– INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Coluna de som utilizando array de Transdutores de Ultrasons

Paulo Jaime Estrela Janganga
(Grau académico)

Trabalho Final de Mestrado para Obtenção do Grau de Mestre em Engenharia de Electrónica e Telecomunicações

Orientador:
Doutor Joel Vera Cruz Preto Paulo

Júri:
Presidente:
Doutora Maria Manuela Almeida Carvalho Vieira

Vogais:
Doutor Mário Pereira Véstias
Doutor Joel Vera Cruz Preto Paulo

Dezembro de 2015

Resumo

Este relatório documenta a Tese de Mestrado com o tema *Audio Spotlight* - Coluna de Som utilizando Arrays de Transdutores de Ultrassons realizado por Paulo Jaime Estrela Janganga.

Este trabalho assenta na implementação de um sistema de áudio direcional *Audio Spotlight*. Atualmente começam-se a dar os primeiros passos ao nível da comercialização de produtos com este tipo de tecnologia. No entanto, existem ainda aspetos técnicos que necessitam de ser resolvidos antes da sua aplicação no nosso dia-a-dia.

O problema principal é a distorção que um sistema deste tipo gera ao utilizar interações não-lineares de ondas de alta frequência (ultrassons) para gerar ondas de baixa frequência dentro da gama audível pelos seres humanos. Depois de resolvido este e outros problemas adjacentes ao sistema, o seu potencial é promissor, sendo a lista de aplicações práticas e comerciais muito abrangente. Alguns exemplos são:

• Num automóvel, em que cada um dos ocupantes poderá ouvir a sua própria música, sem perturbar os companheiros de viagem;

• Numa conferência, onde cada membro da audiência poderá ouvir o áudio no seu idioma, sem recorrer à utilização de headphones;

• Em sistemas de anúncios sonoros nas ruas e praças, que poderão ser substituídos por mensagens áudio direcionadas a cada um dos transeuntes, eliminando a poluição sonora.

Esta tese está organizada em três partes:

1. Teoria e Evolução de sistemas de Audio Spotligh;

2. Implementação do Sistema de Audio Spotlight na FPGA Spartan 3E;

3. Implementação do Sistema de Audio Spotlight no DSP TMS320C6713;

4. Resultados experimentais.

A primeira parte introduz todos os aspetos teóricos referentes à tecnologia, para que seja possível compreender os fenómenos físicos e a matemática associada a um sistema deste tipo. Na segunda parte é explicada a forma como o sistema foi implementado na FPGA Spartan 3E, com a explicação de todos os módulos constituintes da aplicação. A terceira parte é idêntica à segunda, mas desta vez a implementação é realizada no DSP TMS320C6713. Na última parte é feita a demonstração experimental dos conceitos aplicados nesta tese recorrendo à captura de sinal com osciloscópio.

Abstract

This report documentary Master's Thesis with the theme Audio Spotlight - Sound Column using Ultrasound Transducers Arrays conducted by Paul Jaime Estrela Janganga.

This work will be based on the implementation of an Audio Spotlight system, which is a way audio technology. These days, begin to take the first steps in terms of marketing of such products, but there are still some issues that need to be resolved before the technology really come on a day-to-day of all of us. The main problem is the distortion that generates such a system by using nonlinear interaction of high frequency waves (ultrasound) to generate low frequency waves audible to humans.

After solved the problems surrounding the system, the benefits are huge with a list of practical applications and virtually endless commercials:

• In a car each occupant can listen to your own music, without disturbing fellow travelers;

• At a conference, each audience can receive the sound in your language, without the use of headphones in the ears;

• Ad systems on the streets and squares may be replaced by targeted messages to each of the passers-by, eliminating noise pollution.

This document is divided into three distinct parts:

1. Theory and Evolution of Audio Spotlight systems.

2. Implementation of the Audio Spotlight system in Spartan 3E FPGA.

3. Implementation of the Audio Spotlight system in the TMS320C6713 DSP.

The first part will introduce all the theoretical aspects related to technology, so you can understand the physical phenomena and the mathematics associated with such a system.

In the second part it is explained how the system was implemented in FPGA Spartan 3E, with the explanation of all the constituent modules of the application.

The third part is identical to the second, but this time the implementation is done in DSP TMS320C6713.

At the end of the last two parts there is an area reserved for testing in order to view the results, especially through oscilloscope captures.

Acrónimos

SSB	Single Sideband
LSB	Lower Sidebands
USB	Upper Sideband
SC	Supress Carrier
SSB-WC	single-sideband with carrier
DSB	double side band
THC	TotalHarmonic Current
SPL	Sound Pressure Level
ADC	Analog to digital converter
DAC	Digital to analog converter
AM	Amplitude modulation
SFDR	Spurious Free Dynamic Range
VHDl	VHSIC Hardware Description Language
DSP	Digital Signal Processor
DA	Distributed Arithmetic
IP	Intellectual Property
RMS	Root mean square
RAM	Random access memory
MAC	Multiplication and accumulate
FFT	Fast Fourier transform

Índice

Lista de Figuras

Capítulo 1: Introdução

Este trabalho efetua um estudo sobre o método predominante de se alcançar um feixe de som direcional. A primeira parte tem como objetivo introduzir a tecnologia, apesentando os conceitos basilares adjacentes a qualquer sistema que tenha como premissa atingir um elevado grau de direccionalidade. Para além disso também é apresentada a evolução que os diversos conceitos abordados foram sofrendo ao longo dos tempos. O segundo capítulo do trabalho vai incidir essencialmente no estado de arte dos diferentes módulos do sistema de áudio spotlight, com principal enfoque nos trabalhos desenvolvidos por Bertkay e Pompei, pois estes foram os investigadores que mais contribuíram para o estado atual da tecnologia, sendo então estudada a forma como estes dois investigadores fizeram a abordagem teórica e matemática ao problema, tendo especial foco na questão das não-linearidades.

No terceiro capítulo são abordadas as metodologias a seguir para desenvolver o sistema, e paralelamente inicializa-se a abordagem prática ao se descrever a implementação de simulações em Matlab de forma a se perceber o funcionamento e resposta do sistema às diferentes arquiteturas, assim como para se antever a melhor forma de se implementar o sistema na FPGA e no DSP. O quarto capítulo incide na abordagem prática, com descrição pormenorizada das tecnologias utilizadas (FPGA Spartan 3E e DSP TMS320C6713 DSK), assim como das arquiteturas desenvolvidas para realizar os sistemas de Áudio Spotlight, acompanhadas pelos resultados obtidos. Por último entra-se no quinto capítulo que aborda as limitações encontradas no sistema, as futuras investigações e por fim a conclusão.

A dissertação de mestrado pretende abordar a diretividade sonora em várias vertentes testando várias arquiteturas de modulação, e valendo-se ora da banda lateral superior ou da inferior resultante da modulação DSB-WC de forma a comparar a performance do sistema em diferentes casos, também para depurar os problemas inerentes aos sistemas desenvolvidos de forma a encontrar a solução mais viável para os resolver.

Basicamente a diretividade de uma fonte sonora está relacionada com a relação existente entre o comprimento de onda a reproduzir e as dimensões da fonte sonora. Para fontes sonoras mais pequenas, ou do tamanho do comprimento de onda a reproduzir, vamos obter um modo de propagação omnidirecional; contrariamente, para fontes sonoras com um tamanho bastante maior que os comprimentos de onda transmitidos por esta, vamos ter o som a propagar-se com um elevado grau de direccionalidade. As ondas sonoras estão compreendidas entre os 20Hz e os 20Khz e têm comprimentos de onda que vão desde os 17,2 milímetros até aos 17,2 metros assumindo a velocidade de propagação igual a 344 metros por segundo e tendo em conta a seguinte formula:

$$\lambda = \frac{c}{f}$$

λ = Comprimento de onda da onda sonora.

c = velocidade de propagação das ondas sonoras no ar.

f = frequência da onda sonora.

Tendo em conta estes pressupostos, as fontes sonoras que se encontrem dentro destas dimensões vão ter um comportamento omnidirecional. A ideia adjacente ao som direcional é ao invés de transmitir o som na faixa audível, passar a transmitir a informação sonora em ultrassons, com comprimentos de onda centrado nos milímetros de forma a obter comprimentos de onda bastante mais pequenos que as fontes sonoras para que se consigam criar feixes de som estreitos o suficiente para alcançar o pressuposto referido anteriormente de ter a fonte sonora com uma dimensão superior aos comprimentos de onda emitidos de forma a se alcançar um feixe de som direcional.

Os ultrassons são completamente inaudíveis ao ouvido humano, mas as tecnologias de som direcional baseiam-se nas não-linearidades presentes no meio de propagação para que se consiga converter os ultrassons em som audível. No ar as não-linearidades estão presentes na forma de uma perturbação da velocidade do som em função da pressão e densidade do ar.

As não-linearidades podem ser expressas na forma de séries de Taylor[4], pelo que se pode considerar o primeiro termo não linear como a relação com a função quadrática:

$$y = \left(\sum_{i}^{\infty} a_i \sin \omega_i\right)^2 \quad (1)$$

Partindo do pressuposto que uma coleção de ondas sinusoidais é elevada ao quadrado, demonstra-se que através das não linearidades é criada uma onda constituída pelas soma e diferença das frequências originais.

$$y = \sum b_{i,j} \left(\sin(\omega_i + \omega_j) + \sin(\omega_i - \omega_j)\right) \quad (2)$$

Caso as frequências $\omega_i \; e \; \omega_j$ estejam presentes na zona dos ultrassons, os termos $\omega_i + \omega_j$ podem ser ignorados, pois resultam em frequências ultrassónicas adicionais.

Já os termos $\omega_i - \omega_j$, caso tenham sido escolhidos corretamente vão cair dentro da gama audível. As não-linearidade causam uma transferência de energia das frequências ultrassónicas para as frequências audíveis. O comprimento do feixe de som é limitado pela absorção dos ultrassons pelo ar, podendo atingir vários metros de comprimentos. Regra geral alcança uma distância bem maior do que os feixes produzidos pelos sistemas de som normais, que reproduzem frequências na gama do som audível.

1.1 - Histórico

A história do som direcionado inicializa-se a meio do século 19 quando Helmholtz introduz o conceito de geração de tons adicionais através da combinação de dois ou mais tons. Este trabalho foi desenvolvido recorrendo ao uso de tons gerados em violinos. O estudo na altura teve um grande impacto no campo da acústica, e deu iniciou a novas áreas de investigação, mas carecia ainda de algum formalismo físico e matemático de forma a dar suporte a um campo com tanta especificidade como as não-linearidades.

Foi então em meados do século vinte que se deu uma nova reviravolta neste campo científico, pois apareceram novas investigações, especialmente ao nível do desenvolvimento de sonares que vieram impulsionar a área. Posteriormente à paragem que se seguiu aos estudos de Helmholtz's, a tecnologia não-linear começou novamente a ser abordada a partir de meados de 1960. Inicialmente começou por ser estudada pela Marinha dos Estados Unidos e da União Soviética de forma a aperfeiçoar o alcance e a precisão dos sonares. É sabido que grande parte das tecnologias conhecidas pelo mundo atual tiveram os seus primeiros passos na investigação dentro do mundo militar. Já por volta de 1980 foram os japoneses que trouxeram alguns avanços no desenvolvimento do sistema não-linear, mas este apresentava na altura algumas limitações, especialmente referentes à qualidade do som, associada a um nível de distorção bastante elevado, e para além disso o sistema apresentava um elevado custo de desenvolvimento, fabrico e manutenção. Estes problemas ficaram por resolver até 1988, ano em que o Dr. F. Joseph Pompei do MIT lançou uma publicação que descrevia pormenorizadamente uma forma eficaz de reduzir a distorção, fazendo com que o sistema se tornasse bastante mais atrativo. A metodologia desenvolvida por Pompei perdura até hoje como o método predominante para se desenvolver sistemas sonoros diretivos.

1.2 -Cronologia de desenvolvimento

1850

Os princípios básicos das operações não lineares começaram a ser delineados no século 19, data em que Helmholtz's lançou uma publicação de nome "Combination tones in violins" [23], onde realizou experimentos com a combinação de tons. Este cientista introduziu ao mundo a teoria descrevendo a geração de não linearidades através da combinação de tons, mostrando que um par de tons gerados por uma fonte comum podem gerar um par extra de tons no ar, com frequências correspondentes à soma e à diferença das frequências dos tons iniciais. Porém, apesar das experiências de Helmholtz's provarem a presença de tons adicionais no ar correspondentes à subtração e adição de um tom sobre o outro, a sua teoria não era sustentável a nível científico, e foram necessários muitos anos até começarem a ser encontradas soluções viáveis que introduzissem o mundo das não linearidades à comunidade científica.

Dezembro de 1962

O Professor de física Peter Westervelt da Universidade de Brown publicou uma investigação de nome "Parametric Acoustic Array"[21], onde o professor considerava a interação de ondas sinusoidais num determinado volume, calculando a pressão devido às não-linearidades dentro de uma pequena porção do volume. No estudo é referido a geração de um tom adicional ao longo do feixe constituído por um sinal de dois tons.

Foram tidos em conta vários pressupostos de forma a simplificar o trabalho, tais como:

1. O Tom gerado vai ter exatamente a mesma amplitude dos dois tons iniciais.
2. Um feixe colimado primário perfeito.
3. As duas sinusoides atenuam de forma idêntica ao mesmo sample rate.
4. Etc.

Este trabalho não apresentava a complexidade dos últimos desenvolvimentos, mas já apresentava um elevado nível de precisão, o que proporcionou de facto uma framework de desenvolvimento para os próximos investigadores refinarem o seu trabalho.

Outubro de 1965

H.O. Berktay publica um estudo de nome "*Possible Exploitation of Non-Linear Acoustics in Underwater Transmitting Applications*", que permite ter uma visão teórica bastante mais assertiva e conclusiva em relação aos arrays paramétricos acústicos.

Este trabalho foi de facto mais geral e completo pois explora vários casos em que por exemplo os sinais primários são cilíndricos ou esféricos. Para além disso abordou os planos das ondas colimadas em observação. Bertkay também não se limitou somente aos estudos de casos envolvendo sinais de dois tons, pois introduziu o conceito de envolvente, que permitiu que o estudo fosse aproximado da realidade, pois os altifalantes paramétricos não estão limitados a gerar somente um ou dois tons de cada vez.

Capítulo 2: Revisão de Literatura

Este capítulo foi elaborado de forma a se compreender o estudo teórico por trás dos sistemas de Áudio Spotlight. Nele são abordados de forma mais aprofundada, os estudos de Bertkay e Pompei [4], que presentemente ilustram o estado de arte da tecnologia.

2.1 - Base teórica da acústica não linear de ultrassons

As equações matemáticas que caracterizam a acústica não linear são por si só bastante complexas, e infelizmente não existe uma solução analítica geral. Normalmente recorre-se a uma simulação em computador para se tentar aproximar de situações realistas. No entanto, em 1965 Bertkay realizou uma análise que permitiu simplificar algumas das premissas tendo como resultado uma solução que tornou possível expressar o nível de pressão sonora (SPL) do sinal desmodulado em termos da pressão sonora da portadora ultrassónica modulada em amplitude [5]. De notar que o processo de desmodulação é extremamente danoso em termos de perdas, sendo que o mínimo previsto é de 60 dBs ao se passar do SPL ultrassónico para o SPL da onda acústica audível. O grande avanço de Berktay foi o facto deste conseguir antecipar um esquema de pré-compensação derivado da sua expressão, ao efetuar a raiz quadrada da envolvente do sinal em banda base (E) e seguidamente ao integrar o resultado duas vezes, de forma inverter o efeito da segunda derivada de tempo parcial[11]. Para conseguir atingir este resultado era usado um circuito analógico equivalente, em que um ampop com feedback realizava a operação de raiz quadrada, enquanto um equalizador era o circuito análogo a uma operação de integração. De seguida temos a expressão de Berktay's:

$$P_2(\mathrm{x},\mathrm{t}) = \mathrm{K}.\,P_c^2.\frac{\partial^2}{\partial t^2}E^2(x,t) \tag{3}$$

Onde

$P_2(\mathrm{x},\mathrm{t}) = Pressão\ sonora\ da\ onda\ secundária\ audivel$

$K = Variável\ que\ comporta\ vários\ aspectos\ fisicos\ do\ sistema$

$P_c = SPL\ da\ portadora\ ultrasónica$

$E(\mathrm{x},\mathrm{t}) = Função\ Envolvente\ (DSB - AM)$

A equação diz-nos que a pressão sonora da onda ultrassónica desmodulada audível, ou seja, o sinal audível à saída é proporcional à raiz quadrada da segunda derivada da função da envolvente, ou seja, do sinal de entrada. O sistema de pré-compensação consiste em antecipar as operações do sinal de entrada, e aplicar as operações inversas ao mesmo, assumindo que o sinal de saída vai ser parecido com o sinal de entrada depois de aplicadas as operações inversas. Em meados de 1990 a tecnologia de áudio spotlight já conseguia operar, mas ainda com um elevado nível de distorção. Por esta altura os esquemas de pré-compensação fizeram com que se procurasse otimizar a resposta em frequência dos transdutores ultrassónicos [4]. A pesar a este facto os transdutores também precisaram de ter uma resposta de frequência mais ampla de forma a acompanhar os processos de pré-compensação digitais. Assim, em 1998 foi efetuado um estudo em simulação de computador de forma a quantificar a ineficiência da resposta em frequência dos transdutores usando o esquema de pré-compensação da expressão de Berktay's.

Em 1990 um artigo escrito por Pompei descreveu um novo protótipo de transdutor capaz de atender às crescentes exigências de transdutores com uma resposta em frequência mais ampla e dinâmica, uma vez mais baseado na expressão de Bertkay's. Finalmente a tecnologia que originalmente tinha sido desenhada para operar como sonar debaixo de água tinha alcançado um patamar de qualidade praticamente idêntico ao dos sistemas de som normais, graças ao sistema desenvolvido por Pompei.

2.2 - Base matemática

Comparativamente, pode afirmar-se com toda a certeza que existe muito menos estudos científicos realizados para as matrizes paramétricas ao ar livre, do que para as mesmas em ambiente subaquático. Também se pode afirmar que as aplicações em meio subaquático são bastante diferentes das aplicações ao ar livre, mas estas proporcionam uma framework teórica bastante sólida, a partir da qual se pode desenvolver um trabalho conciso e objetivo para a propagação não linear em espaço aberto. As equações que exprimem a propagação não linear de ondas acústicas são bastante similares com a equação geral da propagação de ondas, excetuando do facto de nestes casos não se descartar os termos não-lineares.

