

ISEL

Instituto Superior de Engenharia de Lisboa

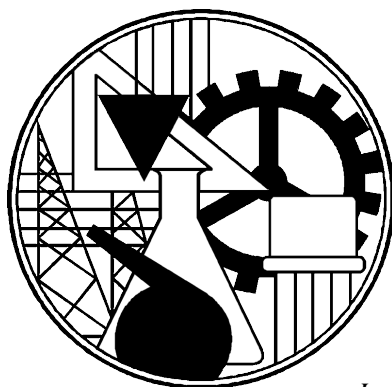
Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

O Maestro

Setembro de 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



ISEL

Instituto Superior de Engenharia de Lisboa

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

O Maestro

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

Setembro 2011

Autores:

Ana Sofia Duarte Correia, N.º 31831 (a31831@alunos.isel.pt)

Diogo Sérgio Esteves Cardoso, N.º 32466 (a32466@alunos.isel.pt)

Orientadores:

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

Artur Jorge Ferreira (arturj@isel.pt)

Resumo

Hoje em dia existem inúmeros dispositivos de tamanhos, formas e técnicas de funcionamento muito diversificados, os quais produzem música para nosso entretenimento. Essa disseminação da música desperta o interesse dos ouvintes em aprofundar o seu conhecimento musical, designadamente a aprender e a produzir música através, por exemplo, da aprendizagem de um instrumento musical. Como qualquer outra forma de arte, o domínio da música não é tarefa fácil, exigindo normalmente o acompanhamento e a direcção de um mestre ou professor, especialmente na fase inicial da aprendizagem. Por esta razão, seria desejável ter à disposição um equipamento com um sistema dedicado de detecção das notas que estão a ser tocadas e que as mostrasse em forma de pauta. Assim a componente humana, o mestre ou professor, poderia nalguns casos ser dispensável no treino da aprendizagem do instrumento que se pretende aprender.

A solução apresentada neste projecto consiste na aquisição e respectivo processamento do sinal produzido por um instrumento e a sua apresentação numa pauta musical. Após um estudo inicial, o algoritmo escolhido para a detecção das frequências das notas musicais foi o de *Goertzel* pela sua eficiência computacional e baixa complexidade aritmética.

Realizou-se a solução proposta sobre a plataforma *PC* e sobre o sistema dedicado *yaab2294*, com arquitectura *ARM*. Este contém um *ADC* para a aquisição de sinal.

Palavras-chave: Sistema dedicado, *Analog to Digital Converter (ADC)*, Algoritmo de *Goertzel*, notas musicais, *Advanced Risc Machine (ARM)*.

Agradecimentos

Aos nossos orientadores pelo apoio, compreensão e paciência nas diversas etapas do projecto.

Ao professor Edmundo Azevedo e Miguel Azevedo pela revisão do relatório com uma perspectiva de fora.

Aos nossos colegas pelas sempre bem-vindas distrações.

Aos nossos pais e irmãos pelo apoio e compreensão dado ao longo da execução deste projecto.

Para os nossos pais e irmãos

Para os nossos amigos

pelo seu constante apoio.

Conteúdo

Resumo	I
Agradecimentos	III
Lista de Figuras	IX
Lista de Tabelas	X
1. Introdução	1
1.1 Motivação	1
1.2 Objectivos e Descrição	2
1.3 Análise de Requisitos.....	3
1.4 Soluções existentes	4
1.5 Organização do documento	5
2. Formulação do Problema.....	7
2.1 Conceitos musicais	7
2.1.1 Notas musicais.....	8
2.2 Notação Musical	10
2.2.1 Pauta	10
2.2.2 Claves musicais	11
2.2.3 Figuras musicais.....	11
2.2.4 Alterações musicais.....	12
2.2.5 Tempo musical	13
2.2.6 Compasso	14
2.3 Instrumentos.....	14
2.3.1 Instrumento de Teste	15
3. Detecção de frequência.....	17
3.1 Discrete Fourier Transform(DFT)	17
3.2 Fast Fourier Transform(FFT).....	18

3.3 O algoritmo de Goertzel	18
3.3.1 Descrição	19
3.3.2 Características	21
3.4 Escolha do Algoritmo	22
3.5 Implementação do algoritmo de <i>Goertzel</i>	23
3.6 Tratamento da Resolução do Goertzel	24
3.7 Filtragem do sinal	27
3.7.1 Filtros <i>FIR</i>	28
4. Implementação	31
4.1 Linguagem de programação	31
4.2 Algoritmo de Goertzel	32
4.3. GoertzelController	32
4.3.1 Configuração e Parametrização	33
4.3.2 Samples Manager	34
4.3.3 Goertzel Filters	34
4.3.4 Results Controller	35
4.3.5 Funcionamento e Características	36
4.4 Cálculo da duração das notas	38
4.4.1 Funcionamento	39
4.4.2 Limitação do módulo GoertzelTimeController	39
4.5 Arquitectura PC (Windows)	41
4.6 Arquitectura ARM	42
4.6.1 Aquisição de sinal	42
4.6.2 Sistema Operativo	42
4.6.3 Port da infra-estrutura para ARM	44
4.6.4 Resumo de sistema	45
5 Testes e Resultados	47

5.1 Testes ao algoritmo de Goertzel	47
5.1.1 Preparação	47
5.1.2 Teste Funcional	48
5.1.3 Teste Temporal.....	49
5.2 Testes à infra-estrutura <i>GoerzelController</i>	50
5.2.1 Preparação	50
5.2.2 Teste Funcional	51
5.2.3 Teste Temporal.....	52
5.3 Testes temporais aos sistemas operativos	53
5.4 Conclusão dos testes	54
6. Conclusão	55
6.1 Dificuldades e desafios	56
6.2 Trabalho futuro	57
Referências	73

Lista de Figuras

FIGURA 1 - FUNCIONAMENTO DO MAESTRO.	2
FIGURA 2- ARQUITECTURA DE <i>SOFTWARE</i> DO PROJECTO	3
FIGURA 3 - REPRESENTAÇÃO DE UMA OITAVA NA ESCRITA EUROPEIA	9
FIGURA 4 - REPRESENTAÇÃO DAS OITAVAS	9
FIGURA 5- EXEMPLO DE UMA PAUTA COM CLAVE DE SOL.	10
FIGURA 6 - AUMENTO DO TEMPO DE UMA SIMÍNIMA.	13
FIGURA 7 - METRÓNOMO ANALÓGICO	13
FIGURA 8- REPRESENTAÇÃO DO TEMPO DE UMA SEMIBREVE EM TEMPO REAL OU EM BPM	13
FIGURA 9- REPRESENTAÇÃO DE UM COMPASSO.	14

FIGURA 10 - PIANO	15
FIGURA 11 - DIAGRAMA DE BLOCOS DE UM FILTRO DE GOERTZEL.	19
FIGURA 12 - MÁQUINA DE ESTADOS DE UM FILTRO DE <i>GOERTZEL</i> . O ESTADO "CALCULAR ENERGIA RELATIVA" REFERE-SE À EQUAÇÃO (9).....	23
FIGURA 13 - DIAGRAMA DE BLOCOS DE UM FILTRO <i>FIR</i>	28
FIGURA 15 - REPRESENTAÇÃO DISCRETA DOS COEFICIENTES DOS FILTROS <i>FIR</i>	30
FIGURA 14 - 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
FIGURA 16 - DIAGRAMA DE BLOCOS DO PROCESSAMENTO DE SINAL.	32
FIGURA 17- DIAGRAMA DE BLOCOS DO MÓDULO GOERTZEL FILTERS.	
FIGURA 18 - FLOWCHART DO FUNCIONAMENTO DO GOERTZELCONTROLLER.....	36
FIGURA 19 - FLOWCHART TO TRATAMENTO DE RUÍDO DA INFRA-ESTRUTURA.	38

Lista de Tabelas

TABELA 1- NOTAS MUSICAIS E AS SUAS FREQUÊNCIAS.	9
TABELA 2- CLAVES MUSICAIS: SÍMBOLO E SIGNIFICADO.....	11
TABELA 3- REPRESENTAÇÃO DOS SÍMBOLOS DOS TEMPOS QUE UMA NOTA MUSICAL PODERÁ TER.	12
TABELA 4 - ALTERAÇÕES OU ACIDENTES MUSICAIS.....	12
TABELA 5 - GAMA DE FREQUÊNCIAS PRODUZIDAS POR ALGUNS INSTRUMENTOS MUSICAIS.	14
TABELA 6 - GAMA DE FREQUÊNCIAS (D.C.A - DIFERENÇA COM A ANTERIOR).	16
TABELA 7 - ALGUMAS FREQUÊNCIAS DA TABELA 6.	25
TABELA 8 - VALORES DE N E DAS FREQUÊNCIAS DE AMOSTRAGEM PARA AS FREQUÊNCIAS DO PIANO.....	26
TABELA 9 - EXEMPLOS DE FREQUÊNCIAS COM O MESMO COEFICIENTE.	27
TABELA 10 - PRECISÃO TEMPORAL DAS GAMAS A DETECTAR COM UM $F_s = 8800$	40
TABELA 11 - PRECISÃO TEMPORAL DAS GAMAS A DETECTAR COM UM $F_s = 8800$ E TEMPO ABSOLUTO DE UMA SEMIBREVE IGUAL A 2 SEGUNDOS (96 BLOCOS).	41
TABELA 12 - MEMÓRIA UTILIZADA PELO <i>O MAESTRO</i>	46

