online]. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5213896&tag=1> |

x