2.2.1- Equação padrão

Nesta equação é estabelecida a relação entre a pressão P, a densidade (ρ) e a entropia (s). Para um fluido em geral, a pressão em função da densidade e entropia é não linear, pelo que pode ser definida numa série de Taylor num ambiente isotrópico de pressão ambiente (P0).

$$P = P(\rho, s) \tag{4}$$

$$P = P_0 + \left(\frac{\partial P}{\partial \rho}\right)_{s,0} (\rho - \rho_0) + \frac{1}{2!}\left(\frac{\partial^2 P}{\partial \rho^2}\right)_{s,0} (\rho - \rho_0)^2 + \cdots \tag{5}$$

A pressão e a densidade caracterizam-se por:

$$p = P - P_0 \tag{6}$$

$$\rho' = \rho - \rho_0 \tag{7}$$

Fazendo a substituição:

$$p = A\left(\frac{\rho'}{\rho_0}\right) + \frac{B}{2!}\left(\frac{\rho'}{\rho_0}\right)^2 + \frac{C}{3!}\left(\frac{\rho'}{\rho_0}\right)^3 + \cdots, \tag{8}$$

Onde:

$$A = \left(\frac{\partial P}{\partial \rho}\right)_{s,0} \tag{9}$$

$$B = \partial_0^2 \left(\frac{\partial^2 P}{\partial \rho^2}\right)_{s,0} \tag{10}$$

$$C = \partial_0^3 \left(\frac{\partial^3 P}{\partial \rho^3}\right)_{s,0} \tag{11}$$

A velocidade do som num fluido isotrópico é definida da seguinte forma:

$$\begin{aligned} & c^2 \left(\frac{\partial P}{\partial \rho}\right)_s \\ &= \left(\frac{\partial P}{\partial \rho}\right)_{s,0} + \left(\frac{\partial^2 P}{\partial \rho^2}\right)_{s,0} (\rho - \rho_0) + \frac{1}{2!}\left(\frac{\partial^3 P}{\partial \rho^3}\right)_{s,0} (\rho - \rho_0)^2 + \cdots \\ &= c_0^2 + B\frac{\rho'}{\rho_0^2} + \frac{C}{2}\frac{{\rho'}^2}{\rho_0^3} \end{aligned} \tag{12}$$

Depois de efetuar a raiz quadrada, a expansão binominal permite achar uma solução para c:

$$\frac{c}{c_0} = 1 + \frac{B}{2}\left(\frac{\rho'}{\rho_0}\right) + \frac{1}{4}\left[\frac{C}{A} - \frac{1}{2}\left(\frac{B}{A}\right)^2\right]\left(\frac{\rho'}{\rho_0}\right)^2 + \cdots \tag{13}$$

Onde c_0 é a velocidade de uma onda no ambiente, e é igual a $\sqrt{A/\rho_0}$

Caso a densidade seja assumida como pequena perante a densidade do ar, os termos contendo os termos quadráticos superiores a $\left(\frac{\rho'}{\rho_0}\right)$ podem ser omitidos. Para uma onda plana esta aproximação permite afirmar que $u \ll c_0$, em que u é a velocidade da partícula, ou equivalentemente $p \ll \rho_0 c_0^2 \approx 197 dB\ SPL$. Pelo que para um gás diatómico isotrópico como o ar a equação de estado pode ser escrita diretamente como:

$$p = P_0 \left(\frac{\rho}{\rho_0}\right)^{\gamma} \tag{14}$$

Em que $\gamma = \frac{c_p}{c_v}$ é a razão de diferentes temperaturas. No ar $\gamma = 1{,}4$, logo

$$\frac{c}{c_0} = 1 + \frac{\gamma - 1}{2}\left(\frac{\rho'}{\rho_0}\right) \tag{15}$$

2.2.2-Equação de Continuidade

Para um cubo de ar estacionário de volume dV, área de face dA e comprimento de aresta dx, a taxa de massa de ar que flui para dentro do cubo ao longo do eixo x, menos a taxa de massa de ar que flui para fora é igual à taxa de aumento da massa de ar dentro do volume:

$$dV\frac{\partial \rho}{\partial t} = \rho u dA - \rho u|_{x+dx} dA \tag{16}$$

$$= \left[\rho u - \rho u - \frac{\partial(\rho u)}{\partial x} dx\right] dA$$

$$= \frac{\partial(\rho u)}{\partial x} dV$$

Combinando as três dimensões espaciais e cancelando dV têm-se o seguinte resultado:

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \vec{u}) = 0 \tag{17}$$

2.2.3 - Equação de Euler

Considerando o volume do ar dV, de massa dm, é sabido a partir da segunda equação de Newton $F = ma$

Após alguma manipulação matemática obtém-se:

$d\vec{f} = \vec{a}dm = dP d\vec{A}$

- A pressão no lado esquerdo do volume é de P.
- A pressão no lado direito do volume é de $P + \frac{\partial \rho}{\partial x} dx$

Fazendo com que a força resultante seja caracterizada pela seguinte expressão:

$$d\vec{f} = -\frac{\partial \rho}{\partial x} dV \tag{18}$$

A expressão a três dimensões toma a seguinte forma:

$$d\vec{f} = -\nabla \mathrm{P dV} \tag{19}$$

Já a aceleração do volume de ar pode ser escrita da seguinte forma:

$$\vec{a} = \frac{d\vec{u}}{dt} = \lim_{dt \to 0} \frac{\vec{u}_1 - \vec{u}_0}{dt} \tag{20}$$

Onde:

$$\vec{u}_1 = \vec{u}_0 + \frac{\partial \vec{u}}{\partial x} u_x dt + \frac{\partial \vec{u}}{\partial y} u_y dt + \frac{\partial \vec{u}}{\partial z} u_z dt + \frac{\partial \vec{u}}{\partial t} dt \quad (21)$$

Em que:

$$\frac{\partial \vec{u}}{\partial x} u_x dt + \frac{\partial \vec{u}}{\partial y} u_y dt + \frac{\partial \vec{u}}{\partial z} u_z dt = (\vec{u}.\nabla)\vec{u} dt \quad (22)$$

Pelo que a aceleração da massa de ar corresponde a:

$$\vec{a} = \frac{\partial \vec{u}}{\partial x} + (\vec{u}.\nabla)\vec{u}$$

Que substituindo pela equação de força dá:

$$d\vec{f} = \vec{a} dm = \vec{a} \rho dV$$

E a equação de Euler transforma-se em:

$$-\nabla \mathrm{p} = \rho \left[\frac{\partial \vec{u}}{\partial x} + (\vec{u}.\nabla)\vec{u} \right] \quad (23)$$

2.2.4-Combinando as equações

A este ponto da derivação da equação linear de onda os termos não lineares como $(\vec{u}.\nabla)\vec{u}$ são também eles linearizados e a equação final é simplificada para:

$$\nabla^2 p = \frac{1}{c_0^2} \frac{\partial^2 \mathrm{p}}{\partial t^2} \quad (24)$$

Como os termos não lineares são necessários, e a não linearidade é fraca, é necessário um método perturbativo de forma a nos aproximarmos de uma solução realista. Caso as perturbações no som tenha um μ pequeno,

$$\frac{\rho'}{\rho_0} \sim \frac{\mathrm{p}'}{\mathrm{p}_0} \sim \frac{u}{\mathrm{c}_0} \sim \mu \quad (25)$$

Ao serem consideradas as ondas ao longo do eixo dos x's, pode ser escrita uma função de propagação arbitrária F em termos de $t - \frac{x}{\mathrm{c}_0}$

$$\rho', u_x, p = F\left(t - \frac{x}{\mathrm{c}_0}\right) \quad (26)$$

A forma da onda é alterada pelas não linearidades enquanto percorre o meio ambiente, em ambas as direções, transversal e axial. Existe uma relação direta da alteração da forma de onda com a difração e as não linearidades:

- Transformações axiais: Não linearidades.
- Transformações transversais: Difração.

Caso seja assumido que as alterações transversais são mais acentuadas que as axiais a solução da equação de onda assume a seguinte forma:

$$\rho', u_x, p' = \varphi\left(t - \frac{x}{c_0}, \mu x, \sqrt{\mu} y\right) \tag{27}$$

Os argumentos podem ser definidos da seguinte forma:

$$\tau = t - \frac{x}{c_0}, x' = ux, y' = \sqrt{\mu} y \tag{28}$$

Ao se efetuar as definições acima indicadas as derivadas parciais são automaticamente simplificadas para:

$$\frac{\partial \rho'}{\partial x'} \sim \frac{\partial u_x}{\partial x'} \sim \frac{\partial p'}{\partial x'} \sim \mu^2, \frac{\partial \rho'}{\partial y'} \sim \frac{\partial u_y}{\partial y'} \sim \frac{\partial p'}{\partial y'} \sim \mu^{3/2}, \mu^y \sim \frac{\partial u_x}{\partial y} \sim \mu^{3/2}$$

Quando as últimas equações são combinadas, os elementos de menor ordem $O(\sqrt{\mu}), O(\mu), O(\mu^2)$

Podem ser combinados, e os de ordem mais elevada podem ser descartados. Depois de muita manipulação nas equações chega-se a uma equação final, de nome KZK (Khokhlov, Zabolotskaya, e Kuznetsov)[18].

$$\frac{\partial^2 p}{\partial z \partial \tau} = \frac{c_0}{2} \nabla_r^2 p + \frac{\delta}{2c_0^3} \frac{\partial^3 p}{\partial \tau^3} + \frac{\beta}{2\rho_0 c_0^3} \frac{\partial^2 p^2}{\partial \tau^2} \tag{29}$$

Esta equação é uma boa aproximação para feixes de som direcionais ao longo do eixo.

2.3 - Modulação e desmodulação de feixes de ultrassons direcionáveis para a transmissão e renderização de áudio

A modulação em amplitude teve sempre um papel vital no estudo dos parametric arrays. Dentro dos esquemas de modulação em amplitude, o DSB [3] é um dos mais simples de perceber e implementar.

Este tipo de modulação é alcançando multiplicando uma portadora sinusoidal pelo sinal que contém a informação, que no nosso caso é um sinal áudio. O seu espectro é composto por duas bandas laterais, simétricas em relação à frequência da portadora. A envolvente da modulação AM é o sinal de áudio propriamente dito, sendo que o recetor do sinal modelado em AM é simplesmente um detetor de envolvente. Neste ponto podem-se inferir dois pontos:

Primeiro que tudo é aceitável usar o trabalho de Berktay's baseado na envolvente do sinal de forma a ter um modelo básico para calcular a pressão, pois no AM a envolvente é a diferença de pressão.

Em segundo lugar, e mais importante é o facto das interações não lineares acontecerem entre ambos os componentes de frequências do esquema de modulação AM causando distorção.

Neste tipo de modulação estão presentes dois tipos de distorção.

Distorção intra-sideband causada pelos efeitos da subtração dos diferentes componentes em frequência.

Distorção inter-sideband causada pela interação entre as componentes presentes em ambas as bandas laterais.

Neste último caso as duas bandas laterais interagem uma com a outra causando os efeitos indesejáveis da distorção. Este tipo de distorção pode ser reduzida significativamente. Kite et al propôs um algoritmo de pré-processamento que reduz de forma viável a distorção inter-sideband, e é o que é usado na maioria das aplicações em uso hoje em dia. Este algoritmo é baseado no trabalho de Bertkay's, sendo que o que faz é aplicar a operação oposta presente no modelo de Bertkay's.

De acordo com o modelo, uma onda colimada é consistida pela seguinte pressão de uma onda modelada em amplitude:

$$p_1(t) = P_1 E(t)\sin(\omega_c) \tag{30}$$

Em que:

P_1 : Amplitude da portadora

$E(t)$:A envolvente modulante

ω_c : A frequência angular da portadora

Consequentemente o sinal depois de desmodulado através do fenómeno da não linearidade vai ser transformado num som audível com a seguinte expressão:

$$p_d(t) = \frac{\beta P_1^2 A}{16\pi\rho_0 c_0^4 z\alpha}\frac{\partial^2}{\partial t^2}E^2(\tau) \tag{31}$$

Em que:

$\beta = (\gamma + 1)$: Coeficiente de não linearidade β_{ar}=1,2

γ : Razão de temperaturas

A : Área de radiação do transdutor

ρ_0 : Densidade do ar

c_0 : Velocidade do sinal a transmitir

z : Distância axial

α : Coeficiente de absorção do ar para a portadora

A expressão mostra a dependência da pressão do sinal de áudio desmodulado com a envolvente do sinal modelado em amplitude. A envolvente é por definição um sinal em banda de base. Esta expressão apresentada por Pompei é uma adaptação do modelo para colunas em parametric arrays de Berktay's, onde o sinal modelado é um sinal de áudio. Mais uma vez é referido o trabalho de Pompei, pois este foi um dos poucos e primeiros produtos comerciais a ser

desenvolvido. O modelo baseia-se no estudo de Bertkay's, na modulação AM e no algoritmo de Kite's. O algoritmo de pré-processamento de Kite's aplica um integral duplo de forma a compensar a derivada de segundo grau e uma raiz quadrada para compensar as não linearidades quadráticas no modelo de Bertkay's. A portadora é multiplicada pelo sinal de áudio pré-processado. Ao serem aplicados sequencialmente, o algoritmo de Kite's e a solução de Bertkay's permite que na saída esteja presente uma boa atenuação na banda lateral inferior do espectro AM devido à dupla derivação e dupla integração. Pelo que o pré-processamento de Kite's reduz significativamente a distorção inter-sideband.

Pensando neste modelo salta á vista que a melhor solução a adotar de forma a reduzir a distorção inter-sideband é aplicar o esquema de modulação single side band (SSB), que tem como característica principal o facto de o seu espectro apresentar somente uma banda lateral. Aqui pode ser escolhida tanto a Lower Sideband (LSB) como a Upper Sideband (USB). O padrão do SSB usado em telecomunicações é supress carrier (SC), mas no caso das aplicações por parametric array é necessário usar a portadora para que seja possível gerar as diferenças nas frequências no sinal de áudio. Este tipo de modulação é designada por sinal SSB-WC (single-sideband with carrier) que significa que é adicionada a portadora por via acústica ou elétrica, depois de ser efetuada a modulação SSB padrão. Ao se ter somente uma banda lateral, não é necessário ter em conta a distorção inter-baseband, pelo que doravante não é necessário recorrer ao pré-processamento de Kite's. Somando ao facto que este modelo de modulação AM permite uma maior versatilidade dado ao facto que se pode escolher uma das duas bandas laterais atendendo às características do equipamento, e á resposta em frequência do transdutor, assim como à absorção do canal de transmissão. Uma vantagem visível deste modelo é também o facto de este usar de forma mais eficiente a largura de banda disponível dado somente usar-se uma banda lateral.

2.4 - Distorção em sistemas de ultrassons

Como já foi mencionado anteriormente, a distorção é uma variável muito presente nos sistemas de ultrassons. No capítulo anterior, que aborda os processos de modulação e desmodulação, foi referido o facto das interações não lineares acontecerem entre ambos os componentes de frequências do esquema de modulação AM causando dois tipos de distorção.

Distorção intra-sideband causada pelos efeitos da subtração dos diferentes componentes em frequência.

Distorção inter-sideband causada pela interação entre as componentes presentes em ambas as bandas laterais.

Para além da distorção causada pela modulação, um dos desafios mais preeminentes das operações envolvendo operações quadráticas é o facto que estas introduzem harmónicos no sinal, aumentando a largura de banda exponencialmente como nos mostra a figura que se segue. Quando o sinal é modelado com ultrassons o sistema de reprodução tem de reproduzir de forma fiável a largura de banda de todo o sinal da forma mais precisa possível, pelo que limitações na largura de banda, ou mesmo não uniformidades na resposta em frequência ultrassónica vão levar a um aumento na distorção.

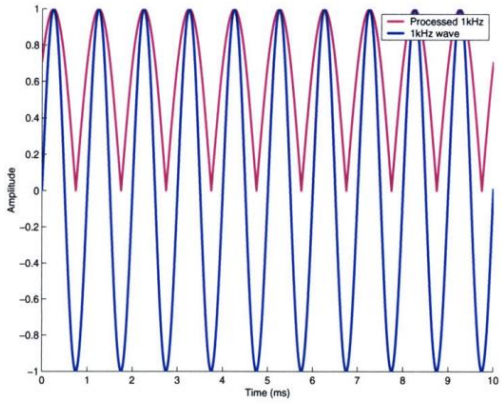


Figura 1: A Azul temos o sinal antes de ser processado e a lilás o sinal pós processado.

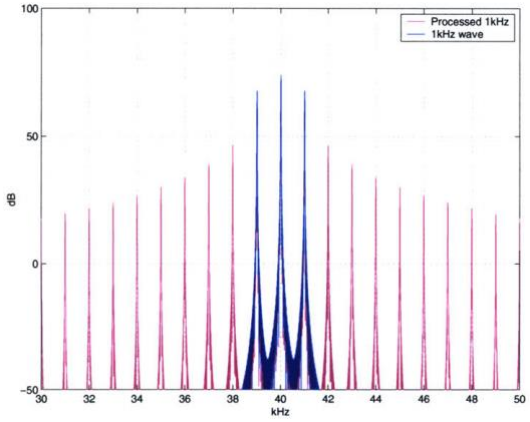


Figura 2: Azul - Sinal de entrada Lilás - Sinal processado

Para além dos fatores geradores de ruído, mencionados anteriormente, há que ter em conta a atenuação associada a frequências elevadas e as características do ambiente que contribuem significativamente para a redução da potência efetiva da propagação e reprodução do sinal.

O figura que se segue ilustra a relação existente entre a frequência do som e a atenuação num ambiente a 20 graus Celsius. Como se pode verificar, quanto mais elevada a frequência, maior a atenuação, o que faz com que se tenha de ter este parâmetro em conta. Para além disso a humidade também vai influenciar a propagação do som, como se pode verificar. Em termos práticos, verifica-se que para uma frequência de 50Khz, e com uma humidade relativa de 40%, a 1 atmosfera de pressão, temos uma atenuação de 1dB a cada metro, o que por si só já é um valor bastante significativo, tendo em conta que o gráfico não conta com a atenuação provocada pela desmodulação do sinal de ultrassons para som audivel.

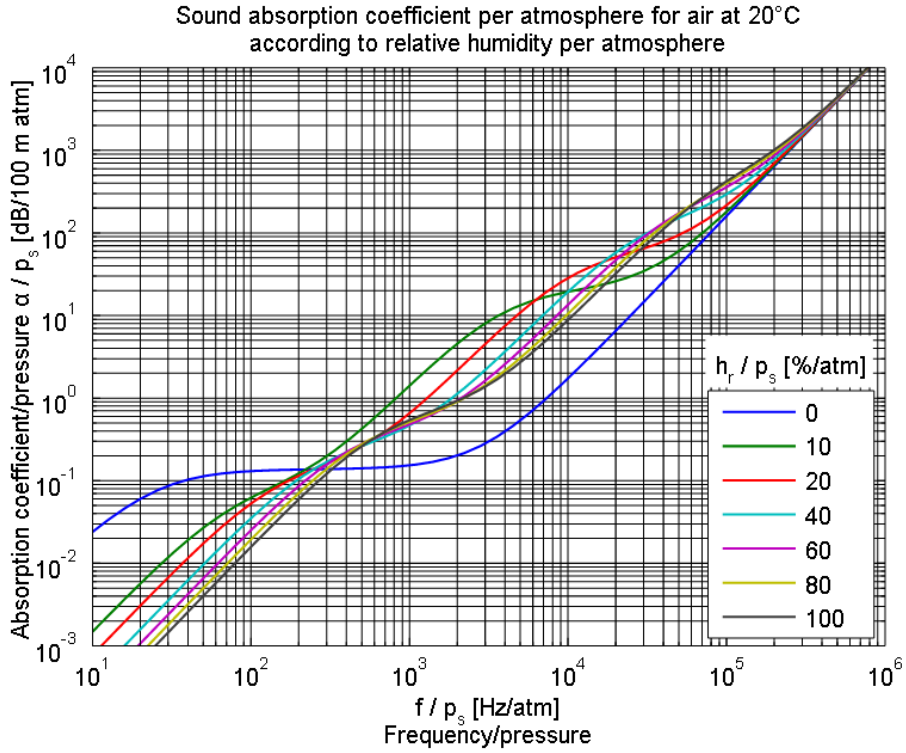


Figura 3:Atenuação do sinal ultrassónico

2.5 - Metodologia de redução da distorção do trabalho prático

Assumindo o campo distante, foi mostrado que:

$$p_{2(x,0,\tau)} = \propto \frac{d^2}{d\tau^2} E^2(\tau) \tag{32}$$

Assim sendo, para reproduzir um sinal de áudio g(t), a solução mais simples é apenas efetuar o integral duplo e a raiz quadrada[11] antes de modular:

$$E(t) = \left(1 + m \iint g(t) dt^2\right)^{\frac{1}{2}} \tag{33}$$

O fator m corresponde à modulação. Caso o sinal modulado $E(t) sin w_c t$ seja transmitido precisamente, o sinal desmodulado vai ser um sinal áudio livre de distorção, mas de facto, transmitir um sinal áudio modelado com ultrassons não é uma tarefa simples para sistemas tradicionais, em particular dado ao facto da largura de banda disponibilizada pelos transdutores ser limitada, o que pode levar a níveis de distorção extremamente elevados.