TABELA 13 - RESULTADOS DO TESTE TEÓRICO AO ALGORITMO DE <i>GOERTZEL</i> COM SINAIS COMPOSTOS POR MÚLTIPLAS SINUSOIDES	48
TABELA 14 - RESULTADO DO CÁLCULO DO TEMPO DE PROCESSAMENTO DO ALGORITMO DE <i>GOERTZEL</i>	49
TABELA 15 - RESULTADOS DO TESTE TEÓRICO À INFRA-ESTRUTURA COM MÚLTIPLAS FREQUÊNCIAS.	51
TABELA 16 - TEMPOS RELATIVOS DE PROCESSAMENTO DA INFRA-ESTRUTURA.	52
TABELA 17 - RESULTADO DE TESTES TEMPORAIS COM VÁRIAS FREQUÊNCIAS NO SINAL..	53
TABELA 18 - TEMPOS DE COMUTAÇÃO.....	53

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

A música faz parte do quotidiano das pessoas de todas as idades e classes sociais. Esta disseminação quase universal da música faz com que muitos tenham consigo um dispositivo de reprodução de música, sendo o exemplo mais marcante o dos leitores de *MPEG*¹ – *Layer3*, vulgarmente designados por *MP3*, que existem praticamente em todos os dispositivos móveis, nomeadamente telemóveis. Este contacto diário com a música faz com que muitos desejem alargar o seu conhecimento sobre música, levando-os a aprender a tocar um determinado instrumento. Apesar de existirem diversos meios de estudo e aprendizagem, a interacção humana no âmbito do processo de aprendizagem é fulcral para os iniciados, uma vez que ainda não dispõem de conhecimentos suficientes para usarem correctamente o instrumento que escolheram. Como tal, necessitam de uma interacção forte que os oriente no processo de aprendizagem musical.

Seria por isso interessante e muito conveniente que, para além de um mestre ou professor, existisse outra alternativa neste contexto de aprendizagem musical. Para tal, propomos neste trabalho a criação de *O Maestro*, um sistema dedicado que detecta as notas que estão a ser tocadas no instrumento. Desta maneira, *O Maestro* permite assim que os iniciados comparem as notas tocadas com as que efectivamente deveriam ser tocadas, ajudando assim no processo pedagógico de aprendizagem. Este sistema produz uma pauta musical a partir do som recolhido do instrumento que está a ser tocado.

¹ *Moving Picture Experts Group*

1.2 Objectivos e Descrição

A Figura 1 ilustra o diagrama de blocos dos elementos do projecto e a interacção entre eles.

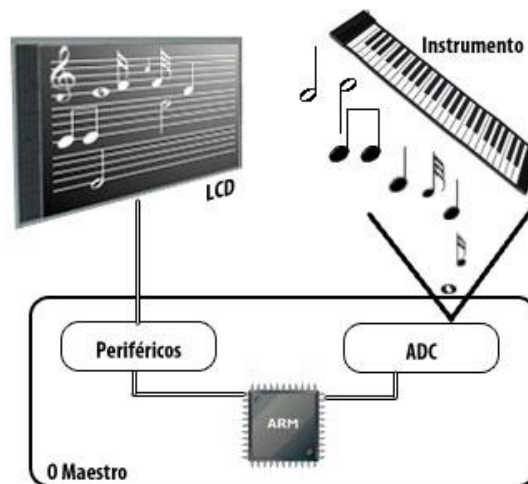


Figura 1 - Funcionamento do Maestro.

O Maestro é um sistema dedicado sobre a arquitectura *Advance Risk Machine (ARM)*[1] que obtém as notas musicais produzidas por um determinado instrumento e processa-as sob a forma de uma pauta musical. Para a recolha de amostras utiliza-se um *Analog to Digital Converter (ADC)* associado ao microcontrolador. Para mostrar as notas tocadas utiliza-se um *Liquid Crystal Display (LCD)* gráfico *touch screen*, como ilustra a Figura 1. No protótipo desenvolvido para o input e output utiliza-se uma porta série associada ao microcontrolador, uma vez que o foco do projecto foi a criação de uma infra-estrutura multi-plataforma de processamento de sinal.

Como apresenta a Figura 2, a componente de *software* deste projecto está dividida em três camadas:

1. *Hardware*, responsável pela interacção directa entre os periféricos internos e externos do microcontrolador.
2. Abstracção ao *hardware*, que 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; é, ainda, a camada onde o algoritmo de *Goertzel* é implementado.

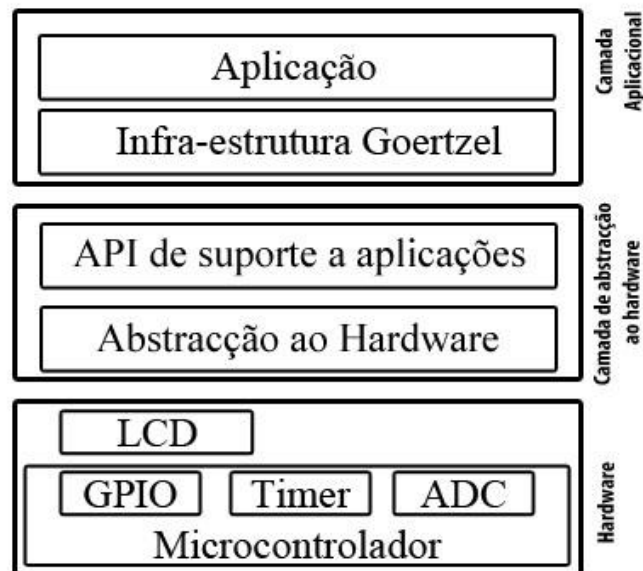


Figura 2- Arquitectura de *Software* do Projecto

O projecto foi desenhado de forma a que o código produzido seja portátil entre arquitecturas, ou seja, toda a camada aplicacional foi implementada de forma a que não dependesse de qualquer característica exclusiva de uma determinada arquitectura.

A implementação do projecto assume que existe hardware/software que realize a aquisição de sinal áudio.

1.3 Análise de Requisitos

Após a análise dos requisitos do projecto, constatou-se que um dos problemas mais relevantes consiste na recolha e processamento das amostras de som. As frequências que se pretendem adquirir e processar estão na banda de 27 Hz a 4186 Hz. Assim, respeitando o teorema de *Nyquist* [2], é necessário, no mínimo, utilizar uma frequência de amostragem superior a 8372 Hz. Para a aquisição do som foi utilizado o

ADC interno do microcontrolador *ARM* que funciona com 10 *bits* por amostra na gama dinâmica de amplitude de 0 a 3,3 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:

- Placa de desenvolvimento *yaab2294*.
 - Microcontrolador baseado na arquitectura *ARM7TDMI - LPC2294* da *NXP* [3].
 - 8 *Mbytes* de memória *RAM*.
 - 8 *Mbytes* de memória *ROM*.
 - *ADC* a 10 *bits*.
 - CPU com velocidade máxima de 60 MHz.
 - *LCD RGB* gráfico (320x240 *pixels*) com *touch screen*.
 - *Ethernet ENC28J60*.
 - Leitor *SD Card*.
- Ferramentas *open-source* da *GNU* para desenvolvimento sobre a arquitectura *ARM*.

1.4 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 que o utilizador toque uma nota musical, normalmente no centro da escala (*e.g.* 440Hz LÁ3), sendo posteriormente o som emitido pelo instrumento avaliado e comparado com a nota tocada pelo utilizador.

A aplicação *Note Detector* [4] realizada em *C#* e *C++* que apresenta características idênticas às da solução que se apresenta neste projecto (*O Maestro*). O *Note Detector* obtém o sinal áudio a partir da placa de som de um PC e utiliza uma biblioteca escrita na linguagem *C++* que implementa o algoritmo *Fast Fourier Transform* (FFT)[2] para identificar as frequências que se encontram no sinal obtido.

Das aplicações semelhantes à proposta neste projecto nenhuma cumpre todos os objectivos do *Maestro*. As aplicações de afinação, apesar de funcionarem numa vasta gama de arquitecturas, estão normalmente comprometidas ao número fixo de notas que conseguem detectar. Em alternativa 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, o algoritmo utilizado para detectar as frequências tem uma elevada complexidade aritmética, como irá ser descrito no Capítulo 3.

1.5 Organização do documento

Este documento está dividido em 6 capítulos.

O Capítulo 2 descreve a formulação do problema e apresenta a explicação os conceitos básicos sobre música.

O Capítulo 3 apresenta os algoritmos de detecção das frequências estudados no âmbito do projecto, as motivações para a escolha do algoritmo Goertzel[2] e a sua descrição detalhada.

O Capítulo 4 contém toda a descrição da implementação do projecto, nas diferentes arquitecturas, *ARM* e *PC*.

No Capítulo 5 são apresentados os resultados dos testes experimentais para a validação das implementações e das opções tomadas.

Por fim, o Capítulo 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

O objectivo do *Maestro* é desenvolver uma infra-estrutura portátil para qualquer arquitectura que identifique notas musicais e as represente sob a forma de uma pauta musical. O principal desafio encontrado foi a realização da componente de processamento de sinal (detecção das frequências), uma vez que a complexidade do processamento, tanto computacional como aritmético, deveria ser relativamente baixo para que possa ser realizado em diferentes arquitecturas com diferentes especificações. Consequentemente, sendo necessário efectuar um levantamento de requisitos à complexidade do processamento a realizar, foram seleccionados os seguintes:

- 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 várias frequências.

Existem vários algoritmos de detecção de frequências, mas com base nos requisitos apresentados acima e no factor de portabilidade, foram considerados dois algoritmos: *Goertzel* na versão optimizada[2] e *Fast Fourier Transform (FFT)* na versão *Cooley-Turkey*[2].

Dado que este projecto aborda conceitos da área da música foi necessário elaborar um estudo sobre esta área. Abordam-se assim conceitos fundamentais sobre música, designadamente, sobre a notação musical, os tempos musicais e as notas musicais, tal como se descreve nas subsecções seguintes.