Capítulo 3: Metodologia e Dados

Neste capítulo vão ser abordados os métodos e processos de trabalho usados na parte prática da dissertação.

3.1 - Metodologia processual

O processo de desenvolvimento que foi seguido está descrito no modelo que se segue.

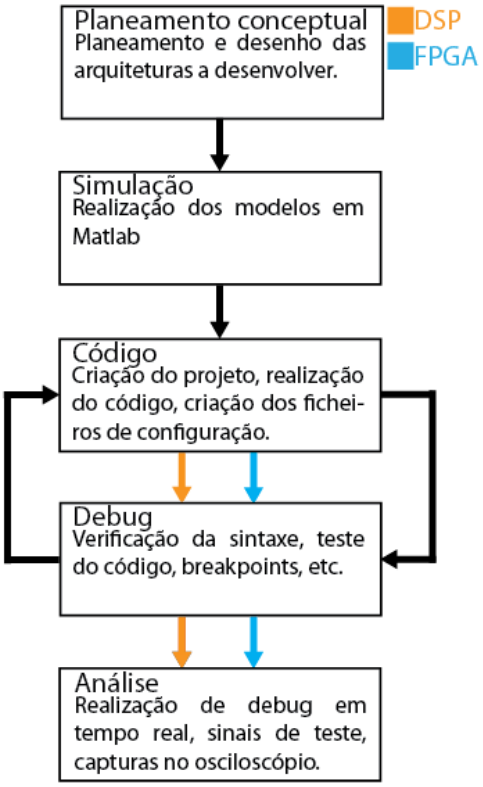


Figura 4: Processo de desenvolvimentos dos sistemas.

O planeamento conceptual vai consistir na definição das arquiteturas a ser implementadas. Este ponto vai ser decisivo na escolha das parametrizações a seguir. Eventualmente e dada a forma como os problemas técnicos foram surgindo, foram havendo ligeiras alterações na forma como as arquiteturas foram sendo seguidas. Um exemplo é que inicialmente na implementação no DSP tanto na primeira como na segunda arquitetura eram para ser realizados testes sobre a banda lateral inferior e superior, mas dado ao facto da banda lateral superior ter mais ruído e a banda reduzida (8kHz), somente foram avaliadas as bandas inferiores. Seguidamente passa-se à simulação dos modelos em Matlab. A partir deste ponto o modelo vai ser repetido por duas vezes; uma para o DSP e outra para a FPGA como demonstra o diagrama de blocos. Os dois passos seguintes na prática acabam por ser um só, pois à medida que se foi desenvolvendo o código, este foi sempre testado em modo de debug, de forma a estudar a forma mais eficiente de organizar sua a estrutura.

Por fim foram realizados testes de forma a inferir conclusões acerca das opções tomadas. Os sinais de teste usados consistiram em sinusoides, sinais swept sine e ruído branco filtrado passa-baixo com uma frequência de corte de 8Khz e sem filtragem.

3.2 - Planeamento conceptual

A primeira arquitetura a ser implementada consiste num filtro passa-baixo de 8Khz de forma a somente passarem a frequências correspondentes à fala. Seguidamente o sinal é modelado por uma portadora de 40Khz de forma a 'puxar' o sinal para a gama de ultrassons. Após a modulação o sinal sofre novamente uma filtragem, mas desta vez passa-banda com frequências de corte iguais a:

- Fc1: 32Khz
- Fc2: 40Khz

Com esta segunda filtragem fica-se somente com a banda lateral inferior da modulação DSB (double-side-band). Depois deste passo, e dado ao facto que este sistema implementa uma modulação DSB-WC (double-side-band with carrier) falta somente adicionar a portadora ao sinal filtrado. A portadora deve, por norma, ter mais 10dB de potência que o sinal filtrado. A próxima imagem ilustra o diagrama de blocos deste sistema.

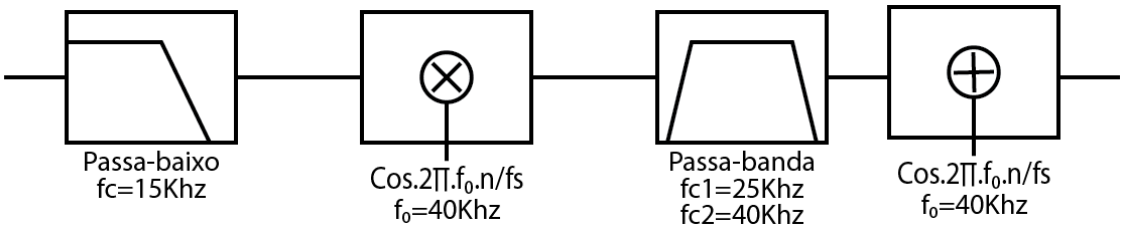


Figura 5: Primeira arquitetura - Sistema DSB-WC

A segunda arquitetura do sistema vai consistir em um filtro passa-baixo que vai ter como frequência de corte 8Khz de forma a só passarem as frequências correspondentes à fala, depois o sinal vai tomar dois caminhos distintos, em que num deles vai passar pelo filtro de Hilbert, e no outro vai ser atrasado N vezes correspondentes a metade do número de coeficientes do filtro de Hilbert de forma a sincronizar as amostras. Seguidamente o sinal no troço onde sofre o atraso é multiplicado por uma portadora com um cosseno a 40Khz, enquanto no troço onde sofre a transformação de Hilbert, o sinal é multiplicado por uma portadora com um seno também ele a 40Khz. Depois de ter ambos os troços concluídos, ambos podem sofrer uma, das duas operações:

- Soma: Caso se queira ter a banda superior.
- Subtração: Caso se queira ter a banda inferior.

Finalmente, o sinal já somado, ou subtraído vai sofrer uma filtragem passa-banda, em que vão ser retiradas as frequências compreendidas entre os 40Khz e os 48Khz.

A imagem que se segue ilustra a arquitetura do sistema num diagrama de blocos de forma a simplificar a compreensão do sistema.

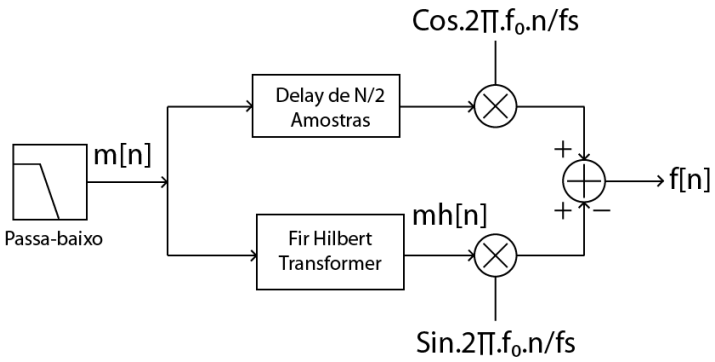


Figura 6: Segunda arquitetura SSB.

3.3 - Implementação do sistema de ultrassons em Matlab

A partir deste momento começa a descrição da execução técnica do trabalho. O primeiro ponto é implementar as várias arquiteturas em Matlab, de forma a se visualizar a resposta do sistema. Na implementação do scripts de simulação do sistema de áudio spotligth da segunda arquitetura, que envolve a filtragem de Hilbert, foi evitado o uso de objetos disponibilizados no Matlab (do tipo filter por exemplo), dado ao facto de se querer aproximar o código do script o máximo possível com o código a implementar nos sistemas em ambas as plataformas, especialmente no DSP.

3.4.1 - Arquitetura 1 – SSB – WC (Filtro passa-banda)

De forma a testar o sistema de áudio spotlight, inicialmente foram criados dois scripts em Matlab para decidir de forma antecipada quais os melhores caminhos a seguir para as duas arquiteturas. Primeiro foi implementada a arquitetura DSB-WC e por fim a arquitetura SSB-WC. Esta foi também uma forma de ter um primeiro contacto com os arrays parametric. O código da aplicação foi o seguinte:

```
format long;
clear all;
close all;
A1 = 2; f1 = 2000; p1 = 0;     % Amplitude, Frequência e fase do sinal
modulante
A2 = 4; f2 = 40000; p2 = 0;    % Amplitude, Frequência e fase da
Portadora
Fs = 128000;                   % Frequência de amostragem
t = 0: 1/250000 : 1;           % Janela temporal.

sinalAudio = A1*sin(2*pi*f1*t + p1); % Sinal modulante
portadora = A2*sin(2*pi*f2*t + p2);  % Portadora

sinalModelado = sinalAudio.*portadora;              % DSB-SC
sinalModeladoPortadora = sinalModelado + portadora;  % DSB-WC
%-----------------------------------------------------
order=63;            % Ordem
Fn=Fs/2;             % Frequência de Nyquist
wl=32000/Fn;         % Frequência de corte inferior normalizada
wu=40000/Fn;         % Frequência de corte Superior normalizada
wc=[wl,wu];          % Vetor das frequências de corte
b=fir1(order,wc);    % coeficientes do filtro
```

```
sinalModeladoPortadoraFiltrado = filter(b,1,sinalModeladoPortadora);
%--------------------------------------------------
ultrassons = audioplayer(sinalModeladoPortadoraFiltrado,Fs);
play(ultrassons)
%--------------------------------------------------
```

O resultado da implementação foi a imagem que se segue. A captura foi dividida e quatro frames distintas de forma a compreender os processos envolvidos na simulação. Na primeira imagem temos a modulação DSB-SC, na segunda imagem foi adicionada a portadora de forma a obter a modulação SSB-WC, necessária para a implementação deste sistema.

A imagem seguinte ilustra o filtro passa-banda realizado, de forma a visualizar as suas frequências de corte. Nesta primeira simulação as frequências de corte usadas foram as correspondentes à banda lateral inferior compreendidas entre:

- Fc1: 32Khz
- Fc2:40Khz

Por fim temos a representação do sinal já filtrado.

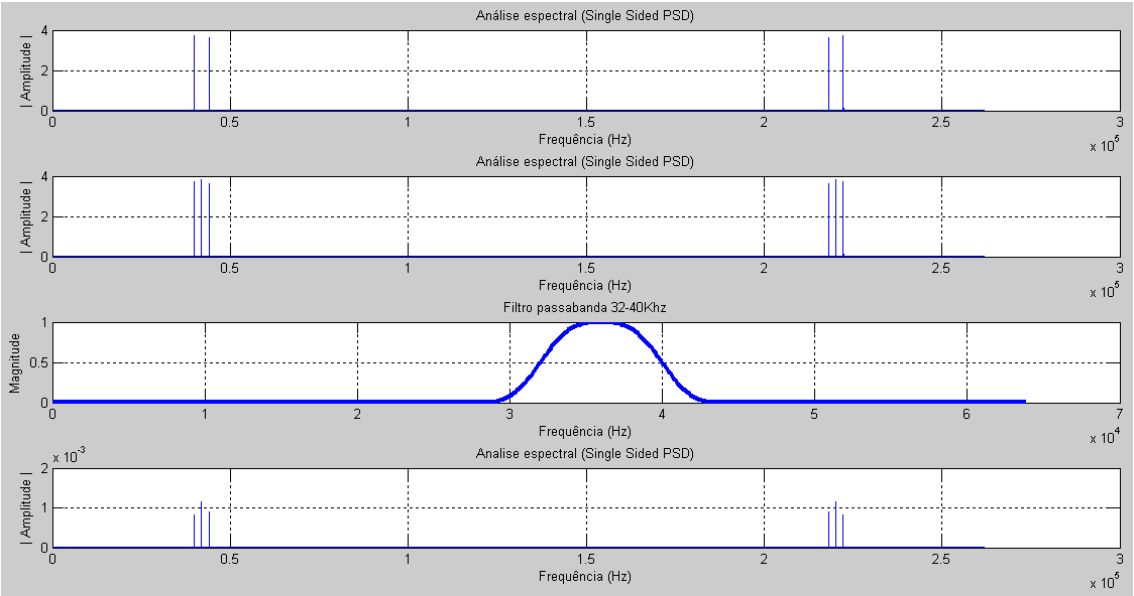


Figura 7: Captura da aplicação de ultrassons gerada em Matlab.

O excerto de código de Matlab não contém a programação para gerar as visualizações de forma a deixar somente o código que à posteriori vai ser implementado no DSP e também para tornar a percepção do código mais intuitiva. O som gerado correspondeu às expectativas, apresentando um elevado grau de directividade, e um nível de potência sonora bastante razoável, com um nível de distorção praticamente omisso.

3.4.2 - Arquitetura 2 – SSB – WC (Filtragem de Hilbert)

```
N = 100000;
fs = 192000;     %Frequência de amostragem
t = (0:N-1)/fs;
fm1 = 5000;Em1 = 2 ;m1 = Em1*cos(2*pi*fm1*t); % Componente 1 da
mensagem
fm2 = 10000;Em2 = 2;m2 = Em2*cos(2*pi*fm2*t); % Componente 2 da
mensagem
m = m1 + m2;                % Mensagem
fc = 40000; portadora = 5*cos(2*pi*fc*t); %Frequência da portadora
d = fdesign.hilbert(50,0.1);
hd = design(d,'firls');
coef = hd.Numerator;
j=1;
xv = zeros(size(m));
xz = zeros(size(m));
NZEROS = length(coef);
tempo = 1;
hilbertDelay = zeros(size(m));
while(tempo <= N)
    for ii = 1 : NZEROS
        xv(ii) = xv(ii+1);
    end;
    xv(NZEROS) = m(tempo);
    sum = 0.0;
    for k = 1 : NZEROS
        sum = sum + (coef(k) * xv(k));
    end;
    mh(tempo)=sum;
    for ii = 1 : N-1
        hilbertDelay(ii) = hilbertDelay(ii+1);
    end;
    hilbertDelay(NZEROS) = m(tempo);
    filterDelay = round(NZEROS/2);
    x1(tempo) = hilbertDelay( filterDelay ).*2.*cos(2*pi*fc*t(tempo));
    x2(tempo) = mh(tempo).*2.*sin(2*pi*fc*t(tempo));
    sbl(tempo) = x1(tempo) - x2(tempo);
    sbu(tempo) = x1(tempo) + x2(tempo);
    sinalModeladoPortadora(tempo)=sbu(tempo)+portadora(tempo);%DSB-WC
    tempo = tempo + 1;
 end;
%-----------------------------------------------------
order=63;           % Ordem
Fn=fs/2;            % Frequência de Nyquist
wl=40000/Fn;        % Frequência de corte inferior normalizada
wu=60000/Fn;        % Frequência de corte Superior normalizada
wc=[wl,wu];         % Vetor das frequências de corte
b=fir1(order,wc);   % Coeficientes do filtro
sinalModeladoPortadoraFiltrado = filter(b,1,sinalModeladoPortadora);
%-----------------------------------------------------
ultrassons = audioplayer(sinalModeladoPortadoraFiltrado,fs);
play(ultrassons)
```

Capítulo 4 –Implementação do sistema na FPGA

Como foi referido anteriormente, o sistema de som direcionado foi inicialmente implementado na FPGA Spartan 3E. Para tal foram usadas as ferramentas da XILINX, especialmente o ISE, e complementarmente também foi usada a ferramenta de simulação Sysgen, disponibilizada pelo MATLAB, que contém componentes nas suas livrarias que virtualizam vários módulos de hardware para serem implementados na FPGA. Dentro deste rol de componentes temos a livraria DSP, que disponibiliza cerca de 90 blocos de hardware virtualizados que podem ser adicionados ao Simulink. A imagem que se segue ilustra alguns dos componentes disponibilizados pela livraria DSP da ferramenta Sysgen.

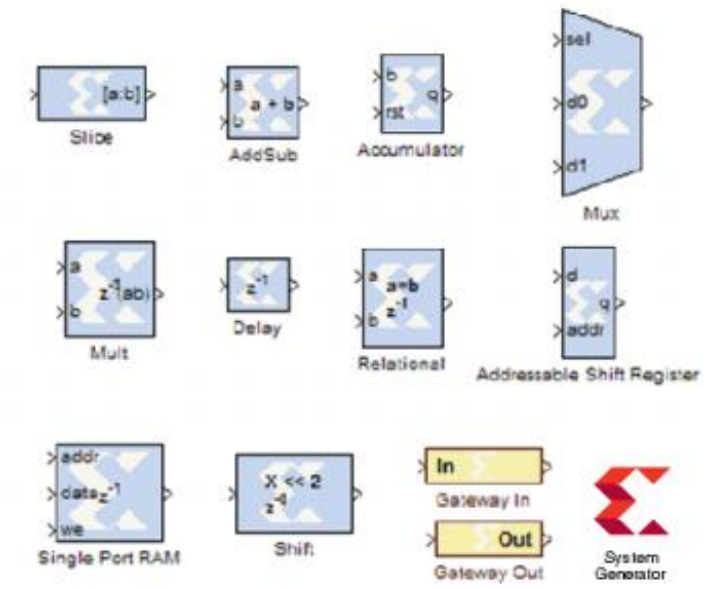


Figura 8:Block set DSP do Sysgen

A ferramenta, para além de poder simular o sistema a implementar na FPGA, com componentes que posteriormente são exportados para a FPGA, tem a capacidade de criar simulações aceleradas através da co-simulação com o hardware. Este processo de simulação é possível dado ao facto da ferramenta Sysgen, através do System Generator, conseguir criar uma simulação captada no Simulink com o Xilinx DSP blockset que vai correr na plataforma de Hardware suportada, que no nosso caso é a Spartan 3E. Este tipo de sistema contém quatro componentes fundamentais para ser implementado

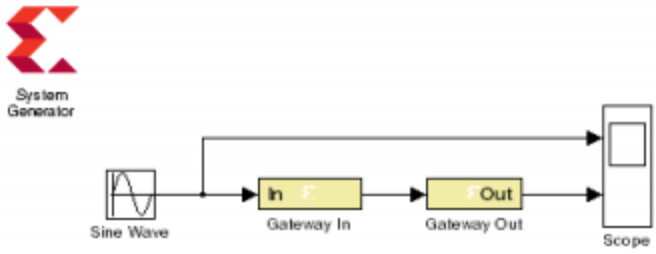


Figura 9: Configuração inicial no Sysgen

Token System Generator: Este componente é um membro constituinte da livraria de módulos disponibilizado para o Simulink, e tem de estar presente em todos os modelos desenhados de forma a ser possível simular e traduzir essa mesma simulação para Hardware. No caso do sistema gerado para a simulação do sistema de ultrassons, foi somente usado um token, e todos os módulos de hardware presentes na simulação estão ligados a este componente, pois é aqui que é configurado o Simulink System Period, assim como o tipo de FPGA, e a linguagem de descrição do hardware (VHDL ou Verilog)

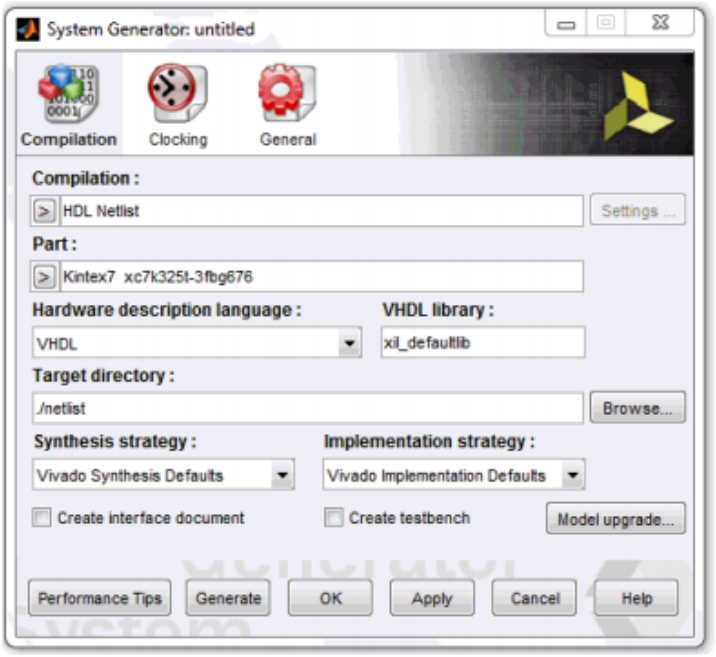


Figura 10:Token System Generator

Gateway In: A ligação entre os componentes Xilinx e os restantes componentes presentes no Simulink é feita através dos componentes de gateway. O Gateway In converte os sinais double para sinais Xilinx. Dentro deste bloco pode ser configurado o número de bits envolvidos na conversão. No caso da simulação de ultrassons, e devido ao número de bits prese ntes no ADC e no DAC da FPGA, foram configurados 12 BITS.