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 energia do som, isto é, quanto maior a grandeza da vibração produzida, mais enérgico e audível é o som. A altura é a velocidade de vibração do som. O timbre é a característica que o identifica. A duração é o intervalo de tempo em que o som é produzido.

2.1.1 Notas musicais

Uma nota musical[5] é caracterizada pela sua frequência (altura) e a sua duração, isto é, a nota é uma sinusóide que se propaga no ar (ou noutro meio) com uma determinada frequência (definida em *Hertz*) que se encontra no intervalo de 16 Hz a 8000 Hz 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. As frequências médias encontram-se no intervalo de 100 a 400 Hz.

Uma escala musical é representada por 7 notas musicais. Existem duas escritas diferentes para as representar numa escala. A primeira, é usada na Europa e as notas são representadas por Dó, Ré, Mi, Fá, Sol, Lá e Si; a segunda, é usada nos países Anglo-Saxónicos, 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, que se designam por acidentes musicais. Estes são representados pelo nome da nota que irá ser alterada. São exemplos 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. As notas Mi#, Fáb, Si# e Dób não estão presentes porque a frequência que estas notas iriam reflectir, seria igual a outras notas existentes; por exemplo, Mi# teria a mesma frequência que Fá. A Figura 3 mostra a representação destas 12 notas no piano.



Figura 3 - Representação de uma oitava na escrita Europeia

Ao conjunto destas 12 notas dá-se o nome de oitava, existindo ao todo 9 oitavas. As oitavas analisadas da esquerda para a direita, em frequência crescente, vão correspondendo a sons com tons cada vez mais agudos. A Figura 4 ilustra este incremento de frequência ao longo das oitavas.

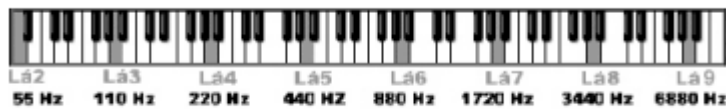


Figura 4 - Representação das oitavas

A Tabela 1 apresenta a frequência de cada nota musical e a oitava a que essa nota pertence.

	C	C#	D	E ^b	E	F	F#	G	G#	A	B ^b	B
Oitavas												
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 1- 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 $2^{1/12}$, ou seja, na razão duodécima de 2. As frequências das notas musicais formam assim uma progressão geométrica. Por exemplo, a frequência da nota seguinte à nota Lá (440Hz) é:

$$\text{PróximaNotaDoLá} = 440 * 2^{1/12} \sim 446 \text{ Hz} = \text{Lá\#}$$

2.2 Notação Musical

A notação musical[5] é um sistema de escrita utilizado para representar uma peça musical através de símbolos gráficos. Com este sistema são identificadas 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 adequadamente a peça musical.

2.2.1 Pauta

A pauta ou pentagrama é utilizada para descrever 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 corresponde apenas uma nota. Além destas linhas principais, existem também linhas suplementares que apenas são utilizadas se existirem notas cuja localização se situe fora das 5 linhas principais (notas mais agudas ou mais graves). Essas linhas não são totalmente desenhadas, mas são apenas o suficiente para o músico perceber a localização exacta da nota. O conjunto de linhas e espaços não tem por si só qualquer significado, sendo necessário ter uma clave para indicar a localização da nota.



Figura 5- 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 contém uma nota de referência com uma posição fixa de forma a indicar quais as posições das restantes notas. A Tabela 2 apresenta os símbolos das claves e o respectivo significado.





Símbolo	Nome	Significado	Nota de Referência
	Clave de Sol	Usada para instrumentos agudos	Sol(3)
	Clave de Fá	Usada para instrumentos graves	Fá(2)
	Clave de Dó	Usada para instrumentos de sons mediadores agudos	Dó(3)
	Clave de Percussão	Usada para instrumentos de percussão	- - - -

Tabela 2- Claves musicais: símbolo e significado.

A existência de vários tipos de claves facilita a leitura musical de vários instrumentos, uma vez que estas estão directamente associadas à gama de notas que cada tipo de instrumento toca Tabela 5. Por exemplo, o violino usa a clave de sol, o violoncelo usa a clave de Fá. O piano é um instrumento que, para escrever todas as notas possíveis necessita de utilizar a clave de Fá e a clave Sol.

2.2.3 Figuras musicais

As figuras musicais têm como objectivo representar a duração de tempo que a nota deve ser tocada. A Tabela 3 apresenta os símbolos destes tempos e as relações temporais entre as diferentes figuras.















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, pelo que 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 4 - 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 (), à duração original é adicionado o tempo de uma colcheia, como ilustra a Figura 6.



Figura 6 - Aumento do tempo de uma simínima.

2.2.5 Tempo musical

O tempo musical é o intervalo entre o som de duas notas consecutivas ou da sua ausência (silêncio). Para marcar o tempo (ou andamento) musical é utilizado o Metrónomo como ilustra a Figura 7.



Figura 7 - Metrónomo Analógico

O metrónomo é um aparelho que mede o tempo musical através da produção de batidas regulares. Estas definem a velocidade da música através do número de batidas que produz por minuto (Batidas por minuto, *BPM*). Quando se associa um *BPM* a uma figura musical (Tabela 3) está-se a definir o seu tempo absoluto.

A Figura 8 ilustra duas formas de representar o tempo absoluto de uma figura musical. Devido à proporcionalidade das figuras musicais (apresentadas na Tabela 3), é apenas necessário definir o tempo absoluto de uma figura musical. Desta forma o compositor poderá indicar ao músico qual o andamento da peça musical.

$$\text{♩} = 1 \text{ segundo} \quad \text{ou} \quad \text{♩} = 60 \text{ BPM}$$

Figura 8- Representação do tempo de uma semibreve em tempo real ou em BPM

2.2.6 Compasso

Um compasso é a forma de dividir quantitativamente em grupo de sons de uma composição musical, com base no som e no silêncio. Estes podem ter diversos tipos de divisão, representados por fracções matemáticas, tais como, $\frac{2}{4}$, $\frac{3}{4}$ e outros. O número superior indica a quantidade de notas que se deve tocar e o número inferior indica a unidade de tempo que deve ser utilizada. A Figura 9 ilustra o exemplo de um compasso.

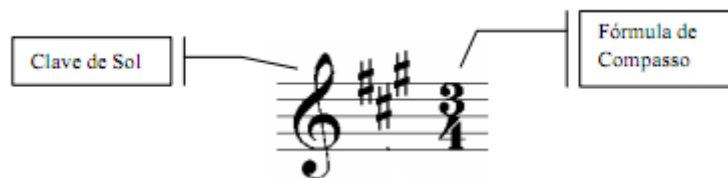


Figura 9- Representação de um compasso.

2.3 Instrumentos

Todas as frequências das notas musicais são únicas, ou seja, são iguais para qualquer instrumento. A distinção entre instrumentos é realizada através do seu timbre. Devidas às suas características físicas nem todos os instrumentos reproduzem todas as notas apresentadas na Tabela 1. A Tabela 5 mostra as gamas de frequência por alguns instrumentos.

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

Tabela 5 - Gama de frequências produzidas por alguns instrumentos musicais.

2.3.1 Instrumento de Teste

Para a realização do projecto foi necessário escolher um instrumento de teste. Para testar a solução de forma abrangente foram tidas em consideração 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, essencialmente nas baixas frequências

Tendo em conta as características anteriormente apresentadas, foi escolhido o piano, ilustrado na Figura 10, como instrumento de teste. O piano é um instrumento da categoria de cordas com um teclado de 88 teclas brancas e pretas que simbolizam as notas musicais. Para produzir o som, o piano contém um conjunto de cordas esticadas presas a uma estrutura de madeira e batentes, que normalmente são designados por martelos.



Figura 10 - Piano

O piano abrange uma vasta gama de frequências, sendo algumas muito próximas, como ilustra a Tabela 6. Assim as frequências a captar e a processar estão na banda de 27 Hz a 4186 Hz, que representam 8 oitavas. É necessário, por isso, utilizar pelo menos 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 6 - Gama de frequências (D.C.A - Diferença Com a Anterior).

Como apresentado na Tabela 6, o piano dispõe de uma largura de banda elevada, bem como de diferenças entre frequências adjacentes pouco acentuadas (27 Hz a 440 Hz), sendo uma opção adequada às características descritas anteriormente nesta Secção.

3. Detecção de frequência

Neste capítulo abordam-se os algoritmos de detecção de frequências estudados no âmbito deste projecto, nomeadamente *FFT* e *Goertzel*. Discutem-se as diferenças entre estes dois algoritmos e apresentam-se as razões que levaram à selecção de um deles para resolver a detecção de frequências.

3.1 Discrete Fourier Transform(DFT)

A *Discrete Fourier Transform DFT*[6] consiste no cálculo da transformada de *Fourier* sobre um sinal discreto representando-o no domínio da frequência, através da amostragem do espectro em determinadas frequências. A *DFT* é muito utilizada como meio de detecção de frequência, dada a sua característica de amostragem de espectro. Seja um sinal discreto x com dimensão N amostras em que x_m é uma amostra desse sinal. O cálculo da *DFT* é realizado através da (1):

$$\begin{aligned} \bar{X}_k &= \sum_{m=0}^{N-1} x_m W^{mk}, \quad k = 0, 1, \dots, N-1 \\ W &= e^{-i\frac{2\pi}{N}}, i = \sqrt{-1} \end{aligned} \quad (1)$$

A complexidade computacional da *DFT* é no pior caso de $O(N^2)$ tornando-a dispendiosa em recursos computacionais. A sua complexidade deve-se ao facto desta necessitar N^2 de multiplicações complexas e de $N(N-1)$ soma complexas. Para valores típicos e práticos de N , na ordem das dezenas e centenas apresenta uma complexidade proibitiva para algumas aplicações.