Gateway Out:Este componente converte o sinal Xilinx novamente num sinal double, de forma ao sinal poder operar no ambiente Simulink.

JTAG Co-Sim: Este é o bloco responsável pela co simulação de hardware. O componente interage com a FPGA, automatizando tarefas, como por exemplo, a configuração da FPGA, a transferência de dados, a sincronização, clock, etc.

4.1 - Acondicionamento de sinal

O sinal a transmitir vai ser capturado pelo ADC da FPGA de forma a ser modulado e transmitido em ultrassons. A imagem que se segue mostra o sistema de captura analógica da FPGA.

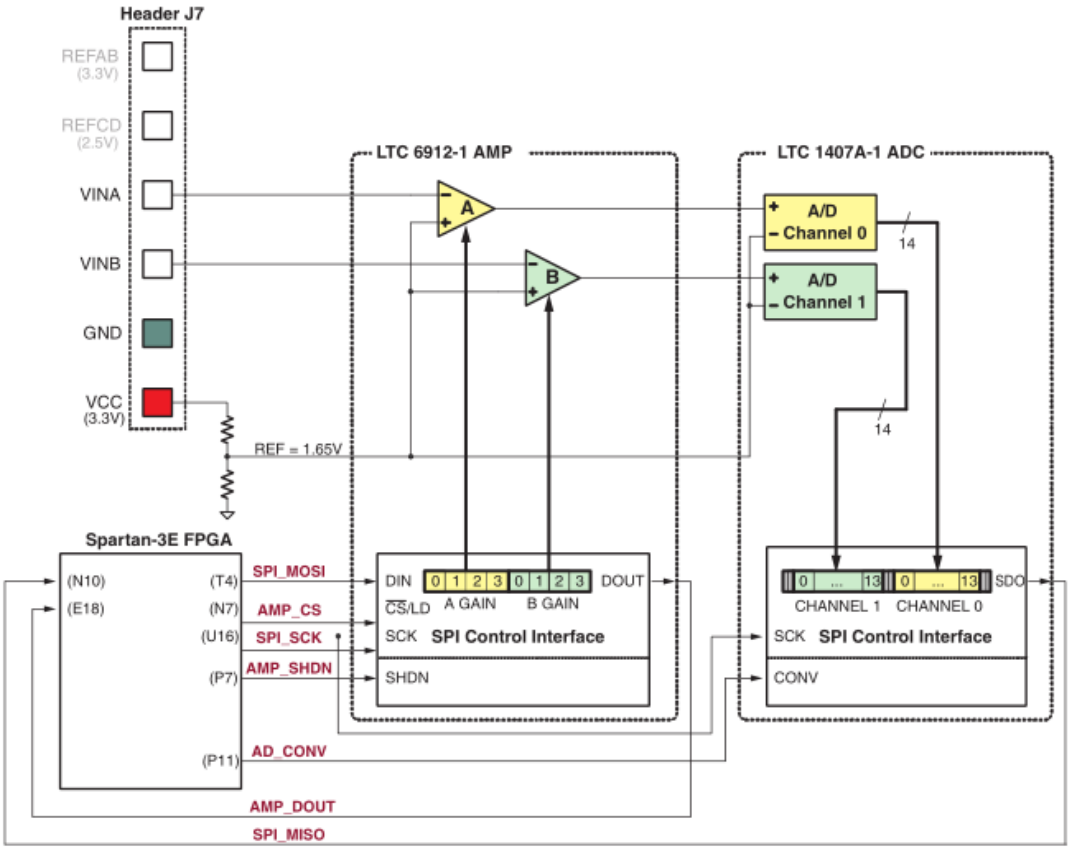


Figura 11: Sistema de captura analógica.

Como se pode verificar o kit da FPGA inclui dois pinos analógicos VINA e VINB que servem de interface com o exterior. Estes canais de captura analógica consistem num pré-amplificador programável (Linear *Technology* LTC6912-1) que amplifica o sinal analógico proveniente das entradas VINA e VINB presentes no conjunto de pinos J7. A saída do pré-amplificador é ligada ao acima referido ADC (Linear Technology LTC1407A-1 ADC). Tanto o ganho do pré-amplificador como os sinais de controlo do ADC são definidos digitalmente pela FPGA, utilizando o protoloco de SPI. O ganho do pré-amplificador escolhido para esta aplicação é o (-1), que proporciona uma excursão dos 0,4**V** até aos 2,9**V**, ideal para o objetivo final da aplicação. A tensão de referência no ADC é de 1,65**V** pelo que este valor é subtraído à tensão de entrada em VINA e VINB, tendo em atenção que a excursão máxima de tensão da aplicação do ADC é de ± 1,25**V** centrados na tensão de referência. De forma a maximizar a performance do ADC é necessário que o sinal de entrada esteja centrado na tensão de referência, e com uma amplitude que vai dos 0,4V até aos 2,9V. Para desenvolver um circuito que permita alcançar estes valores foi realizada uma simulação no LTSPICE. De seguida temos os resultados alcançados.

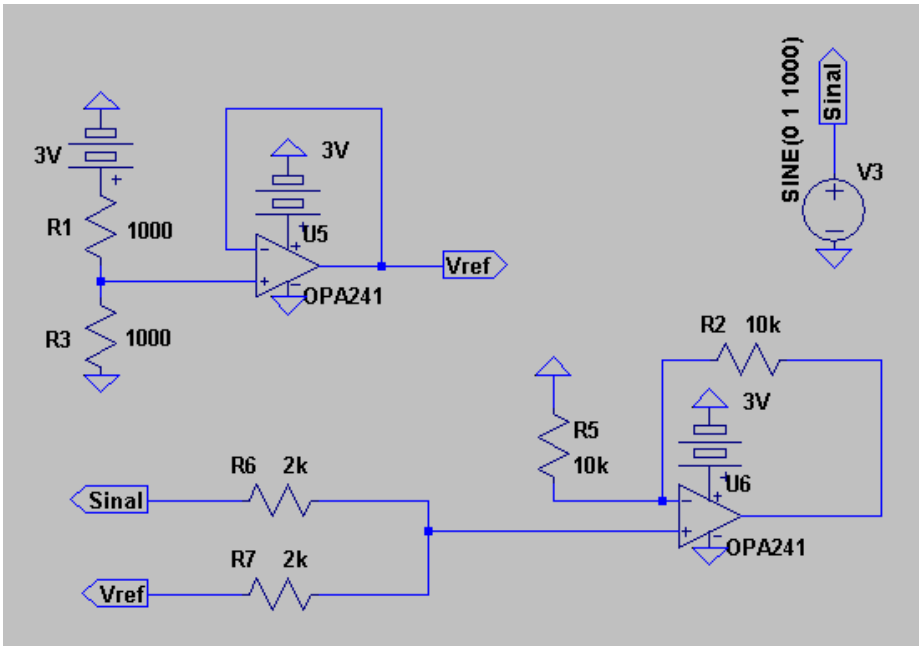


Figura 12: Circuito de acondicionamento de sinal.

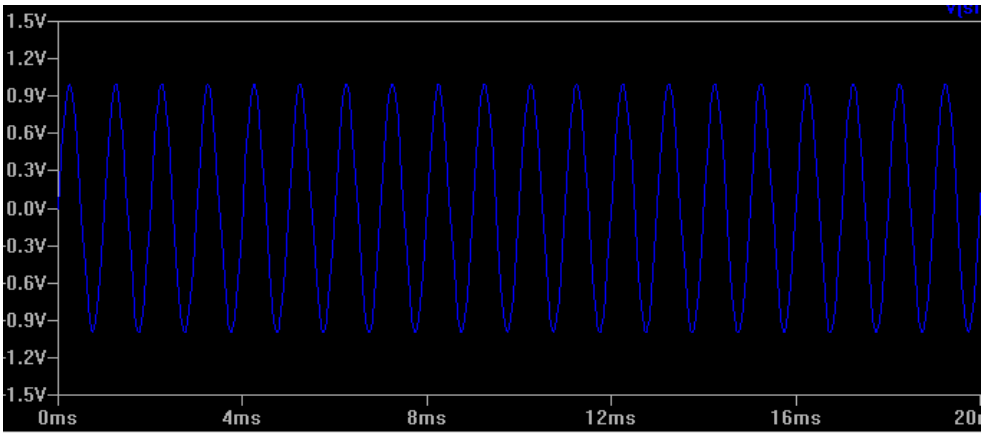


Figura 13: Sinal de entrada

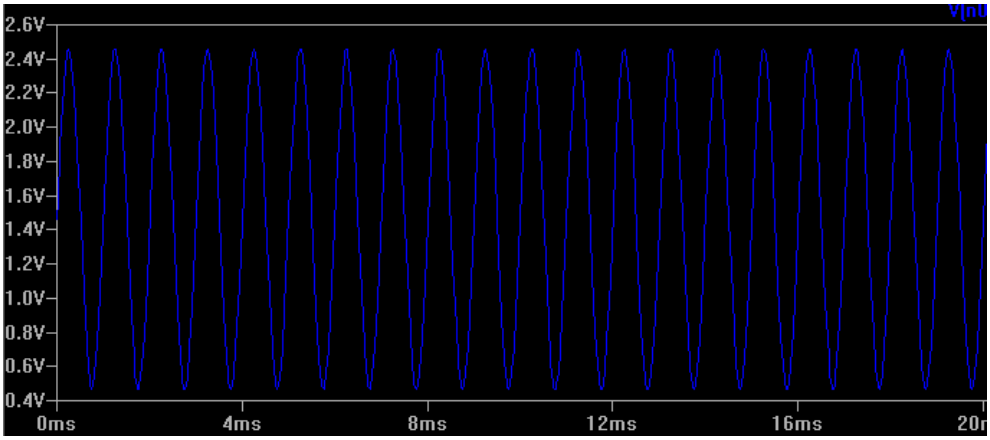


Figura 14: Sinal de saída

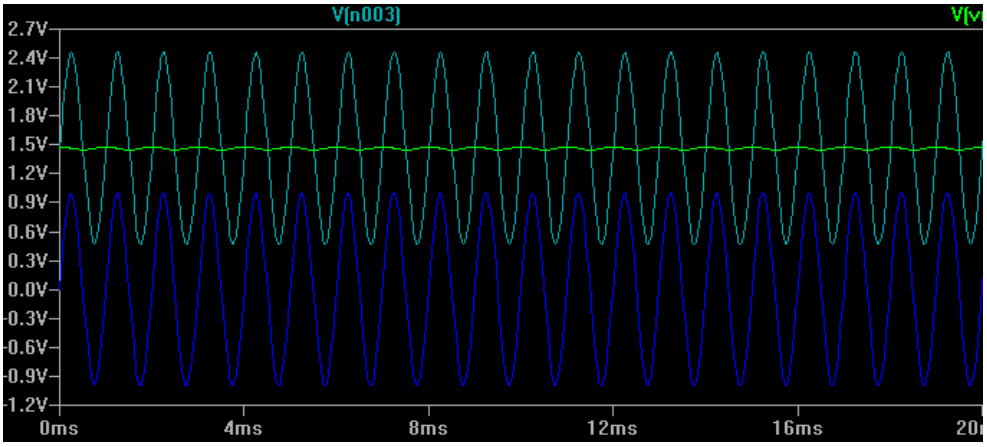


Figura 15: Sinal de entrada, saída e referência.

4.2 - PCB

Para realizar o acondicionamento de sinal foi criada uma placa PCB no programa Altium de forma a maximizar o sinal com o menor nível de ruido possível.

O programa consiste essencialmente em dois ficheiros:

- O Esquemático - Controlo_tensao.SchDoc
- Ficheiro PCB – Controlo_tensao_PcbDoc

De seguida temos as imagens dos dois ficheiros:

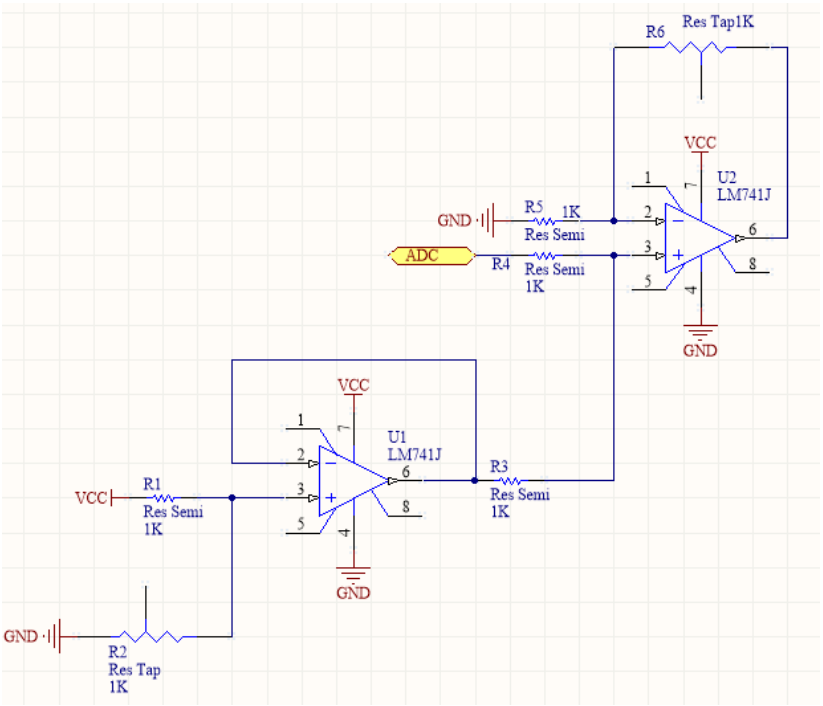


Figura 16: Controlo_tensao.SchDoc

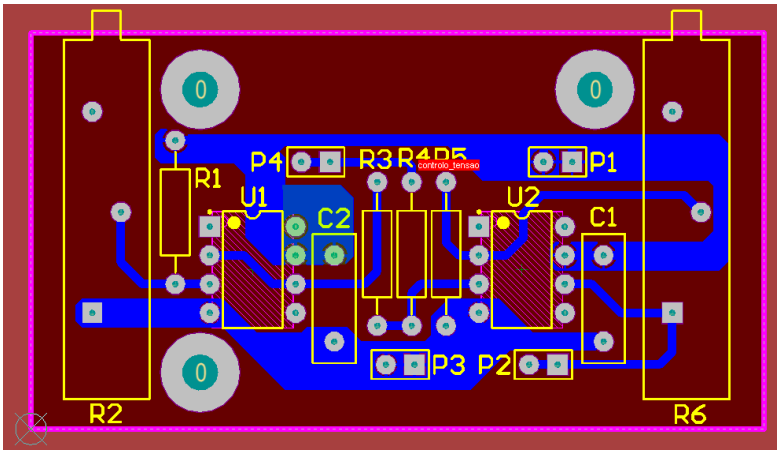


Figura 17: controlo_tensao_PcbDoc

4.3 - Módulo de controlo do ADC-DAC

O módulo de controlo do ADC e do DAC tem por finalidade controlar o sistema de comunicação com o exterior, assente num ADC e num DAC. A comunicação entre a FPGA e o ADC e o DAC faz-se através do protocolo SPI (Serial Peripheral Interface) como foi referido anteriormente.

A imagem que se segue ilustra a forma como o módulo atua para controlar o sinal de entrada e o sinal de saída.

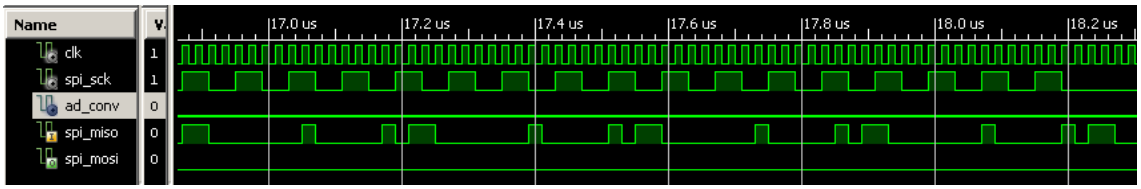


Figura 18:Captura no ISIM do sistema de controlo do ADC/DAC

Como se pode verificar pela imagem, os sinais que formam o módulo de controlo do ADC-DAC são os seguintes:

- CLK: Sinal de relógio
- SPI_SCK: Sinal de relógio do protoloco SPI.
- AD_CONV: Sinal que controla a captura de amostras analógicas.
- SPI_MISO: Master Input, Slave Output
- SPI_MOSI: Master Output, Slave Input

Estes quatro sinais controlam o ciclo de entrada e saída das amostras da FPGA atendendo ao ciclo do ADC e do DAC, enquanto os sinais STARTADC e STARTDAC controlam o processo que está a decorrer no momento, ou seja:

- STARTADC a 1: Indica à máquina de estados que o ADC está a processar uma captura.
- STARTDAC a 1: Indica à máquina de estado que o DAC está a processar uma amostra.

O ADC processa duas amostras de cada vez, uma proveniente do porto VINA e outra proveniente do porto VINB. Cada uma das amostras contém 14 bits no formato complemento para dois. A

trama que contém ambas as amostras é de 34 bits pois no início, meio e fim da trama o canal é colocado em alta impedância durante dois bits de forma a sinalizar o início e fim da trama, ou a entrada de uma amostra do outro porto. A imagem que se segue ilustra a trama do ADC.