3.2 Fast Fourier Transform(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 sucessivamente uma *DFT* com N amostras, por duas *DFT* de comprimento $N/2$, apresentado em (2):

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}. \quad (2)$$

onde X_k é o resultado da *DFT*, N é o número de amostras, e x_2 é o sinal adquirido.

O modo de funcionamento da *FFT* é igual a uma expansão de uma árvore binária de um vector, ou seja, o algoritmo sucessivamente decompõe a *DFT* em duas *DFT*, até chegar à situação da *DFT* de 2 pontos. Esta *DFT* de 2 pontos consiste numa soma e numa subtracção. Após esta decomposição é necessário combinar os resultados de cada *DFT* de cada nó da "árvore".

Com a *FFT* o custo de calcular uma *DFT* é de $\frac{(N \log_2 N)}{2}$ multiplicações complexas e $N \log_2 N$ somas. Assim a complexidade computacional da *FFT* é de $O(N \log_2 N)$.

3.3 O algoritmo de Goertzel

O algoritmo de *Goertzel* foi criado por Gerald Goertzel[7] em 1958. Este algoritmo calcula um coeficiente da *DFT* através de um filtro recursivo [8]. Existem várias versões do algoritmo; neste documento utiliza-se uma versão otimizada que não tira partido de operações complexas para a detecção de frequências.

A Figura 11 ilustra o diagrama de blocos do filtro de *Goertzel*.

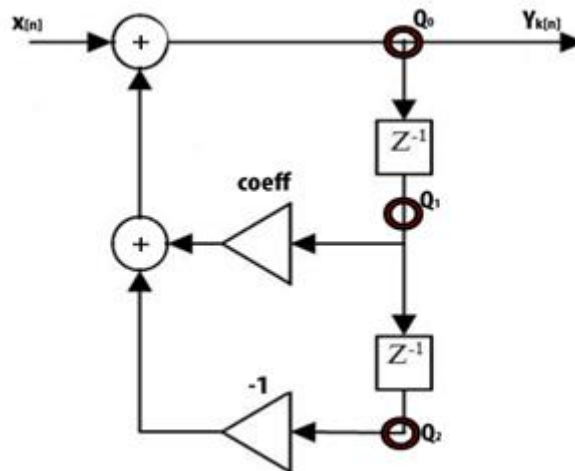


Figura 11 - Diagrama de blocos de um filtro de Goertzel.

3.3.1 Descrição

O algoritmo de *Goertzel*[9][10] detecta a presença de uma dada frequência através da 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 (no domínio do tempo), é possível verificar quanto é que essa 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].

O algoritmo de *Goertzel* utiliza os seguintes parâmetros:

- Coeficiente *coeff*.
- Constante k que representa a frequência que se pretende detectar.
- O valor da frequência de amostragem F_s .
- O valor da frequência que se pretende detectar, F_n .
- O número de amostras do sinal que irão ser processadas, N .

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

$$coeff = 2 * \cos\left(2\pi * \frac{k}{N}\right) \quad (3)$$

$$k = N * \frac{Fn}{Fs} \quad (4)$$

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

A partir da Figura 11 podem deduzir-se as seguintes equações:

$$y[n] = coeff * y[n-1] - y[n-2] + x[n]; \quad (5)$$

onde $y[-1]$ e $y[-2] = 0$

$$Q_0 = coeff * Q_1 - Q_2 + x[n] \quad (6)$$

$$Q_2 = Q_1$$

$$Q_1 = Q_0$$

Em (5) representa-se a relação entre as amostras de entrada $x[n]$ e o resultado do filtro $y[n]$, enquanto que (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 dois últimos 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 à frequência detectada através de (7) .

$$energia^2 = Q_1^2 + Q_2^2 - Q_1 * Q_2 * coeff \quad (7)$$

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 através de:

$$energiaTotal = |X(k)|^2 * 2 \quad (8)$$

Para detectar se uma frequência está presente no sinal compara-se a energia total do sinal com a energia relativa da frequência. Assim é necessário calcular essa energia relativa com:

$$\text{energiaRelativa} = \frac{\text{energiaTotal}}{N} \quad (9)$$

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 $2(N + 2)$ multiplicações e $4(N + 1)$ adições reais, sendo N o número de amostras do sinal na entrada do filtro.

Em memória, em cada instante o algoritmo necessita apenas de ter a amostra actual e os valores intermédios Q_0 , Q_1 e Q_2 , podendo ter em memória não volátil os valores de k e dos 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 por:

$$\Delta = \frac{F_s}{N} \quad (10)$$

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 elas deve ser maior do que o valor da resolução:

$$(b - a) \geq \Delta \quad \text{para } b > a \quad (11)$$

Qualquer frequência no 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 discutido nas secções anteriores. Apesar de o *Goertzel* apenas conseguir detectar uma frequência por cada iteração, é possível detectar várias através da execução de múltiplos filtros *Goertzel* (ver a Secção 3.3.2). Além disso, enquadrando com o projecto, existe um número limitado de frequências que um instrumento consegue emitir simultaneamente.

Comparando com a *FFT*, o algoritmo de *Goertzel* necessita de menos recursos computacionais, necessitando apenas de ter em memória volátil N amostras.

Por exemplo, considerando $N = 200$, a *FFT* realizará $(N \log_2 N) \times 2$ somas e $\frac{(N \log_2 N)}{2} \times 4$ multiplicações reais, ou seja, 920 somas e 920 multiplicações. O algoritmo de *Goertzel* efectuará apenas $2(N + 2)$ multiplicações e $4(N + 1)$ somas reais, isto é, 404 multiplicações e 804 somas no total.

Tanto a *FFT* como o *Goertzel*, descartam o timbre do instrumento, centrando-se na detecção de frequência, ou seja, na presença de energia (amplitude não nula) numa determinada frequência. Concluindo, todos os factores referidos anteriormente tornam o algoritmo de *Goertzel* bastante eficiente e escalável, tornando-o portátil a qualquer tipo de arquitectura.

3.5 Implementação do algoritmo de *Goertzel*

A Figura 12 representa o *flowchart* da implementação do algoritmo de *Goertzel*.

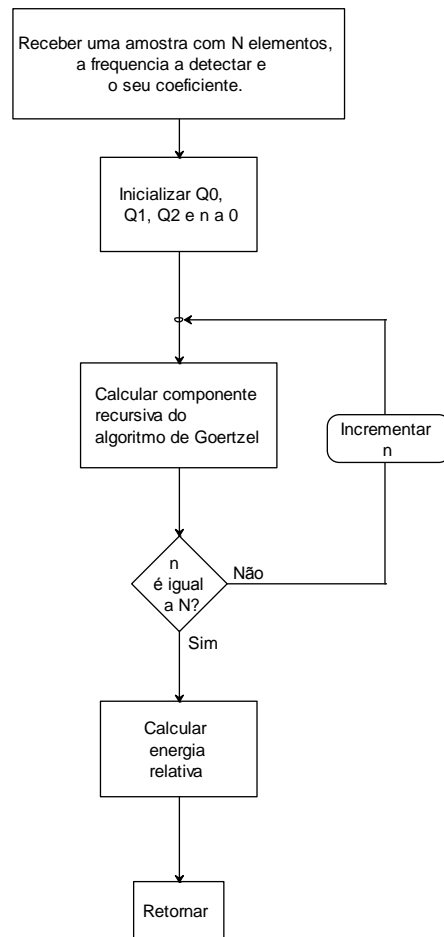


Figura 12 - Máquina de Estados de um filtro de *Goertzel*. O estado "Calcular energia relativa" refere-se à Equação (9)

O funcionamento do algoritmo segue o descrito anteriormente em (5) e Figura 11 apresentas atrás. Este algoritmo utiliza uma equação recorrente e necessita apenas de 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 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).

3.6 Tratamento da Resolução do Goertzel

A resolução do algoritmo de *Goertzel* é dada pela equação (10) descrita anteriormente neste capítulo (Secção 3.3.2). Após a confrontação entre a gama de frequências que se pretende detectar (Tabela 6), a mínima frequência de amostragem que se pode usar e os requisitos de memória concluiu-se que iriam existir problemas na fase experimental do algoritmo de *Goertzel*.

Por exemplo, para os valores de $F_s = 8800$ Hz e $N = 200$ temos $\Delta = 44$. 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 $[440 - \Delta, 440 + \Delta]$ se encontram presentes no sinal, introduzindo, assim, um erro significativo ao processamento das amostras.

A solução ideal seria que o valor de Δ fosse inferior a qualquer diferença entre frequências que se pretendem detectar. Na Tabela 7 encontram-se exemplos de algumas frequências que se pretendem detectar e a diferença entre as mesmas. Para baixas frequências, a necessidade de ter resolução detalhada requer que se deva utilizar 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 7 - Algumas frequências da Tabela 6.

Como ilustra a 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 fosse 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 de (10), o N , para um valor mais próximo de F_s . Por exemplo para $N = 8800$, o valor de Δ seria 1, sendo inferior a todas as diferenças de frequências. Contudo, esta solução 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 F_s , diminuindo assim também o valor de Δ . A consequência desta solução seria que ao diminuir a frequência de amostragem iria diminuir o intervalo de frequências possíveis de serem detectadas, de acordo com o 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 a decimação do sinal adquirido por *software*. Por exemplo para as primeiras frequências da Tabela 7 o seu processamento será realizado com $F_s = 275\text{Hz}$ e

$N = 200$. 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 F_s de 275 Hz bastará que a indexação a esse *array* seja realizada com índices múltiplos de 32 uma vez que,

$$\text{índice} = \frac{F_s}{F_{s\text{Pretendido}}}.$$

Com esta solução construiu-se uma aplicação utilitária que tem como funcionalidade calcular os valores de F_s e N óptimos para processar uma gama de frequências. Na Tabela 8 encontra-se o resultado da execução da aplicação referida anteriormente.

<i>Gama(Hz)</i>	<i>F_s (Hz)</i>	<i>N</i>
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 8 - Valores de N e das frequências de amostragem para as frequências do piano.

Com este tratamento foi possível reduzir a resolução do algoritmo de tal forma que todas as notas fossem correctamente identificadas sem falsas detecções pela resolução empregue. Por exemplo, a resolução para a primeira gama da Tabela 8 ficou $\Delta = \frac{275}{200} = 1,375$ Hz que é um valor adequado, de acordo com os dados apresentados na Tabela 7. Na Listagem 1 encontra-se parcialmente o resultado da execução da aplicação.

```

GoertzelFrequency block5Freqs[] =
{
    { 2093, 0.152647050869377, "C(8)", "DO(8)" },
    { 2217.46, -0.0249321758937832, "C#(8)", "DO#/REb(8)" },
    { 2349.32, -0.212824743993789, "D(8)", "RE(8)" },
    { 2489.02, -0.409796591826102, "D#(8)", "RE#/MIb(8)" },
    { 2637.02, -0.613985438353575, "E(8)", "MI(8)" },
    { 2793.83, -0.822807459405852, "F(8)", "FA(8)" },
    { 2959.96, -1.0327460083145, "F#(8)", "FA#/SOLb(8)" },
    { 3135.96, -1.23926418042578, "G(8)", "SOL(8)" },
    { 3322.44, -1.43668977954882, "G#(8)", "SOL#/LAB(8)" },
    { 3520, -1.61803398874989, "A(8)", "LA(8)" },
    { 3729.31, -1.77503091430041, "A#(8)", "LA#/SIB(8)" },
    { 3951.07, -1.89813380784295, "B(8)", "SI(8)" },
    { 4186.01, -1.97670105124705, "C(9)", "DO(9)" },
};

GoertzelFrequenciesBlock block5 = { 8800, 75, 1, 13, (GoertzelFrequency*)block5Freqs };

```

Listagem 1- Exemplo de um resultado da aplicação utilitária produzida para a gama 2093 - 4186,01 Hz.

3.7 Filtragem do sinal

O facto deste algoritmo se basear apenas 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 9 - 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 reside na complexidade em calcular coeficientes diferentes para todas as 88

frequências quando estas não partilham valores de frequências de amostragem nem de N . A solução teve de ser algo exterior ao algoritmo de *Goertzel* e à sua configuração, tendo-se optado por realizar filtragem passa-banda do sinal antes da aplicação do algoritmo de *Goertzel*.

Esta filtragem foi efectuada para cada gama de frequências (Tabela 8) de tal maneira a que as amostras passadas ao algoritmo de *Goertzel* estivessem filtradas antes deste efectuar a verificação, evitando assim as falsas detecções.

3.7.1 Filtros *FIR*

Para filtrar as amostras foram utilizados filtros do tipo *FIR* (*Finite Impulse Response*), cujo diagrama de blocos genérico está ilustrado na Figura 13.

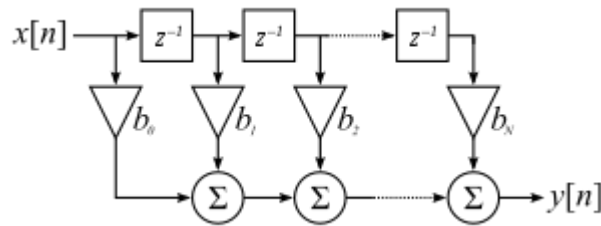


Figura 13 - Diagrama de blocos de um filtro *FIR*

A equação às diferenças do filtro *FIR* é apresentada em (12). Em $y[n]$ temos as amostras filtradas. O número de amostras atrasadas presentes no filtro é o mesmo que o número total de coeficientes K .

$$y[n] = \sum_{k=0}^K b_k \cdot x[n - k] \quad (12)$$

Os coeficientes $b[k]$ 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,

$$b[k] = h[k] = h_{f_1}[k] - h_{f_0}[k], \text{ onde } f_1 > f_0 \quad (13)$$

Os valores de $h_{f_n}(n)$ são dados por:

$$h_{f_n}(n) = \frac{w_0}{\pi} * \text{sinc}\left(\frac{w_0}{\pi} * n\right) \quad (14)$$

O parâmetro w_0 representa a frequência digital, sendo calculado através de

$$w_0 = \frac{2 * \pi * f_n}{F_S} \quad (15)$$

Foi aplicada a janela de *Hamming*[12] sobre a resposta impulsional do filtro passa-banda de forma a minimizar o ganho das frequências próximas das frequências de corte e simultaneamente tornar o ganho aproximadamente constante na banda de passagem. Foi ainda realizada uma normalização do ganho do filtro para que as frequências contidas na banda de passagem tivessem ganho unitário. Na Figura 14 ilustra-se a resposta em frequência de um filtro passa-banda projectado por este processo, com e sem as alterações efectuadas.

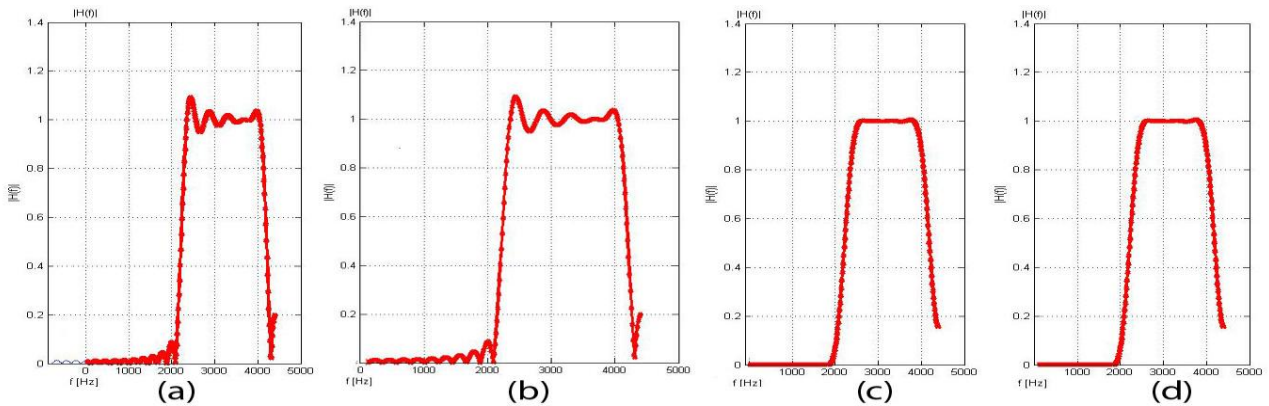


Figura 14 - Resposta em frequência de um filtro passa-banda entre 2217 Hz e 4186 Hz.

- (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.

A aplicação descrita anteriormente foi actualizada de tal forma a que todos os coeficientes dos filtros passa-banda sejam automaticamente gerados. A ordem dos filtros utilizada na Figura 14 foi de 15 coeficientes, ilustrados na Figura 15.

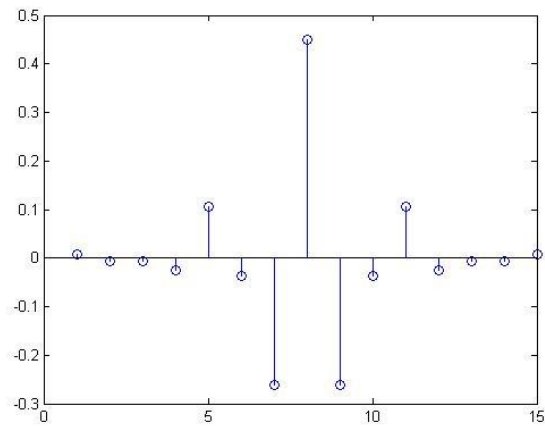


Figura 15 - Representação discreta dos coeficientes dos filtros FIR.

Na Listagem 2 encontra-se o resultado actualizado da aplicação utilitária.

```
GoertzelFrequency block5Freqs[] =
{
    { 2093, 0.152647050869377, "C(8)", "DO(8)" },
    { 2217.46, -0.0249321758937832, "C#(8)", "DO#/REb(8)" },
    { 2349.32, -0.212824743993789, "D(8)", "RE(8)" },
    { 2489.02, -0.409796591826102, "D#(8)", "RE#/MIb(8)" },
    { 2637.02, -0.613985438353575, "E(8)", "MI(8)" },
    { 2793.83, -0.822807459405852, "F(8)", "FA(8)" },
    { 2959.96, -1.0327460083145, "F#(8)", "FA#/SOLb(8)" },
    { 3135.96, -1.23926418042578, "G(8)", "SOL(8)" },
    { 3322.44, -1.43668977954882, "G#(8)", "SOL#/LAB(8)" },
    { 3520, -1.61803398874989, "A(8)", "LA(8)" },
    { 3729.31, -1.77503091430041, "A#(8)", "LA#/SIb(8)" },
    { 3951.07, -1.89813380784295, "B(8)", "SI(8)" },
    { 4186.01, -1.97670105124705, "C(9)", "DO(9)" },
};
double block5filterValues[] =
{
    0.00771114332920066,
    -0.00643552722701511,
    -0.00607663771431723,
    -0.0245428638310063,
    0.104906973936716,
    -0.0374268309086176,
    -0.260482772652715,
    0.449770191745125,
    -0.260482772652715,
    -0.0374268309086176,
    0.104906973936716,
    -0.0245428638310063,
    -0.00607663771431723,
    -0.00643552722701511,
    0.00771114332920066,
};
GoertzelFrequenciesBlock block5 = {8800, 75, 1, 13, block5filterValues, block5Freqs };
```

Listagem 2 - Exemplo de um resultado da aplicação utilitária produzida para a gama 2093 - 4186,01 Hz, com valores dos coeficientes do filtro respectivo.