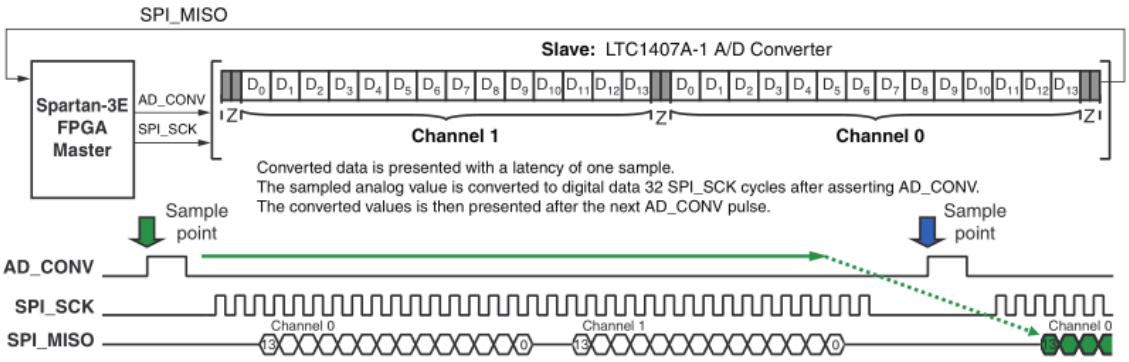


Figura 19: Trama do ADC

4.4 - IP CORES

Para desenvolver a aplicação também se recorreu ao uso de IP Cores disponibilizados pela Xilinx. Um IP (Intellectual Property) Core é uma aplicação disponibilizada que realiza uma determinada operação. Também podem ser disponibilizados agregados de IP Cores desenhados com uma determinada especificidade, mas neste trabalho somente vão ser usados IP Cores isolados. Esta livrarias são de extrema importância pois garantem ao utilizador máxima performance do seu hardware, pois são desenhados tendo em conta a FPGA a usar, e permitem diminuir o tempo de execução do projeto.

Os IP Cores usados são os seguintes:

- DDS Compiler
- Multiplier
- CORDIC

4.4.1 - DDS

Um dos componentes críticos dos sistemas DSP é o gerador sinusoidal. A maior parte das vezes recorre-se a uma Look Tablet de forma a colmatar o problema, mas na maior parte das vezes, caso se queira recorrer a uma sinusoide de elevada performance, esta solução mostra-se ineficaz, tornando-se um componente gerador de ruído adicional.

Um dos problemas mais recorrentes, é o facto de ser muito difícil manter baixo o nível de memória consumida num sistema que exija um nível elevado de SFDR (Spurious Free Dynamic Range) que consiste na relação entre a amplitude RMS da frequência portadora e o valor RMS da componente de ruído mais próxima ou do componente de distorção harmônica, assim como também é muito difícil implementar um gerador de sinusoide que tenha um comportamento linear ao nível da geração do ruído a velocidade muito elevadas de relógio.

No caso desta aplicação, é necessário gerar um seno e um cosseno, de elevada performance, de forma a se proceder à modulação AM DSB e SSB. Para tal viu-se que aqui era necessário recorrer

a um IP Core, de forma a se poder alcançar a performance desejada, e adicionalmente se poder alterar os parâmetros de forma simples e igualmente eficaz.

- Este IP Core contém um gerador de fase e uma Lookup Table para Seno e Cosseno.
- O gerador e a Lookup Table pode ser usados individualmente ou em conjunto, tendo opcionalmente recurso a um sistema de dither.
- A Lookup Table pode ser armazenada em RAM distribuída ou RAM em bloco.
- Sistema de dithering de fase, de forma a espalhar a energia linha espectral para um Spurious Free Range Dinâmico (SFDR) mais elevado.
- Opções de correção de ruído usando o mínimo de recursos de FPGA (Phase Dithering ou Taylor series correction).
- Suporte de SFDR desde 18 dB até 150 dB
- 16 Canais independentes multiplexados no tempo.
- Saída em complemento de 2 com sinais de 3 a 26 bits.

4.4.2 - Multiplier

A operação multiplicação é usada abundantemente em aplicações DSP, sendo necessário alcançar a máxima eficiência na implementação, de forma a obter os melhores resultados a velocidades de relógio elevadas.

O Multiplicador LogiCORE ™ simplifica este desafio abstraindo as especificidades da FPGA ao mesmo tempo que mantém o desempenho e a eficiência de recursos no máximo exigido. O multiplicador é capaz de gerar multiplicadores paralelos, e os multiplicadores de coeficientes constantes. Este IP Core disponibiliza um sistema de estimativa de recursos instantâneos, pelo que o utilizador pode selecionar rapidamente a solução ótima para o seu sistema.

Este IP fornece um bom controlo sobre a latência (pipelining) dos multiplicadores (puramente combinatória para fully pipeline) e symmetric rounding na slice DSP48. A implementação em fully pipeline permitem um desempenho de frequência de clock máximo

- Multiplicador fixed-point de complemento de 2 signed/unsigned
- Multiplicação paralela ou de coeficiente constante
- Suporta entradas que variam de 1 a 64 bits de largura e saídas que variam de 1 a 128 bits de largura com qualquer parte do produto completo selecionável.
- Suporta arredondamento simétrico ao infinito quando se utiliza o DSP Slice
- Instantânea Estimativa de Recursos
- Relógio opcional Ativar e Synchronous Limpar
- Modelos comportamentais VHDL
- Estimativa de Recursos Instantânea
- Para uso com Catalog Vivado® IP e Xilinx Sistema Gerador para DSP

4.4.3-CORDIC

O CORDIC (coordinate rotational digital computer) é um IP LogiCORE da Xilinx que implementa um algoritmo rotacional de coordenadas generalizado, inicialmente desenvolvido para resolver equações trigonométricas iterativamente, e mais tarde o algoritmo foi generalizado de forma a resolver uma gama mais ampla de equações, incluindo a hiperbólica e equações de raiz quadrada. O IP CORE CORDIC implementa os seguintes tipos de equações:

- Conversão Rectangular <-> Polar
- trigonométrico
- Hiperbólico
- Raiz quadrada

4.5-Componentes

A implementação do trabalho assentou na premissa de se tentar tornar o sistema o mais modular possível, de forma a isolar cada uma das ações em módulos diferentes, para que seja mais fácil desenhar e interpretar o sistema, mas também para ser mais intuitivo fazer debug ao sistema.

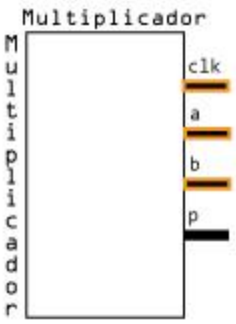
Neste capítulo são enumerados os diferentes componentes que constituem a implementação do sistema de Audio Spotlight. As portas de entrada são referenciadas a amarelo e as saída são referenciadas a preto.

O Componente que se segue tem por finalidade amplificar o sinal, antes de este dar entrada no ADC. A arquitetura deste amplificador é Inversor, e o ganho determinado foi de 1. Este componente partilha o meio SPI com o ADC e com o DAC, pelo que a arquitetura teve de ter este ponto em conta, de forma a sincronizar as portas dos respetivos componentes. A sincronização foi realizada através de uma máquina de estados como foi referido anteriormente.

Mclk: Relógio
RST: Reset
SPI_MOSI: Porta SPI(Master out Slave In)
PREAMP_SCK: Relógio SPI
AMP_CS: A '0' indica que o pre-amplifier está ativo
PRE_AMP_READY: Indica quando o componente está ativo

O multiplicador á imagem do componente anterior também é um IP Core. Havia a hipótese de implementar este Core através de LUTS, mas visto que todos os filtros são implementados através da arquitetura Distributed Arithmetic, no final, sobraram alguns multiplicadores, e dado

ao facto que estes têm um tempo de execução mais rápido, optou-se por usar multiplicadores nesta operação.



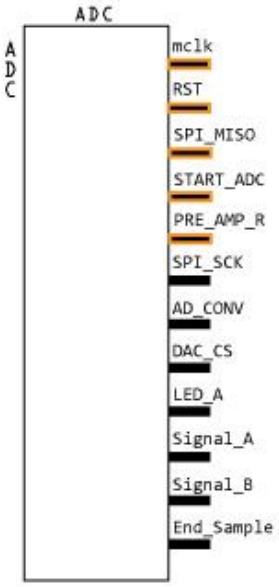
Clk: **Relógio**

A:**Termo A de 14 bits (Cosseno)**

B: **Termo B de 14 bits (Signal_A)**

P:**Produto de 24 bits**

O próximo componente é o responsável pela comunicação com o exterior. Como foi referido anteriormente, as amostras são capturadas a uma frequência de 222Khz. Atendendo a que a portadora do sistema é de 40Khz, esta frequência de amostragem é suficiente para ter um sistema robusto, com uma boa pureza espectral. Este componente partilha o meio SPI com o Pré-amplificador e com o DAC, como já foi referenciado anteriormente. Este componente está a funcionar com um relógio SPI de 12,5Mhz. A partir desse valor, as amostras começam a ter resultados insatisfatórios.



Mclk: **Relógio**

RST: **Reset**

SPI_MISO: **Porta SPI (Master In Slave Out)**

START_ADC: **Indica quando o componente está ativo**

PRE_AMP_R: **Atividade do pré-amp**

SPI_SCK: **Relógio SPI**

AD_CONV: **Sinal para retirar as amostras do ADC**

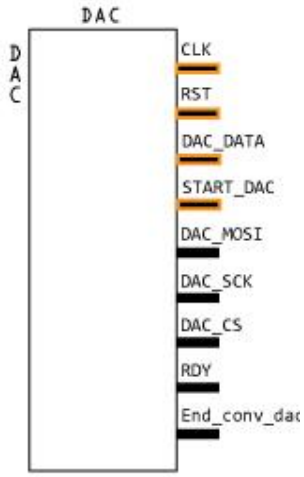
DAC_CS: **Indica a conversão digital – analógica a '1'**

Signal_A: **Captura do porto VINA**

Signal_B: **Captura do porto VINB**

End_Sample: **Indica o final de uma captura**

O DAC é o componente responsável por colocar o resultado da operação de modulação e filtragem no exterior. Foi possível colocar este componente a funcionar com um relógio SPI de 25Mhz, que é um valor bastante significativo para um componente desta natureza.



CLK: **Relógio**

RST: **Reset**

DAC_DATA: **Dados a ser amostrados**

SPI_MISO: **Porta SPI (Master In Slave Out)**

START_DAC**: Indica quando o componente está ativo**

DAC_MOSI: **Porta SPI (Master Out Slave In)**

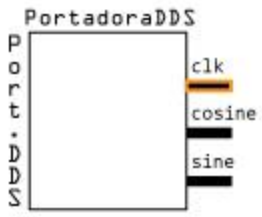
DAC_SCK: **Relógio SPI**

DAC_CS: **Indica a conversão digital – analógica a '1'**

RDY: **Indica que o DAC pode receber uma amostra**

End_Conv_dac: **indica o final de uma amostragem**

Como foi referido anteriormente, o componente gerador da portadora de 40Khz é um IP Core fornecido pela XIlinx, de nome DDS Compiler 4.0. A opção de usar um core, foi de otimizar o resultado final, e de encurtar significativamente o tempo de implementação do sistema na sua globalidade.

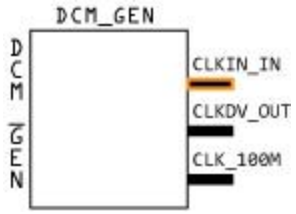


Clk: **Relógio**

Cosine: **Cosseno de 14 bits**

Sine: **Seno de 14 Bits**

O último IP Core usado na implementação deste sistema foi do gerador de relógio que temos a seguir. A geração deste módulo foi de bastante importância, especialmente porque a única forma de obter um relógio de 100Mhz era através deste componente, é essa frequência é bastante importante, pois os filtros exigem que se use uma frequência de execução 384 vezes superior à frequência de amostragem. Como a frequência de amostragem é de 222Khz, a frequência de execução dos filtros terá de ser superior a 85,248Mhz.

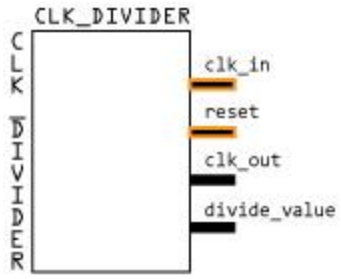


CLK_IN: **Relógio (50Mhz)**

CLKDV_OUT: **Sinal de relógio (25 Mhz)**

CLK_100M: **Sinal de relógio (100 Mhz)**

O módulo que se segue tem por finalidade gerar uma frequência de relógio de 220Khz de forma a termos as operações a decorrerem com uma velocidade semelhante em todos os módulos. A ter em conta que a frequência de amostragem é de 222Khz, mas não existe problema, pois o valor das amostras é guardada num registo, e a operação de modulação vai buscar o valor a esse registo a uma frequência de 220Khz.



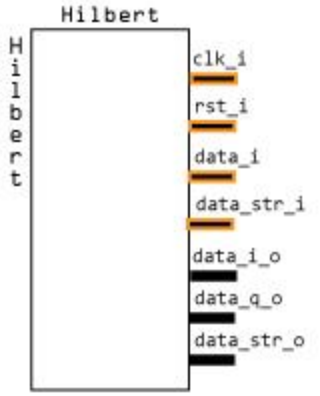
Clk_in: **Relógio**

reset: **Reset**

clk_out: **Sinal de relógio**

divide_value**: Inteiro que indica metade do período em ciclos de relógio.**

O Componente que se segue é responsável por realizar a filtragem de Hilbert, de forma a se poder realizar a modulação SSB. Como se pode verificar o módulo tem duas saídas a I e a Q, e para obter a modulação SSB multiplica-se a saída I por uma portadora em forma de seno a 40Khz e a saída Q por uma portadora cosseno também ela a 40Khz. Depois de realizar essa operação basta somar ambos componentes para se obter a zona espetral superior ou subtrair ambas as componentes para obter a zona espetral inferior da modulação.



Clk_i: **Relógio**

Rst_i: **Reset**

Data_i: **Entrada de dados de 14 bits**

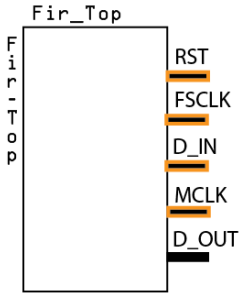
Data_str_i: **Entrada de dados de 14 bits**

Data_i_o: **Saída de dados filtrados em fase**

Data_i_q: **Saída de dados filtrados em quadratura**

Data_str_o: **Entrada de dados de 14 bits**

O proximo componente é o filtro passa-banda. Vão haver duas aplicações distintas, em que o fator de distinção é exatamente este filtro, pois um vai filtrar a banda lateral inferior da modelação DSB e a outra aplicação vai filtrar a banda lateral superior de forma a alcançar a modelação SSB ao se ficar somente com uma banda lateral.



RST: **Reset**

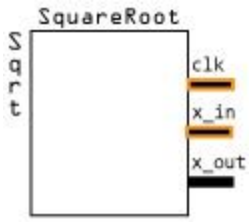
FSCLK**: Relogio (Frequência de amostragem)**

D_IN: **Entrada de dados de 14 bits**

MCLK**: Relogio Minimo de 384 a Frequência de amostragem**

D_OUT: **Saída de dados filtrados em fase**

Os componentes que se seguem realizam as operações necessárias para o tratamento de ruido da aplicação. O componente que se segue recorre também a um IP Core, o CORDIC, que Realiza várias operações através de um rotor algébrico. Entre essas operações está a raiz quadrada, necessária para a redução de distorção nos sistemas de Audio Spotlight.

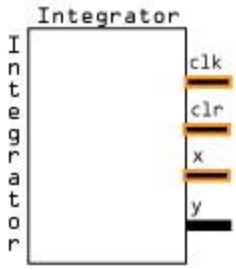


Clk: **Relógio**

X_in:**Entrada de dados**

X_out: **saída de dados**

O componente que se segue realiza a operação de integração. Para tal, este módulo assenta numa arquitetura de moving Integrator, que calcula a área subjacente à onda do sinal.



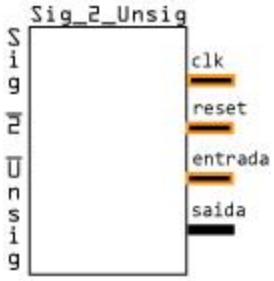
Clk: **Relógio**

Clr: **Clear**

X: **Entrada de dado(14 bits)**

Y: **Saída de dados (14 bits)**

O componente que se segue realiza a transformação do sinal de complemento de dois para inteiro.



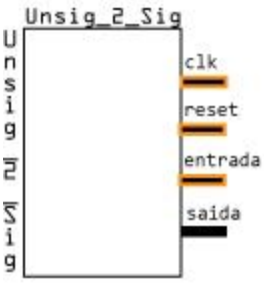
Clk: **Relógio**

Reset: **Reset**

Entrada: **Entrada de dados em complemento de dois**

Saída: **Saída de dados em Unsigned**

O componente que se segue realiza a transformação do sinal de inteiro para complemento de dois.



Clk: **Relógio**

Reset: **Reset**

Entrada: **Entrada de dados em Unsigned**

Saída: **Saída de dados em complemento de dois**

4.5.1-Máquina de estados

Na aplicação foram implementadas três máquinas de estados para os seguintes componentes:

- ADC, DAC e Top_Module

A máquina de estados que se segue é referente ao componente ADC. O primeiro estado corresponde à configuração do pré-amplificador. Como se pode verificar são enviados oito bits para a porta SPI_MOSI que controlam o ganho de cada uma das portas. No segundo estado o sinal AD_CONV é colocado a '1' de forma a ser retirada uma amostra de cada porta (VINA e VINB). O último estado tem a missão de enviar a tensão já digitalizada (em palavras de 14 bits) de cada uma das portas para o porto SPI_MISO. Inicialmente enquanto o contador é menor que dois não é enviado bit nenhum, pois a saída é colocada em alta impedância, depois é enviada a tensão da porta VINA, depois a saída é novamente colocada em alta impedância, para de seguida enviar a tensão da porta VINB.

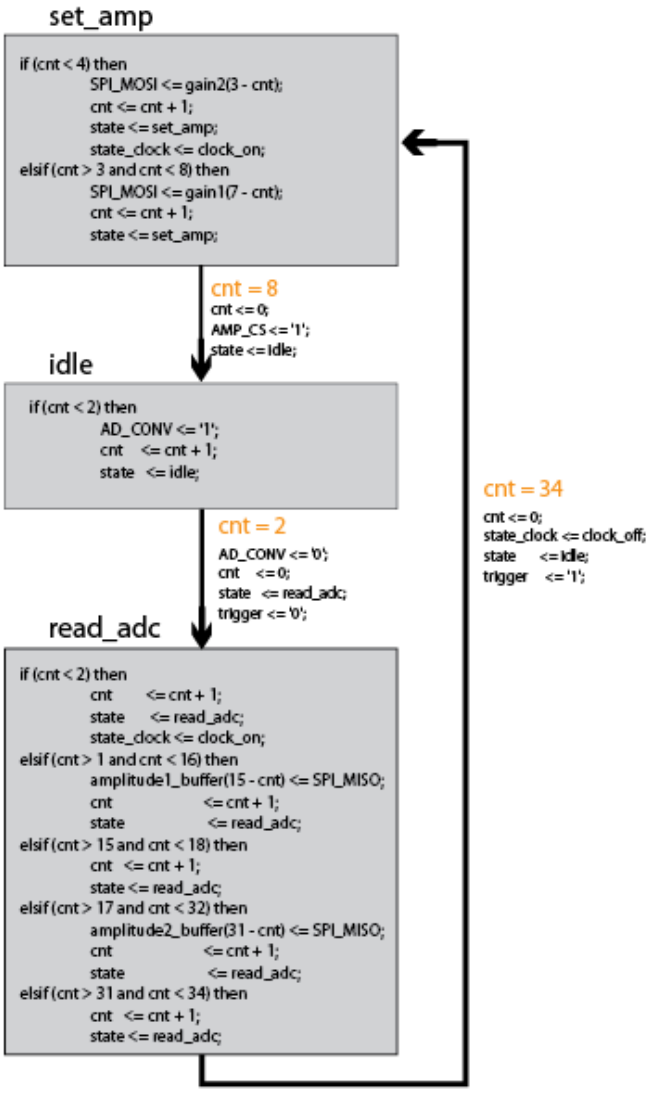


Figura 20: Máquina de estados do componente ADC.