4. Implementação

Este capítulo apresenta as soluções e as opções efectuadas na implementação do projecto. Na realização do projecto foi utilizado o sistema de desenvolvimento colaborativo de *software*, *Github*[13], este utiliza como sistema de controlo de versões o *Git*[14]. Uma decisão que foi tomada *a priori* da implementação foi definir os requisitos mínimos para a infra-estrutura. Os requisitos mínimos foram:

- Uma biblioteca/infra-estrutura que realize operações com *doubles/floats*.
- Uma biblioteca/infra-estrutura que crie e gere 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 escolha da linguagem programática. As opções eram à partida limitadas, uma vez que, como descrito na Secção 1.2, era essencial que o código produzido corresse em qualquer dispositivo com baixas ou altas características a nível de hardware. A escolha ficou reduzida às linguagens *C* e *C++*[15], ambas reconhecidas pela sua eficiência e difusão 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, entre outras modificações e extensões. Uma das características da linguagem *C++* é o facto de ser extremamente flexível levando a que seja propícia a erros do programador. Apesar disso, a linguagem *C++* é muito menos propícia a erros do que a linguagem *C*, por ser uma linguagem fortemente tipificada, sendo o código produzido em *C++* mais genérico (*template metaprograming*[15]) e mais legível levando assim a esta ser a linguagem adoptada na elaboração do projecto.

4.2 Algoritmo de Goertzel

No Capítulo 3 descreveu-se o funcionamento do algoritmo de detecção de frequência e que técnicas foram abordadas para que o algoritmo seja utilizado no âmbito deste projecto. Apresentam-se agora os detalhes de implementação de todos os mecanismos necessários para a sua realização.

Uma das características mais interessantes do algoritmo de *Goertzel* é o facto de este ser paralelizável. Esta característica é fulcral para sistemas em que o tempo de resposta deve ser o mais curto possível, como é o caso tratado neste 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.
- O momento em que as amostras devem ser entregues aos filtros.
- O momento e a forma como os resultados dos filtros devem ser apresentados.
- Toda a sincronização necessária para aceder a variáveis partilhadas.

A Figura 16 representa o *pipeline* de processamento de sinal utilizando o algoritmo de *Goertzel*.

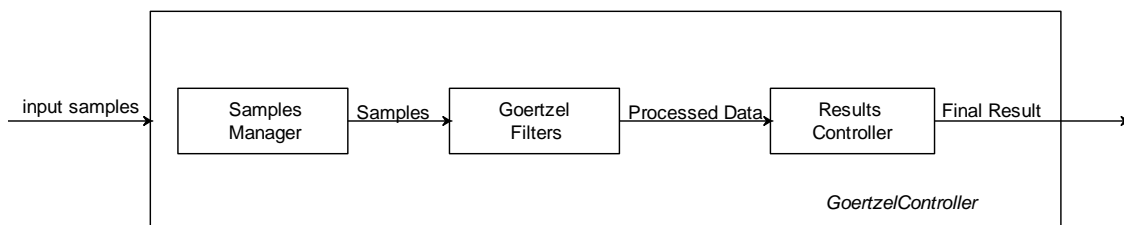


Figura 16 - Diagrama de blocos do processamento de sinal.

Existem três módulos intrínsecos na infra-estrutura. O primeiro passa por armazenar as amostras num buffer interno (*Samples Manager*) onde serão posteriormente enviadas para os filtros *Goertzel*; o segundo é a detecção das frequências presentes na amostra (*Goertzel Filters*); 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 uma estrutura que defina esses valores para que fosse possível implementar código genérico ao nível de funcionamento. As estruturas são as seguintes:

- ***GoertzelFrequency*** - representa uma frequência; contém os valores da frequência e do coeficiente de *Goertzel*.
- ***GoertzelFrequenciesBlock*** - representa uma gama de frequências onde está a frequência de amostragem da gama, o seu valor de N , o salto necessário para realizar a divisão da frequência de amostragem por software (3.6 Tratamento da Resolução do Goertzel), 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*; contém uma referência para a frequência encontrada (*GoertzelFrequency*) assim como 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; contém um *array* de resultados (*GoertzelResult*), quantos resultados estão presentes no *array* e quantos **blocos** foram utilizados para produzir os resultados.

Um **bloco** representa uma sequência 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 módulo da infra-estrutura tem como principal funcionalidade de armazenar 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 fornecer amostras aos filtros de *Goertzel*, sendo para tal necessário ter em *buffer* pelo menos L amostras, sendo o valor de L o máximo N de todas as gamas de frequências previamente calculados. Por fim, o módulo descarta blocos quando o controlador achar necessário fazê-lo, avisando igualmente os filtros que foram descartados blocos.

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 17.

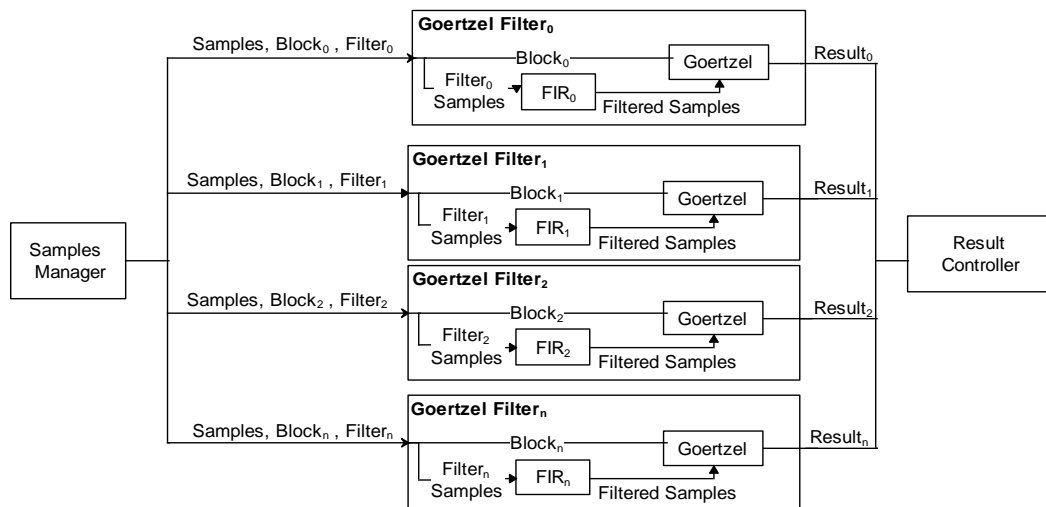


Figura 17 - Diagrama de blocos do módulo Goertzel Filters

O segundo módulo do controlador é 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*, efectuando trabalho à medida que seja possível. Uma particularidade destes filtros é que estes só invocam o *Goertzel* caso exista evidência 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 invocar o *Goertzel* (conter 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, cálculo da energia), caso recebam essa instrução do módulo *Samples Manager*.

4.3.4 Results Controller

O *Results Controller* é o módulo responsável pela gestão dos resultados, necessário pois os filtros podem produzir resultados em alturas diferentes, o que é uma das particularidades desta infra-estrutura. Por exemplo, enquanto a frequência de 4186.01Hz necessita apenas de 179 amostras para que se possa saber se está presente no sinal, a frequência 55 Hz necessita de 2704 amostras. Assim foi utilizada uma técnica de *double buffering*[16] para 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 descobrir todas as frequências detectadas. Com esta informação é possível, por exemplo, que uma aplicação calcule o tempo que uma dada frequência esteve activa nas amostras passadas ao controlador.

4.3.5 Funcionamento e Características

A Figura 17 representa o *flowchart* do controlador:

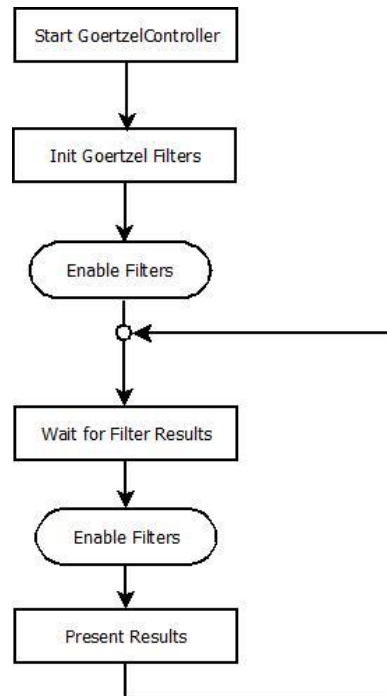


Figura 17 - Flowchart do funcionamento do GoertzelController.

Como já foi referido, esta infra-estrutura não funciona apenas com um fio de execução (3.7.1 Filtros *FIR*), mas sim com vários. Por isso, é natural que apresente os resultados de forma assíncrona e, consequentemente, para que uma aplicação obtenha resultados desta infra-estrutura é necessário registar um *callback* com um protótipo definido, representado na Listagem 3.

```
typedef void (*GoertzelControllerCallback)(GoertzelResultCollection& results);
```

Listagem 3 - Protótipo do callback do GoertzelController.

Na Figura 17 essa característica é também visível, uma vez que o controlador depois de esperar pelos resultados do *ResultsController* reactiva os filtros, antes de chamar o *callback* definido (*Present Results*).

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

- *Single mode* - a aplicação fornece ao controlador amostras singulares ao longo do tempo para este processar. Neste modo o controlador gere toda a informação sobre a amostra e o bloco onde se encontra.
- *Burst mode* - a aplicação utiliza um ponteiro fornecido pelo controlador para guardar amostras. Neste caso a aplicação é responsável pelo cálculo da energia do sinal bem como de toda a lógica necessária para evitar *buffer overflow*[17].

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

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

Outra característica da infra-estrutura é a realização da avaliação 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, sempre que um bloco de sinal é adquirido, é possível avaliar, 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 18 representa essa característica da infra-estrutura, apesar de estar desenhada para o modo *Single*, o comportamento no modo *Burst* é idêntico.

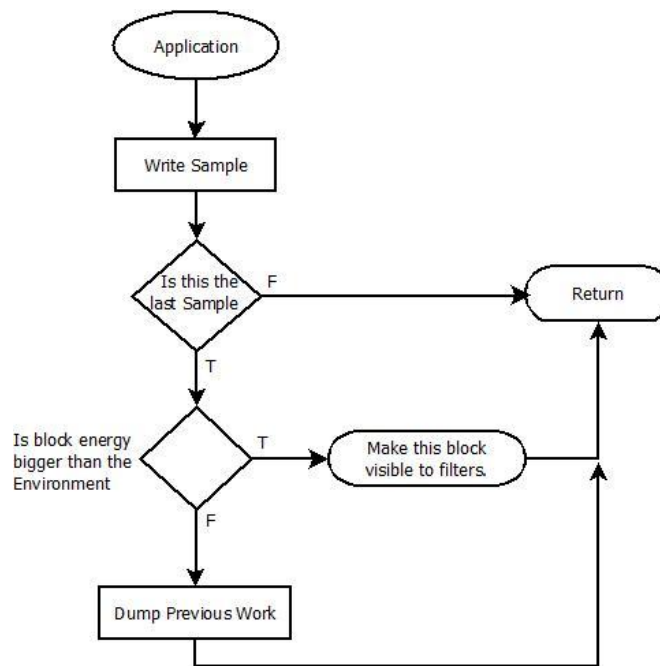


Figura 18 - Flowchart to tratamento de ruído da infra-estrutura.

4.4 Cálculo da duração das notas

Como referido na Secção 2.2.5, o desenho das figuras musicais está relacionado com o tempo que a nota deve tocar. Foi portanto necessário construir um módulo que, através dos resultados fornecidos pelo controlador, calculasse a duração da presença de uma dada frequência. Além disso, para desenhar uma pauta musical é necessário saber, ainda, a ordem com que estas foram tocadas.

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

O número de blocos 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 construído o módulo *GoertzelTimeController*. Este 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.
- Salvar os resultados até que seja oportuno libertá-los para o utilizador.
- Garantir que os acessos à instância possam ser realizados por várias tarefas simultaneamente.

4.4.1 Funcionamento

O módulo *GoertzelTimeController* 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 $F_s = 8800 \text{ Hz}$ e $NrAmostrasPorBloco = 180$, o número de blocos por segundo é 48, calculado através de (16), assim com o número de blocos processados é possível saber o tempo relativo que uma nota foi tocada.

$$NrDeBlocosPorSeg = \frac{F_s}{nrAmostrasPorBloco} \quad (16)$$

A apresentação dos resultados dependem dos diversos factores discutidos na Secção 2.2.5. Este módulo permite configurar quantos blocos são necessários processar até que sejam apresentados resultados.

O módulo *GoertzelTimeController* é ainda responsável por gerar informação que reflecta 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.

4.4.2 Limitação do módulo GoertzelTimeController

Este módulo tem uma limitação funcional, devido à sua dependência directa com o *GoertzelController*, tal como se ilustra na Tabela 10:

Gama (Hz)	Número 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 10 - Precisão temporal das gamas a detectar com um $F_s = 8800$.

A Tabela 10 ilustra a dependência do 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 $F_s = 8800$ Hz); assim quanto maior for o número de blocos necessários para detectar uma certa frequência, maior é a duração temporal da frequência, uma vez que o número de blocos e a duração mínima são directamente proporcionais.

Como ilustrado na Tabela 3 o tempo relativo das notas musicais é representado por figuras musicais cujos valores são proporcionais entre si. Por exemplo, admitindo que o tempo absoluto de uma semibreve é 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 desta frequência é de 1/48. Com esta precisão temporal não é possível detectar a semifusa (1/64).

A limitação agrava-se quando o sinal contém mais de uma frequência. Se for adquirido um sinal durante um segundo onde estejam presentes duas frequências X e Y, a precisão temporal máxima do sinal, 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 11 - Precisão temporal das gamas a detectar com um $F_s = 8800$ 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 determinar) aumenta proporcionalmente com o tempo absoluto da semibreve. 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 utilizada 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 sobre um sistema operativo, a implementação estática da infra-estrutura *GortezelController* é algo desvantajosa, uma vez que não utiliza todos os recursos que a arquitectura disponibiliza (por exemplo, memória dinâmica). Por outro lado, como 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 ARM

As especificações do *yaab2294* o kit de desenvolvimento usado para a elaboração deste projecto foram previamente apresentadas (Secção 1.3). 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, em alternativa à arquitectura PC

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, e o tratamento desta consiste em guardá-la 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 facto de o *PC* ter um sistema operativo em execução a virtualizar o hardware. Assim, para no *ARM* termos uma infra-estrutura desenhada para se executar com 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*[18] - um dos mais completos e divulgados sistemas operativos no mercado, conhecido por estar disponível para várias arquitecturas e *targets* diferentes.
- *FreeRTOS*[19] - um sistema muito mais pequeno que o *eCos*, mas igualmente com alguma presença no mercado.
- *TNKernel*[20]- o sistema mais pequeno de todos, pois praticamente só possui mecanismos de sincronização e de escalonamento de tarefas.

Na escolha do sistema operativo foi tida 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. Uma vez 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 críticas, 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* como o *FreeRTOS* desligam as interrupções em chamadas a secções críticas[21], inviabilizando-os. A solução ideal seria um sistema operativo que apenas desligasse as interrupções quando fosse mesmo necessário (e.g. *context switch*). Optou-se então por desenvolver um sistema operativo que cumprisse todos os requisitos necessários, tal como se descreve de seguida.

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. É baseado numa versão em C++ do núcleo *uthreads* leccionado nas aulas de Programação Concorrente. Em comparação com os sistemas operativos referidos neste capítulo (*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 independente e adjacente a esse facto.

O *mos* está dividido em três partes:

- **Kernel** - o núcleo do sistema operativo, que 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* específicos de um processador/board.

- **System** - a virtualização do hardware, onde são definidos contractos que cada *Port* deve 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, sendo que este apenas armazena 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 uma 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 é mais propícia a erros nomeadamente a *buffer overflow*[17].
- *Sincronização sem desligar as interrupções*: para resolver este requisito foi criado um *lock* de sistema, o qual 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 (por exemplo, *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 à alteração das inicializações 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* para realizar tais operações por software. Contudo, a utilização desta biblioteca 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 resultado iria contra um dos objectivos do projecto, 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* e, logo, todas as operações com valores decimais são realizadas por software, originando um *bottleneck*. A solução encontrada foi abandonar a precisão *floating-point* e usar inteiros a 64 *bits*. Para não perder toda a precisão dos valores decimais, estes foram multiplicados por uma constante, de forma a poder usar aritmética de inteiros, a qual é consideravelmente mais rápida do que a anterior

De modo a otimizar 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 Figura 15 sobre o seu centro (a resposta impulsional é uma função par em torno do seu ponto central). Assim é possível realizar $\frac{N}{2} + 1$ multiplicações por cada iteração do filtro, em vez das N multiplicações sem optimização.

4.6.4 Resumo de sistema

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

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

A Tabela 12 descreve a quantidade de memória utilizada na implementação do *Maestro*.

	Data (byte)	BSS(byte)	Text (byte)	Total(byte)
<i>mos</i>	0	49612	26604	76216
Infra-estruturas	44	584816	23464	608324
Total	44	634428	50068	684540

Tabela 12 - Memória utilizada pelo O *Maestro*.

A quantidade de *bss* utilizada pelo sistema operativo refere-se nomeadamente ao *stack* das tarefas definidas pelo 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 avaliar 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, que consistiram nomeadamente em dois tipos de testes:

- **Temporais** - testem realizados para verificar o tempo de execução de uma dada funcionalidade.
- **Funcionais** - testem realizados com valores teóricos, utilizados nomeadamente para aprovar a implementação do algoritmo e da infra-estrutura.

Neste capítulo apresentam-se os resultados de vários testes dos tipos referidos anteriormente em diferente *targets*, nomeadamente sobre a arquitectura *PC* e *ARM*. Os testes estão ainda 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 se identifica programaticamente 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 do algoritmo que usa valores decimais. De salientar que não foi realizado qualquer tipo de filtragem do sinal, nem foram efectuados testes funcionais. uma vez que o algoritmo nunca iria 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 (F_s) = 8800 Hz.

- Amplitude (A) = 1000.
- Frequência (f_0) é passada como parâmetro.

O cálculo das amostras da sinusóide é efectuado através de:

$$x[n] = A * \sin\left(\frac{2\pi * f_0}{F_s} n\right) \quad (17)$$

5.1.2 Teste Funcional

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

Após a confirmação de 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 Tabela 13 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 13 - 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. 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 na Secção 3.6). De facto, ao testar o algoritmo com frequências com um intervalo curto (por exemplo, primeira oitava do piano) o algoritmo retornava a indicação da presença de frequências que na realidade não estavam presentes.

5.1.3 Teste Temporal

Um dos aspectos mais importantes da escolha de um algoritmo de processamento de sinal é o seu tempo de processamento. Para averiguar esses tempos foram realizados testes que permitiram determinar o tempo que o *Goertzel* demorava a calcular se alguma das 88 frequências estava presente numa dada amostra. Estes resultados são apresentados na Tabela 14.