A imagem que se segue corresponde à representação da máquina de estados do componente DAC. Nesta máquina de estados o processo é muito idêntico ao da máquina de estados anterior correspondente ao ADC. Como se pode verificar a palavra de 32 bits está alojada em DAC_SEND,

e enquanto o valor de índex não foi superior a 31, este é incrementado em um valor, e o valor do bit correspondente ao número do índex em DAC_SEND é enviado para a porta externa do DAC.

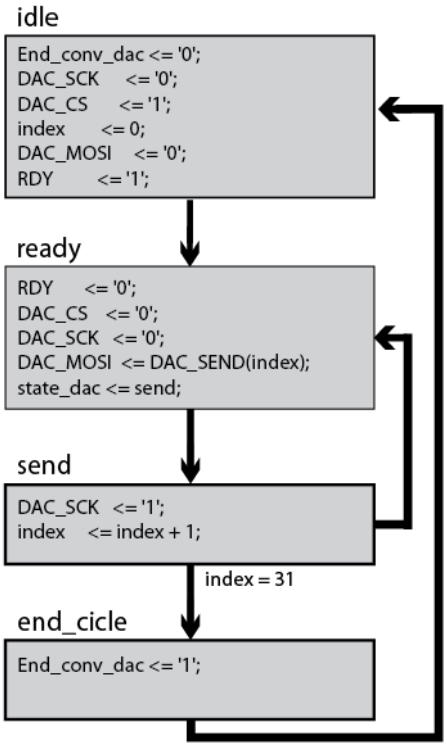


Figura 21: Máquina de estados do DAC.

A ultima máquina de estados corresponde à máquina de estados do componente Top_Module que controla a operação de captura e amostragem do sinal. Todavia, o primeiro estado está reservado para a operação de configuração do pré-amplificador. O segundo estado, fica à espera que entre uma amostra. O terceiro estado processa a captura do sinall por parte do ADC. Como se pode verificar o sinal START_ADC é colocado a '1' de forma a indicar à máquinade estados do ADC para esta iniciar a operação. O estado seguinte, de nome idelAdc volta a colocar o sinal START_ADC a '0' de forma a parar a máquina de estados do componente ADC. Por fim temos os dois estados finais referentes à amostragem do sinal no componente DAC. Estes estados funcioinam de forma igual aos estados do ADC, em que desta vez o sinal START_DAC indica ao componente DAC quand iniciar e parar a sua máquina de estados.

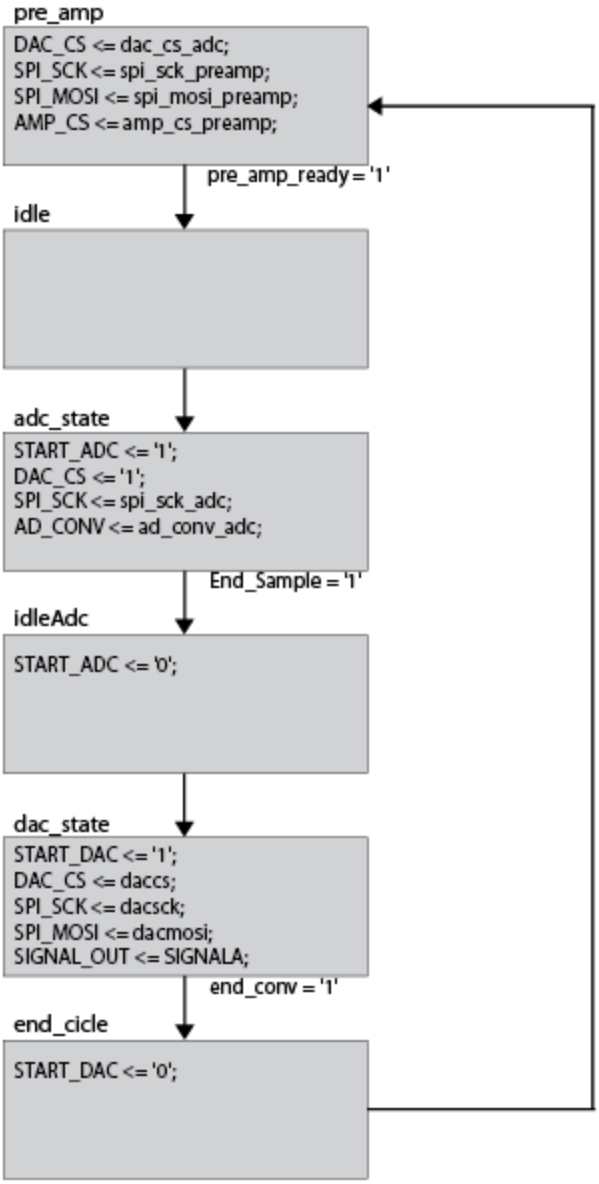
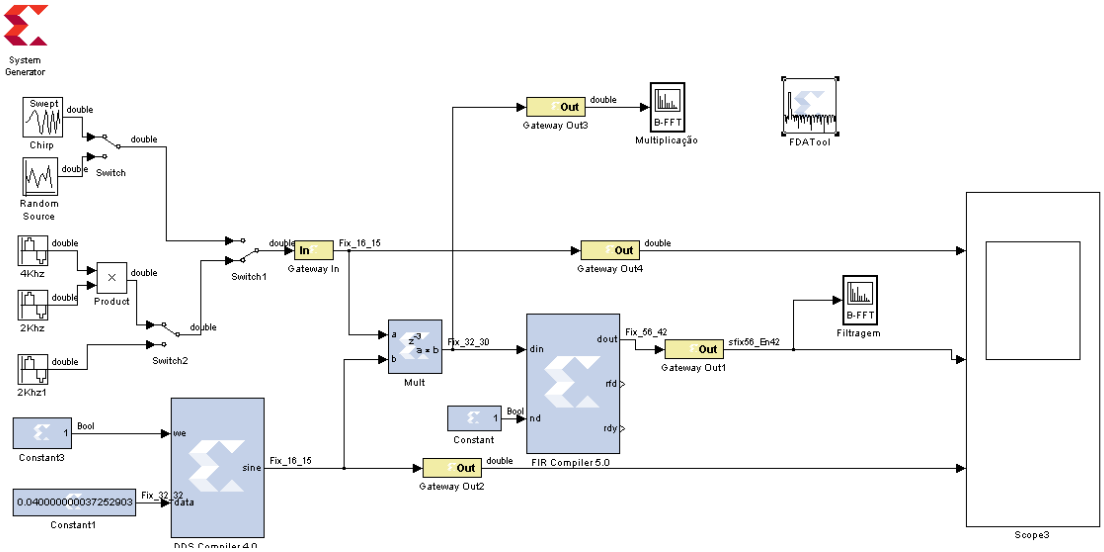


Figura 22:Máquina de estado dos do componente Top_Module

4.6- Simulações – MATLAB (Sysgen)

Por último foi realizada uma simulação recorrendo aos módulos Xilinx de forma a se ter uma simulação mais próxima dos resultados finais. Os resultados também foram os esperados, sendo que depois de gerado o HDL netlist verificou-se que o número de MULT18X18SIOs presentes na placa foi excedido, retirando a hipótese deste circuito ser exportado para a FPGA. No final deste capítulo é apresentado um excerto do Map Report gerado pela síntese do ISE.



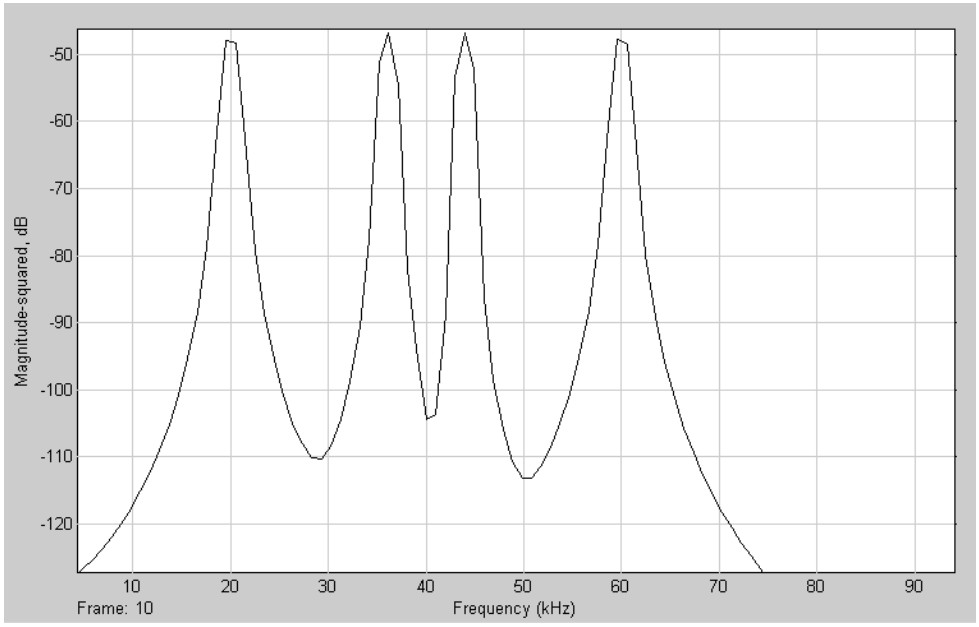


Figura 23: Sinal modulado

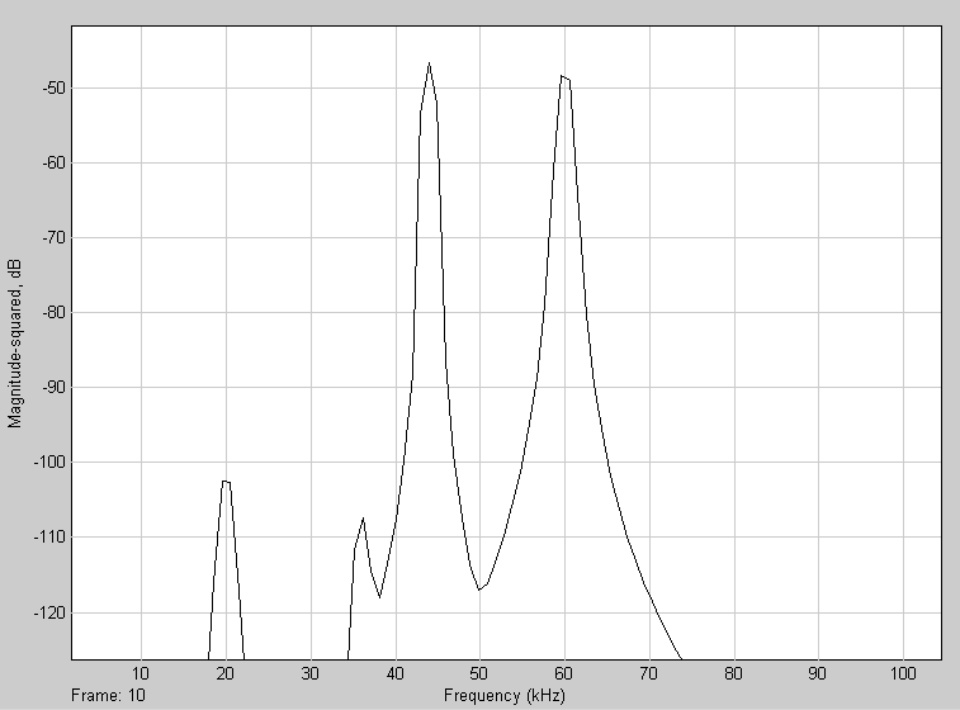


Figura 24: Sinal filtrado

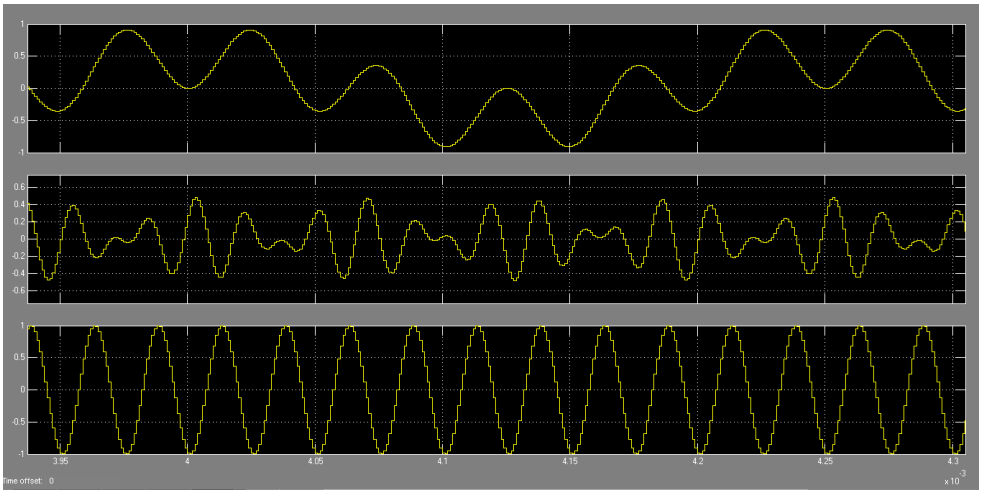


Figura 25: [1]Sinal modulado - [2]Sinal filtrado - [3]Portadora

4.7-Problemas na implementação

A implementação na FPGA Spartan 3-E apresentou dois problemas distintos, sendo que um deles veio a demonstrar ser impossível de contorná-lo. Os problemas foram os seguintes:

1. Problema na geração da portadora:
2. Recursos esgotados:

O primeiro problema consistiu em ter encontrado muitas dificuldades em implementar a portadora a funcionar com frequências superiores a 20Khz. A partir desse valor a sinusoide começava a distorcer em fase em amplitude. Este problema acontecia tanto no caso da

portadora ser gerada internamente com IPCORE ou através de uma Lookup Table, como também no caso de estar a ser mostrada uma sinusoide capturada no ADC.

O segundo problema consistiu na falta de recursos por parte da FPGA, especialmente ao nível de MULT18X18SIOs e de slices, pois ao aplicar somente uma filtragem esses recursos entravam automaticamente no limite. Em baixo está um map report gerado pelo ISE – Xilinx que ilustra a falta de recursos após a síntese do hadware.

MAP REPORT

```
Design Summary
--------------
Number of errors:     1
Number of warnings:   1
Logic Utilization:
  Number of Slice Flip Flops:       2,244 out of   9,312   24%
  Number of 4 input LUTs:           2,012 out of   9,312   21%
Logic Distribution:
  Number of occupied Slices:        1,342 out of   4,656   28%
    Number of Slices containing only related logic:  1,342 out of   1,342 100%
    Number of Slices containing unrelated logic:        0 out of   1,342   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:     2,087 out of   9,312   22%
    Number used as logic:           1,386
    Number used as a route-thru:       75
    Number used as Shift registers:   626

  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

  Number of bonded IOBs:            86 out of    232   37%
  Number of BUFGMUXs:               1 out of     24    4%
  Number of MULT18X18SIOs:          22 out of    20   110%
```

4.8-Resolução dos problemas na implementação

A resolução dos dois problemas foi encontrada depois de alguma pesquisa e mostrou-se bastante eficaz no problema relacionado com a escassez de recursos, enquanto relativamente à portadora apesar do problema ter sido resolvido, a portadora continuou a apresentar alguns harmónicos no domínio da frequência.

Relativamente à solução encontrada em relação à escassez de recursos, a forma encontrada foi recorrer à arquitetura Distributed Arithmetic nos filtros em vez dos multiplicadores do tipo MULT18X18SIOs.

Distributed Arithmetic é uma arquitetura que dispensa o uso de multiplicadores, que regra geral são um recurso escasso na FPGA. Ao invés do uso destes, a arquitetura Distributed Arithmetic os produtos e a soma destes de forma bastante eficiente através de LUTS, que são recursos abundantes nas FPGA's.

Relativamente à portadora, a forma de colmatar o problema foi procurar uma forma de aumentar a frequência de amostragem que inicialmente era de 160Khz, para 222Khz. Este aumento da Frequência de amostragem levou a uma pureza espectral mais elevada não só na portadora, mas em todo o sistema em geral.

4.9-Testes à implementação do sistema na FPGA SPARTAN 3E

De forma a concluir o estudo da implementação do sistema de Áudio Spotlight na FPGA Spartran-3E, foram realizados testes, de forma a compreender como o sistema se comporta nas seguintes vertentes:

- Frequência de amostragem
- Portadora
- Captura no ADC e amostragem no DAC
- Modulação DSB
- Filtros
- SSB(Filtragem de HIlbert)
- Tratamento do ruído

4.9.1-Frequência de Amostragem

A imagem que se segue representa o sinal START_ADC que indica o tempo em que o sistema esta á realizar a operação de captura. Como foi referido no capítulo anterior dedicado ao tema da captura realizada pelo ADC, este retira as amostras de dois portos distintos:

- VINA
- VINB

Para o funcionamento da aplicação somente é necessário a captura do porto VINA, de forma a ser posteriormente modelada pela portadora gerada internamente pelo IP CORE DDS Compiler 4.0.

Como se pode verificar pela imagem, a operação de captura demora 3,2us.

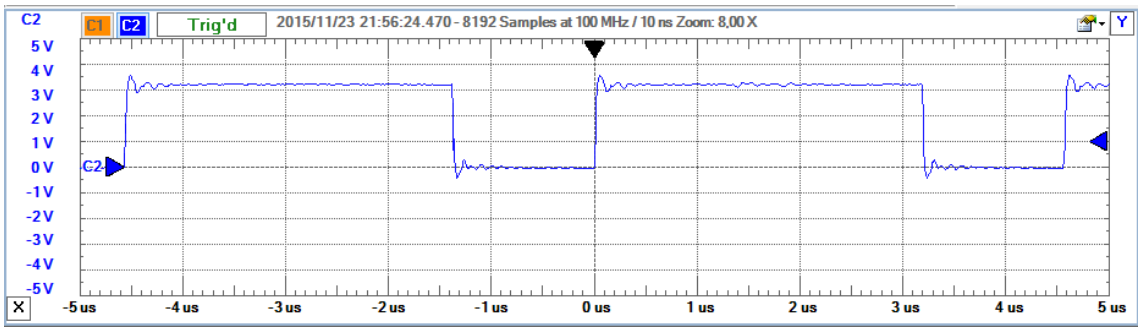


Figura 26: Sinal STAR_ADC (Tempo em que a operação realizada pelo ADC está ativa)

A imagem que se segue ilustra o sinal START_DAC que ilustra o tempo de em que a operação de amostragem por parte do DAC está ativa. Pela imagem verifica-se que o tempo que o DAC demora a realizar a operação de amostragem é de 1,3us.

Somando o tempo das duas operações chega-se ao seguinte tempo de frequências de amostragem:

Tempo de captura do ADC + tempo de amostragem do ADC = 4,5us

$$Frequência\ de\ amostragem = \frac{1}{4{,}5\ us} = 222Khz$$

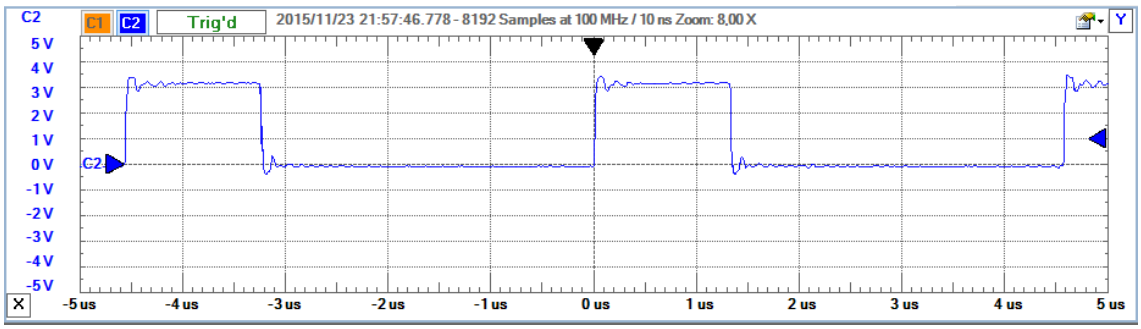


Figura 27: Sinal START_DAC (Tempo em que a operação realizada pelo DAC está ativa)

Os dois processos representados por ambos os sinais são controlados por uma máquina de estados de forma a se conseguir realizar ambas as operações, pois os dois dispositivos (DAC e ADC) partilham o meio SPI, pelo que não podem ser executados paralelamente, e necessitam de uma arquitetura de sincronização.