N	Tempo Total (μ s)	Tempo Relativo (μ s)	Arquitectura	Processador-MHz
200	6022000	602,2	x86	I5-2400
200	3260000	326,0	x64	I7-1600
200	2792000	279,2	x64	I5-2800
200	10545000	1054,5	ARM	LPC2294-60
2000	54117000	5411,7	x86	I5-2400
2000	30967000	3096,7	x64	I7-1600
2000	24585000	2458,5	x64	I5-2800
2000	122060000	12206	ARM	LPC2294-60

Tabela 14 - Resultado do cálculo 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 forma a testar a sua presença; a frequência de amostragem utilizada para estes testes foi de 8800 Hz.

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

Os testes foram executados em várias arquitecturas com diferentes processadores para que fosse possível visualizar o comportamento do com diferentes capacidades de processamento, como mostra a Tabela 14.

5.2 Testes à infra-estrutura *GoerzelController*

Os testes à infra-estrutura foram realizados igualmente em duas arquitecturas diferentes de forma a validar a sua implementação. Os testes apresentados nesta secção sobre a arquitectura *PC* correm sobre o sistema operativo *Windows* e sobre a arquitectura *ARM* usam *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 resolvidos todos os problemas detectados na Secção 5.1.2 sendo necessário testar as soluções adoptadas.

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

5.2.2 Teste Funcional

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

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

Tabela 15 - Resultados do teste teórico à infra-estrutura com múltiplas frequências.

A análise dos resultados mostra que todas as percentagens das frequências aumentam consideravelmente, devido ao facto de ocorrer 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 880 Hz tem uma percentagem consideravelmente mais baixa em comparação com todas as outras, o que se deve 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 detecção falsa, comprovando que o tratamento realizado para adequar a resolução do Goertzel foi adequado.

5.2.3 Teste Temporal

Ao contrário do teste temporal realizado para o algoritmo de *Goertzel*, o teste da infra-estrutura não pode passar apenas pela entrega de 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 modulares 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. A Tabela 16 mostra os resultados de testes singulares (amostras que só contenham uma frequência).

<i>Gama (Hz)</i>	Tempo	Tempo	Frequência utilizada (Hz)
	processamento (μ s) PC (<i>aprox.</i>)	processamento (μ s) ARM (<i>aprox.</i>)	
27,5 - 61,7354	795	399620	55
65,4064 - 146,832	670	399400	103.826
155,563 - 349,228	655	398870	329.628
369,994 - 830,609	193,5	389943	587.33
880 - 1975,53	148,2	89480	1567.98
2093 - 4186,01	127,9	58170	4186.01

Tabela 16 - Tempos relativos de processamento da infra-estrutura.

Na Tabela 16 nota-se imediatamente a discrepância de valores entre as primeiras gamas e as últimas, que se deve à divisão por software da frequência de amostragem (Secção 3.6). É interessante comparar os valores da Tabela 14 com os da Tabela 16, já que a infra-estrutura conseguiu (para gamas altas) ser mais eficiente do que a implementação do algoritmo. Tal deve-se ao facto de a infra-estrutura não processar sempre as amostras e, além disso, uma vez que esta sabe descartar silêncio (Secção 4.3.5), 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 obtidos encontram-se na Tabela 17.

Frequências (Hz)	Tempo processamento(μ s)	Tempo processamento(μ s)
	PC(aprox.)	ARM(aprox.)
55, 329.628, 4186.01	792	411320
329.628, 783.991, 1244.51, 4186.01	343	455570
65.4064, 329.628, 783.991, 1244.51, 4186.01	792,23	456230

Tabela 17 - Resultado de testes temporais com várias frequências no sinal.

A Tabela 17 mostra que a escolha de tornar a infra-estrutura *GoertzelController* multi-tarefa foi correcta, uma vez que 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 presentes 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 entre si durante um número de iterações, sendo o tempo medido no início e no fim de todas as iterações.

Sistema Operativo	Tempo de comutação
	(μ s)
FreeRTOS	5
TNKernel	6
eCos	>200
mos	2

Tabela 18 - 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 inferior aos 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 mais rápida será a 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 capítulo de tempos de execução sobre a arquitectura *ARM*, foram produzidos com a versão optimizada descrita na Secção 4.6.3.

Com os resultados apresentados anteriormente 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 repetidos 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

Este trabalho descreve o desenvolvimento e a implementação de um sistema de detecção de frequências de modo a criar uma aplicação que interpretasse o som e o representasse sob a forma de uma pauta musical. Pela sua simplicidade e baixa complexidade aritmética, foi utilizado o algoritmo de *Goertzel*. No desenvolvimento do projecto foi criada uma infra-estrutura que controla o funcionamento do algoritmo e que tira partido das suas características e, ainda, um módulo que determina a duração temporal da frequência do sinal. Como auxiliar, foi criada uma aplicação que actua sobre 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 e especificações diferentes para validar as soluções propostas. Para a arquitectura *ARM* foi desenvolvido o sistema operativo *mos*, após a invalidação de outras soluções presentes no mercado, para resolver a dependência de suporte a multi-tarefa das infra-estruturas, bem como para assentar conhecimentos adquiridos na área de sistemas operativos.

O objectivo deste projecto foi integralmente cumprido, o que é provado pela possível utilização futura do software desenvolvido neste trabalho.

Da implementação deste projecto resultaram duas componentes singulares e interessantes.

- A infra-estrutura produzida para o algoritmo de Goertzel que permite de uma maneira simples (programaticamente) detectar frequências num sinal em qualquer *target*.
- O *Micro Operating System*, um sistema operativo construído praticamente de raiz que não é mais do que um aglomerado de conceitos utilizados em grandes sistemas operativos (*eCos*, *Windows*, *FreeRTOS*) de uma forma compacta e funcional e adequado ao problema em causa.

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, não foram infelizmente cumpridos os prazos com que o grupo se tinha comprometido, já que a fase inicial do projecto exigiu o estudo dos conceitos básicos de processamento de sinal, bem com a interiorizar o modo de funcionamento do algoritmo de *Goertzel*.

Outro dos maiores desafios encontrados na realização deste projecto foi a aprendizagem da linguagem C++, apesar de ter sido a escolha dos autores (Secção 4.1). Apesar de ser uma linguagem de médio nível, a linguagem C++ é muito flexível e propícia a erros do programador o que conduziu inevitavelmente a um gasto de tempo considerável na correcção de erros que apenas existiram 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 construídos 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 *debugging*, que é uma funcionalidade relativamente simples, demorasse três ou quatro vezes mais caso o *target* fosse diferente.

Outro grande desafio foi a criação do sistema operativo, que 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 transformá-la num serviço disponível pela *web*.

Uma das características deste projecto é o facto de ser facilmente extensível com novas funcionalidades. 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

- [1] ARM. The Architecture for the Digital World. [Online]. Último acesso: 9 de Setembro 2011
<http://www.arm.com>
- [2] R. Schafer A. Oppenheim, *Discrete-Time Signal Processing 2nd edition.*: Prentice Hall , 1999.
- [3] Keil. LPC2294 User Manual. [Online]. Último acesso: 9 de Setembro 2011
http://www.keil.com/dd/docs/datashts/philips/user_manual_lpc2119_2129_2194_2292_2294.pdf
- [4] Thomas Miller. TMIV ann the dude. [Online]. Último acesso: 9 de Setembro 2011
<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] R. E. Morrow E. O. Brigham. IEEE Explore. [Online].
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5217220&tag=1>
- [7] Gerald Goertzel, "An Algorithm for the Evaluation of Finite Trigonometric Series," *American Mathematical Monthly* 65, pp. 34-35, 1958.
- [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]. Último acesso: 9 de Setembro 2011
<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]. Último acesso: 9 de Setembro 2011
<http://www.eetimes.com/design/embedded/4025660/Detecting-CTCSS-tones->
- [12] Julius O. Smith III. Spectral Audio Signal Processing. [Online]. Último acesso: 9 de Setembro 2011
https://ccrma.stanford.edu/~jos/sasp/Hamming_Window.html

- [13] Github Social Coding. [Online]. Último acesso: 9 de Setembro 2011
www.github.com
- [14] Git the fast version control system. [Online]. Último acesso: 9 de Setembro 2011
<http://git-scm.com/>
- [15] Bjarne Stroustrup, *The C++ Programming Language*, 3rd ed.: Addison-Wesley , 1997.
- [16] 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>
- [17] Tim Leek, Richard Lippmann Michael Zhivich. Dynamic Buffer Overflow Detection. [Online].
<http://www.cs.umd.edu/~pugh/BugWorkshop05/papers/61-zhivich.pdf>
- [18] Anthony J. Massa, *Embedded Software Development with eCos*: Techne Group, 2002. [Online].
Último acesso: 9 de Setembro 2011
<http://ecos.sourceware.org/>
- [19] The FreeRTOS Project. [Online]. Último acesso: 9 de Setembro 2011
<http://www.freertos.org/>
- [20] Yuri Tiomkin. (2011, Aug.) TNKernel real-time system. [Online]. Último acesso: 9 de Setembro 2011
<http://www.tnkernel.com/>
- [21] Richard Barry, *Using the FreeRTOS Real Time Kernel - Standard Edition*., 2010.
- [22] Julius O. Smith III. (2011, Junho) Center for Computer Research in Music and Acoustics.
[Online]. Último acesso: 9 de Setembro 2011
https://ccrma.stanford.edu/~jos/sasp/Hamming_Window.html
- [23] G. D. Bergland. A guided tour of the fast Fourier transform. [Online].
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5213896&tag=1>
- [24] Chuan C. Chang, *Fundamentals of Piano Practice*., 2009.

[25] Curt Sachs, *Rhythm and Tempo: A Study in Music History*. New York: Norton, 1953.