4.9.2-Portadora

Como já foi referido anteriormente a portadora é gerada a partir do core DDS 4.0 Compiler. Como se pode verificar na imagem que se segue a portadora está centrada nos 40Khz com uma potência de 0dBV. Não foi possível eliminar por completo a presença de alguns harmónicos presentes nas seguintes frequências:

Frequência	*Potência*
40,235Khz	-32,76dBV
19,375Khz	-35,438dBV
43,145 Khz	-36,229dBV
60,47 Khz	-37,59dBV
39,55 Khz	-38,564dBV

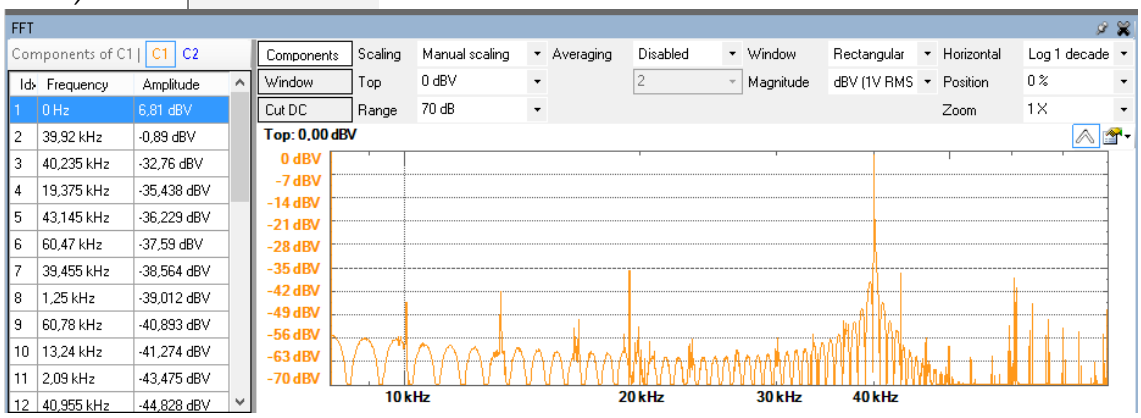


Figura 28:Portadora [Sinusoide a 40Khz]

A imagem que se segue representa a portadora no domínio do tempo. Como se pode verificar na onda, caso se olhe atentamente, verifica-se que cada período da onda tem sensivelmente 5 pontos para caracteriza-la. Este resultado corresponde ao esperado pois a frequência de amostragem é de 220Khz, e se se realizar a seguinte operação:

$$\frac{Frequência\ de\ amostragem}{Frequência\ da\ portadora} = \frac{220Khz}{40Khz} \approx 5\ pontos$$

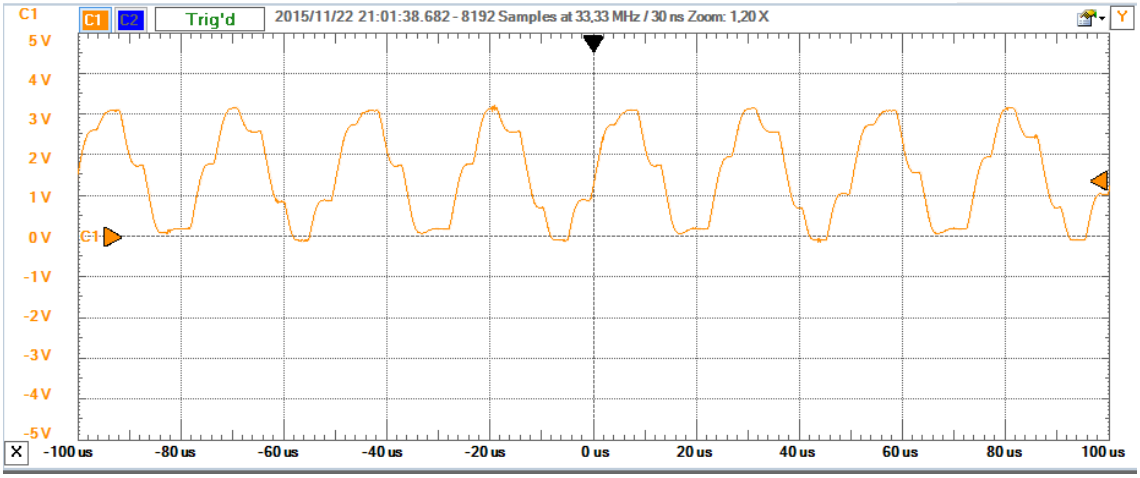


Figura 29: Portadora no domínio do tempo.

4.9.3-Sinal capturado pelo ADC

Neste capítulo foram realizados testes de captura e amostragem de uma sinusoide por parte do ADC e do DAC. Como se pode verificar na imagem, o sinal apresenta uma potência de 10bDV, e apresenta dois harmónicos significativos coma s seguintes frequências:

Frequência	*Potência*
49,295Khz	-43,21dBV
69,295Khz	-44,267dBV

Há que ter em conta que há medida que a frequência da sinusoide vai sendo aumentada, vão sendo gerados novos harmónicos, até que ao chegar aos 40Khz, a sinusoide apresenta os mesmos harmónicos que a portadora.

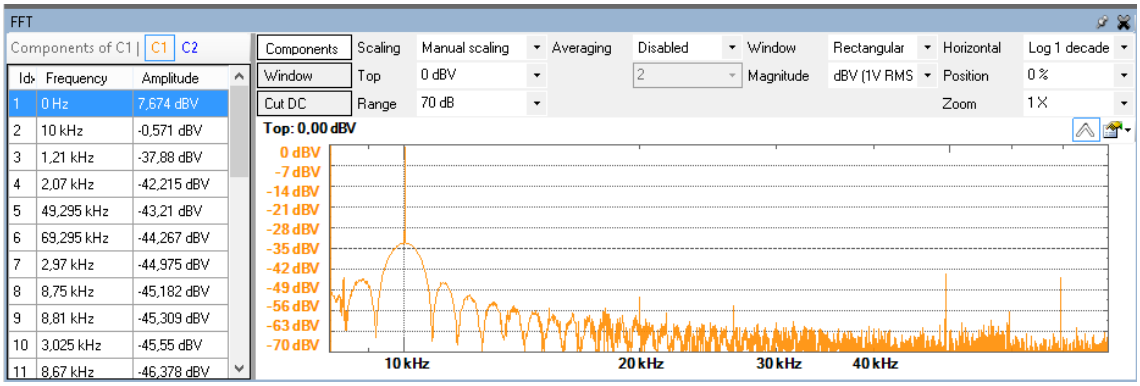


Figura 30: FFT do sinal capturado pelo ADC e amostrado pelo DAC.

Como já foi referido anteriormente, o sinal é amostrada a uma frequência de amostragem de 220Khz. A imagem seguinte apresenta a captura do sinal no domínio do tempo. Há que verificar que neste exemplo o sinal apresenta uma excursão que vai desde os 0,4V até aos 2,9V, que é o máximo permitido pelo ADC presente na FPGA.

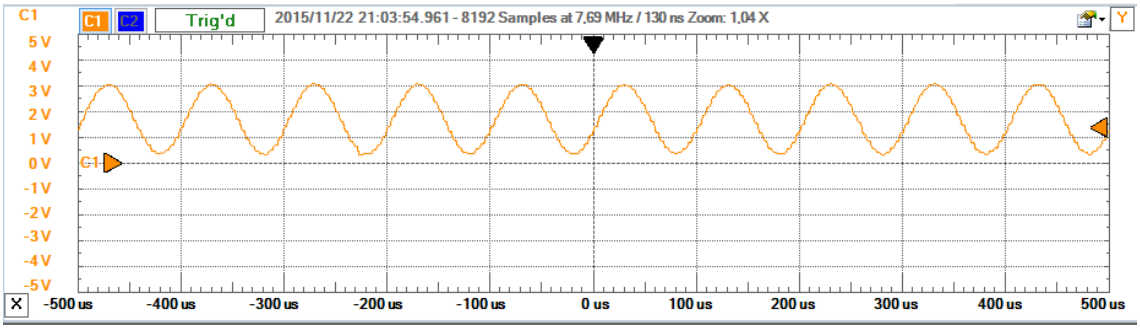


Figura 31:Sinal capturado pelo ADC e amostrado pelo DAC no domínio do tempo.

4.9.4-Sinal Modelado

Depois de realizada a operação de captura e amostragem do sinal, foi chegada a hora de começar a entrar nas partes mais críticas do sistema. A primeira a ser tratada foi a modelação. Como já foi referido anteriormente a modelação ficou a cabo de um IP Core disponibilizado pela Xilinx; o multiplier. OIP Core dá a hipótese usar a arquitetura distributed arithmetic que usa LUTs como recurso, ou então usar multiplicadores, que usa MULT18X18SIOs como recurso. Neste caso, e de forma a agilizar o processo foi usada a segunda opção, pois esta é bastante mais rápida. Como se pode verificar pela imagem, a modelação da sinusoide de 10Khz apresenta resultados bastante satisfatórios, pecando somente, pelo facto de perder um pouco de potência na operação.

Como se pode verificar pela imagem, os pontos de maior potência são:

Frequência	*Potência*
29,92Khz	-32,76dBV
39,92Khz	-35,438dBV
49,92Khz	-22,515dBV

Estes valores correspondem a uma modelação DSB-WC como previsto.

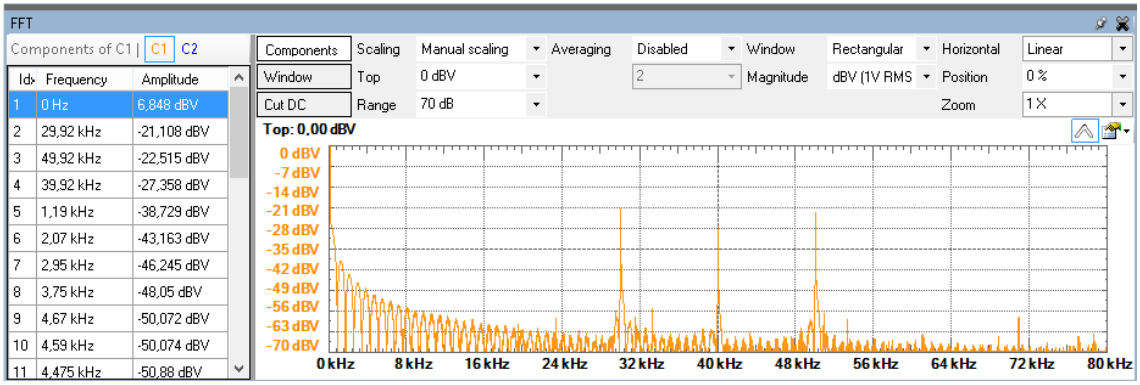


Figura 32: Sinal modulado no domínio da frequência.

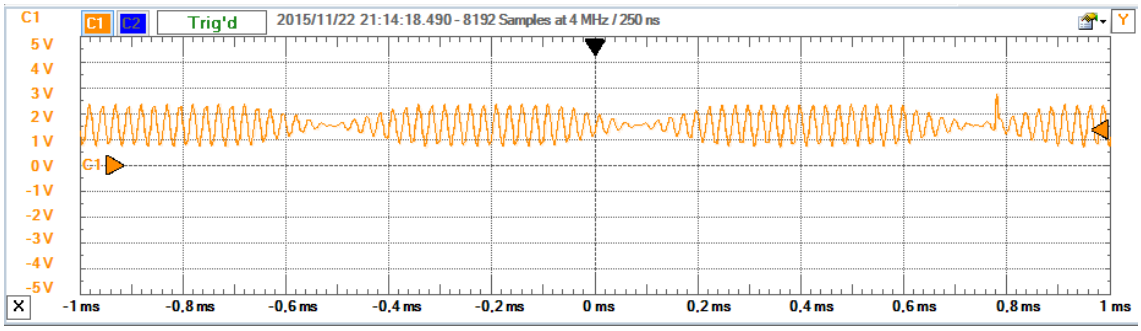


Figura 33:SInal modulado no domínio do tempo.

4.10-Filtragem

A filtragem, à imagem da modulação, é outro ponto crítico da aplicação, pois nesta reside também o bom desempenho do sistema de um modo global. Como foi referido anteriormente, no plano do modelo conceptual, vão haver três tipos de sistemas, que vão ser caracterizados essencialmente pelo seu tipo de filtragem, que são:

- Filtro passa-banda com fc=20-40Khz
- Filtro passa-banda com fc=20-40Khz
- Filtragem de Hilbert

A finalidade dos três tipos de filtragem é obter a modo de modulação SSB (Single Side Band). Nos subcapítulos que se seguem vai ser demostrada a forma como se alcançou cada uma das filtragens, acompanhados de imagens do resultado que estas tiveram sobre um sínal de ruído branco. Os filtros passa banda inicialmente foram implementados através do script Matlab, que se segue:

```
A_stop1 = 60;     % atenuação na primeira banda de corte = 60 dB
F_stop1 = 20e3;   % limite da banda de corte = 20000 Hz
F_pass1 = 24e3;   % inicio da banda passante = 24000 Hz
F_pass2 = 40e3;   % fim da banda passante = 40000 Hz
F_stop2 = 44e3;   % limite da segunda banda de corte = 44000 Hz
A_stop2 = 60;     % atenuação na segunda banda de corte = 60 dB
A_pass = 1;       % Nível de ripples permitidos na banda passante = 1
dB
Fs = 227e3;
Order = 100;
fdes = fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2', F_stop1,
F_pass1, F_pass2, F_stop2, A_stop1, A_pass, A_stop2, Fs);

FiltroPassaBanda = design(fdes,'equiripple', 'filterstructure',
'dffir');

FiltroPassaBanda.Arithmetic = 'fixed';
FiltroPassaBanda.InputWordLength = 12;
FiltroPassaBanda.InputFracLength = 11;
specifyall(FiltroPassaBanda);
FiltroPassaBanda.OutputWordLength = 12;
FiltroPassaBanda.OutputFracLength = 11;
FiltroPassaBanda.RoundMode = 'round';
FiltroPassaBanda.OverflowMode = 'saturate';
%fvtool(FiltroPassaBanda)

FL = length(FiltroPassaBanda.numerator)

dalut = [ones(1, floor(FL/8))*8, mod(FL, 8)]

workingdir = 'DA_FIR_Filter_PassaBanda_1910';
generatehdl(FiltroPassaBanda, 'DALUTPartition', dalut,
'TargetDirectory', workingdir);
edit(fullfile(workingdir, 'FiltroPassaBanda.vhd'));
```

E o respetivo testbench:

```
userstim = [];
for n = [ 50, 100, 150, 200, 250, 300, 500, 600, 700, 800, 1000,
1200,1500,2000,2500,3000,3500,4000,4500,5000,5500,6000,7500,8000,850
0,9000,9500,10000,15000,16000,16500,17000,17500,18000,18500,19000,19
500,20000,20500,21000,21500,22000,22500,23000,23500,24000,24500,2500
0,25500,26000,26500,27000,27500,28000,28500,29000,29500,30000,30500,
31000,31500,32000,32500,33000,33500,34000,34500,35000,35500,36000,36
500,37000,37500,38000,38500,39000,39500,40000,40500,41000,41500,4200
0,42500,43000,43500,44000,44500,45000,45500,46000,46500,47000,47500,
48000,48500,49000,49500,50000,50500,51000,51500,52000,52500,53000,53
500,54000,54500,55000,55500,56000,56500,57000,57500,58000,58500,5900
0,59500,60000,60500,61000,61500,62000,62500,63000,63500,64000,64500,
65000,65500,66000,66500,67000,67500,68000,68500,69000,69500,60000,70
500,71000,71500,72000,72500,73000,73500,74000,74500,75000,75500,7600
0,76500,77000,77500,78000,78500,79000,79500,80000]
userstim =[userstim,sin(2*pi*n/Fs*(0:Fs/n))];
end
xrange = (0:length(userstim) - 1);
y = filter(FiltroPassaBanda, userstim);
subplot(2,1,1); plot(xrange, userstim);
title('HDL Butterworth filter in Stimulus');
xlabel('Sample #');
subplot(2,1,2); plot(xrange, y);
title('HDL Distributed Arithmetic FIR filter out Response');
xlabel('Sample #');
```

Mas ao ser implementado este filtro adicionava mitos harmónicos, além de diminuir bstante a amplitude do sinal.

Por fim, acabou por se usar um programa externo para gerar os filtros. O programa tem por nome **dfalz1.exe** e provou ser muito adequado às necessidades desta aplicação, pois tem parâmetro de configuração que ajudaram a encontrar um filtro que se enquadre com a aplicação.

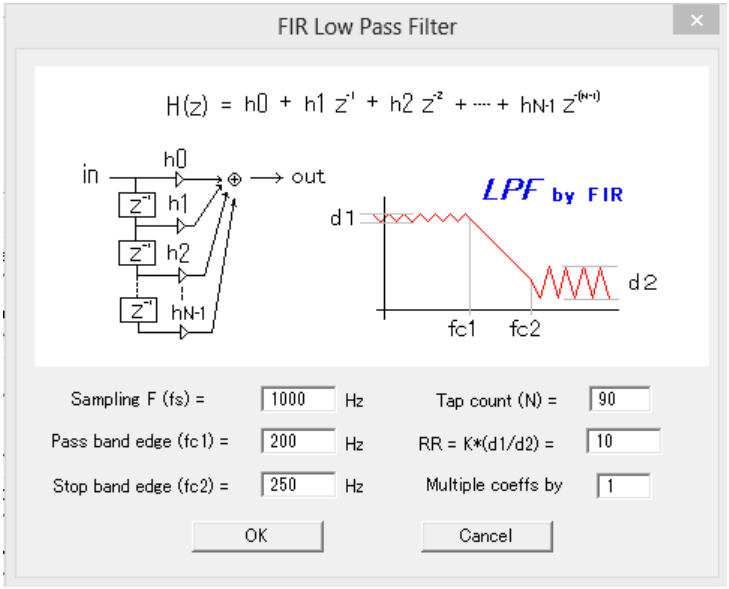


Figura 34:Opções de configuração.

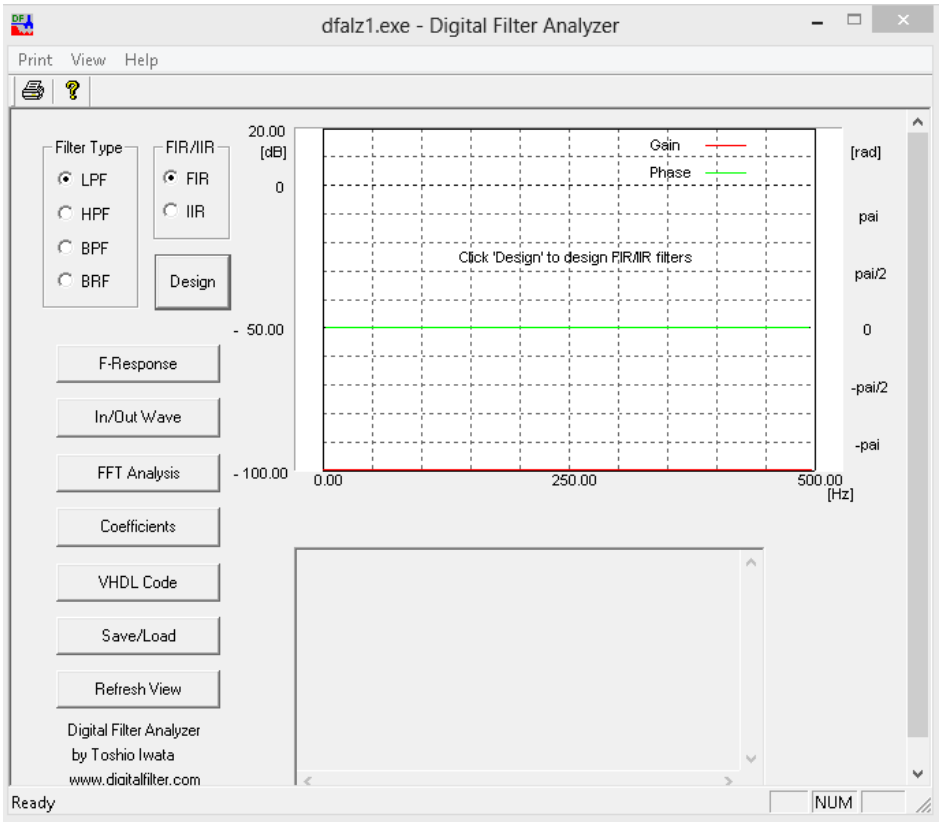


Figura 35: Programa dfalz1.

4.10.1-Filtro Passa-Banda 20 - 40Khz

O primeiro filtro a ser abordado é o filtro passa-banda com uma banda passante que vai dos 20Khz até aos 40Khz. A configuração do filtro foi a seguinte:

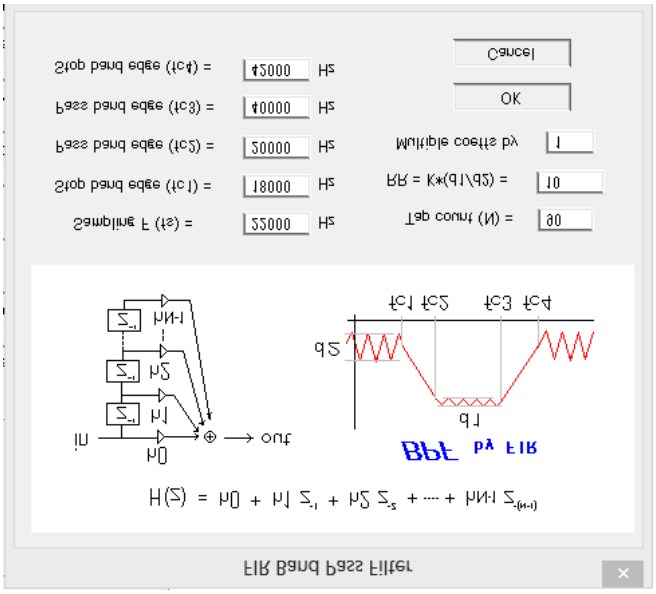


Figura 36: Configuração do Filtro Passa-Banda 20 - 40Khz

O resultado obtido foi o seguinte:

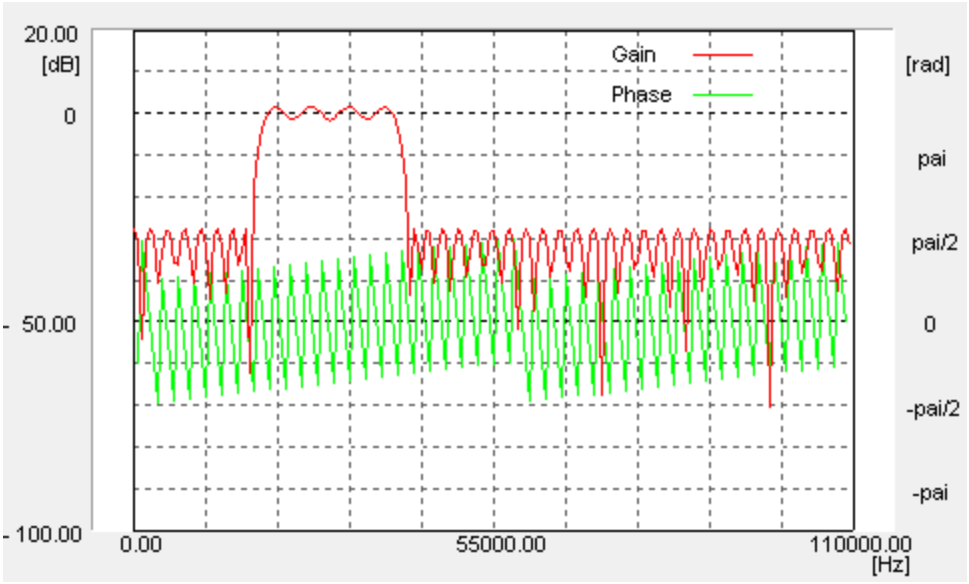


Figura 37: Banda passante do Filtro Passa-Banda 20 - 40Khz

Após a implementação do filtro, foi realizado um teste com um sinal de ruído brande de forma a visualizar a banda-passante. Como se pode verificar pela imagem que se segue, o resultado é idêntico à imagem anterior, o que nos leva a concluir que o filtro vai ter um bom desempenho no sistema de Audio-Spotlight.

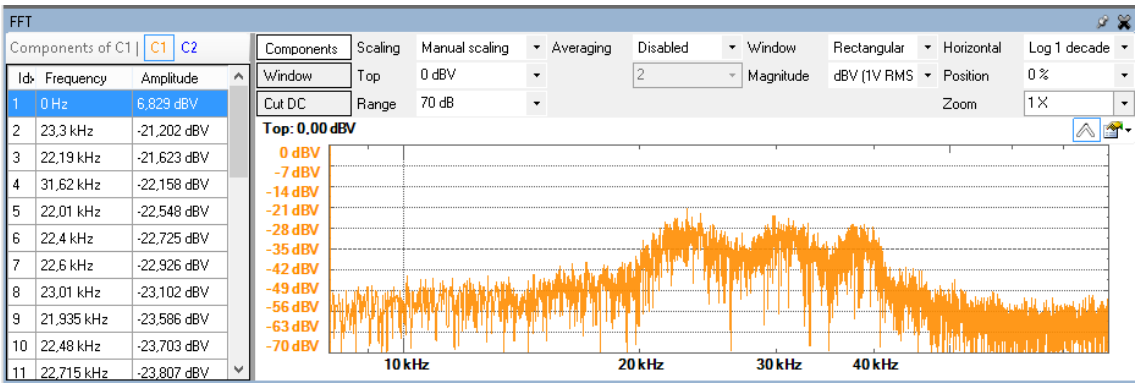


Figura 38: Captura do Filtro Passa-Banda 20 - 40Khz ao injetar um sinal de ruído branco.

4.10.2-Filtro Passa-Banda 40 - 60Khz

De forma a aproveitar a banda lateral superior da modelação DSB, foi também construído um filtro com uma banda-passante dos 40 aos 60Khz. A imagem que se segue representa a configuração usada para desenhar o filtro no programa dfalz1.

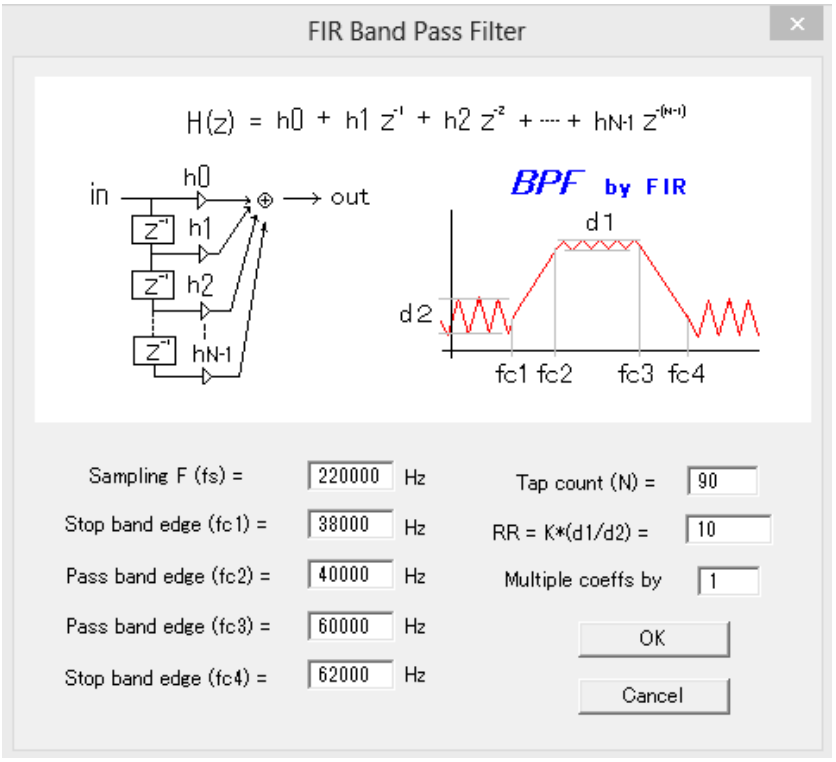


Figura 39: Configuração do Filtro Passa-Banda 40 - 60Khz

A imagem que se segue ilustra a banda passante resultante.

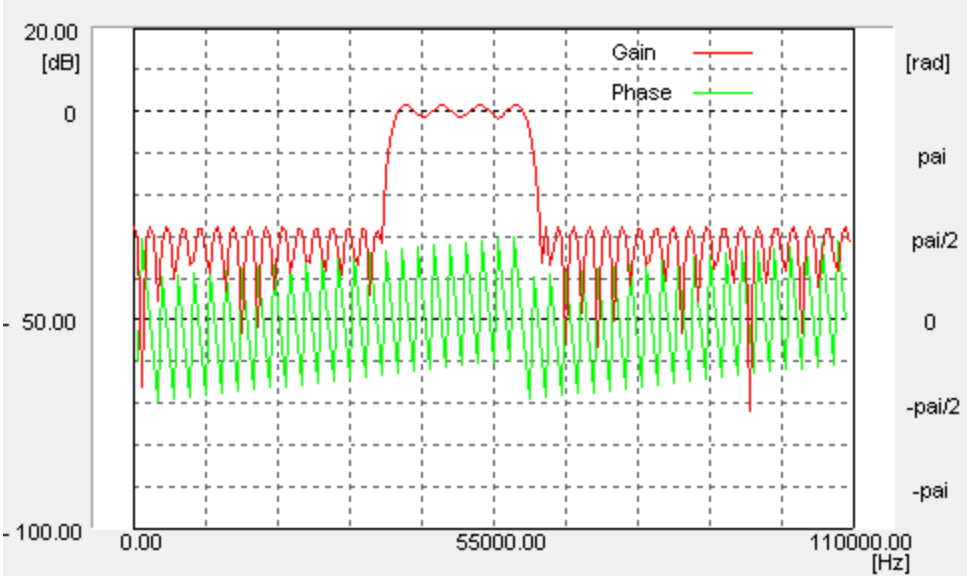


Figura 40:Banda passante do Filtro Passa-Banda 40 - 60 Khz

À imagem do filtro anterior que filtrava a banda lateral inferior da modelação DSB, também este filtro passa-banda, que filtra a banda lateral superior, apresenta um resultado idêntico a imagem anterior após ter sido injetado um sinal de ruído branco.

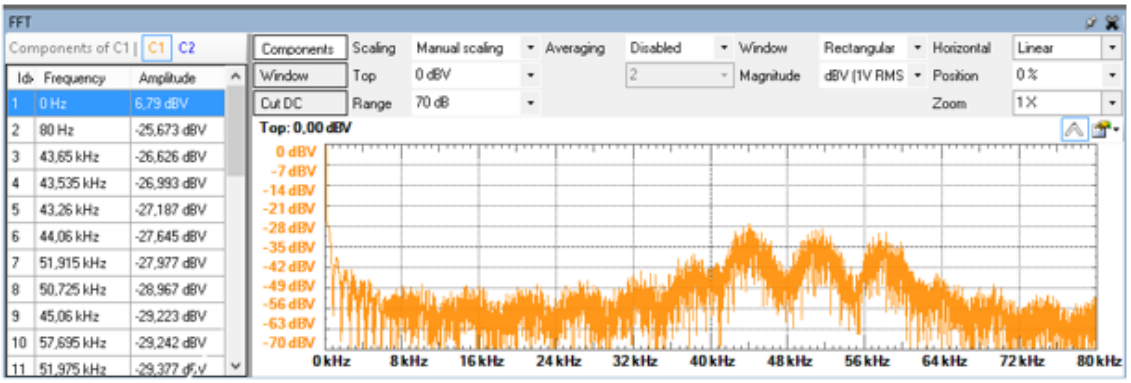


Figura 41: Captura do Filtro Passa-Banda 40 - 60Khz ao injetar um sinal de ruído branco.

4.10.3-Filtragem de Hilbert

. A transformação de Hilbert é um componente muito importante em sistemas de comunicação, através da capacidade de gerar a modelação single sideband, assim como a capacidade de detetar mudanças de fase e de amplitude. Neste caso a filtragem de Hilbert é usada como já foi referido anteriormente para gerar a modulação single sideband de forma a se retirar uma das bandas laterais. A transformada de hilbert é neste caso é alcançada formulando a operação como se trata-se de uma operação de filtragem, aproximando assim a Transformada de Hilbert a um filtro digital.

Devido à não causalidade e à resposta infinita ao impulso, é muito dificil alcançar uma boa aproximação com recursos de hardware moderados, como é o caso da implementação do sistema na FPGA Spartan 3E.A implementação do filtro está representado no diagrama de blocos que se segue:

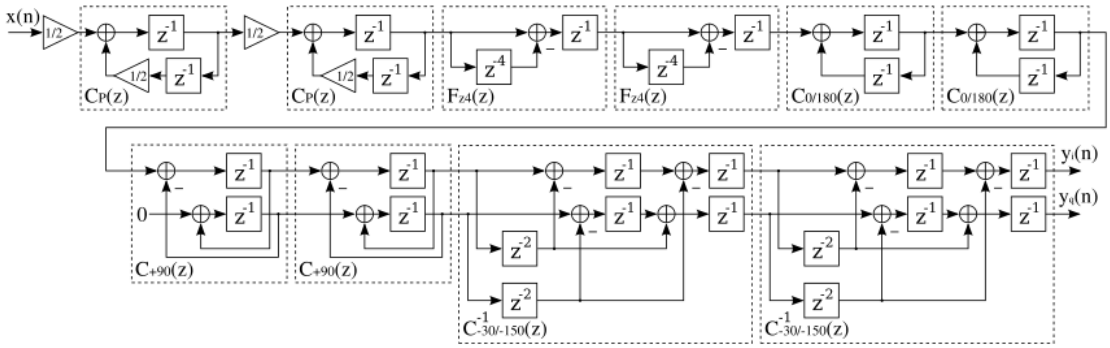


Figura 42: Diagrama de blocos do filtro de Hilbert.

Capítulo 5 – Implementação no DSP TMS320C6713s

Para além da implementação na FPGA ,o trabalho vai ser realizado no DSP TMS320C6713, que é um microcontrolador desenhado especialmente para processamento de sinal. Inicialmente , como foi referido anteriormente o recurso ao DSP deveu-se ao facto de a determinada altura se ter chegado a um impasse na FPGA relativamente ao recursos disponíveis nesta. Mais tarde o problema na FPGA foi resolvido através da arquitetura Distributed Arithmetic que solucionou o problema da escassez de multiplicadores, mas neste ponto já estava desenvolvida grande parte da aplicação no DSP, pelo que se viu vantagens em acabar de desenvolver o sistema em ambas as plataformas e comparar os resultados. Como é sabido este tipo de processamento é baseado em tipos específicos de operações como por exemplo:

- Filtragem e FFT
- Conversões entre domínios do tempo e frequência

Estas operações recorrem a multiplicações e adições recursivas, ou seja, o processamento de sinal baseia-se essencialmente na realização de operações MAC (multiplication and accumulate). Se para efetuar estas operações recorrer-se a um microprocessador padrão, que executa as multiplicações através de adições recursivas, vamos ter as operações MAC processadas por um excessivo número de operações de adição, o que vai fazer com que o processamento se torne bastante lento. É aqui que entra a grande vantagem do DSP que vem munido com unidades MAC otimizadas para este tipo de operações, que conseguem executar a mesma operação num único ciclo. Como exemplo um DSP de 150 MIPS pode processar aproximadamente 32 milhões de amostras de dados por segundo, sendo que num microprocessador normal este valor é reduzido em 3 milhões de amostras por segundo. Os DSP's à imagem dos microcontroladores normais vêm equipados com vários periféricos. O DSP usado é o TMS320C6713 que contêm um chip com o codec AIC23 que serve como ADC e DAC. Este comunica com o DSP através do protocolo SPI. A imagem que se segue ilustra a placa de expansão do DSP TMS320C6713 DSK.

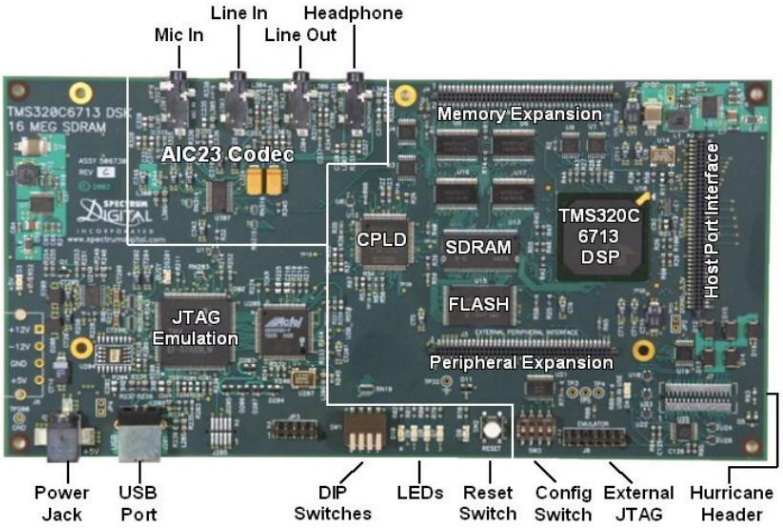


Figura 43: DSP TMS320C6713 DSK

5.1- Codec AIC23

Para efetuar as operações de in/out o DSP TMS320C6713 DSK usa uma driver equipada com o codec de áudio AIC23. O Codec facilita a integração do ADC e do DAC no mesmo chip, e contem 2 canais analógicos stereo, com frequência de amostragem de 8 a 96 kHz e a sua comunicação com o exterior realiza-se através de duas portas McBSP que transferem as amostras para uma porta série, sendo que a porta McBSP0 transfere os comandos e a porta McBSP1 transfere os dados. De seguida temos algumas das características do Codec AIC23.

- Dois canais analógicos stereo
- Pré-amplificação para microfone
- Ligação direta às colunas
- Frequências de amostragem configuráveis: 8,16, 24, 32, 44, 48, 96 kHz
- Quantização a 16, 20, 24, 32 bit

5.1.1-Registos e respetivas funcionalidades do Codec AIC23

- **Register 0:** Volume do canal esquerdo de entrada. Valor por defeito - 0x0017.
- **Register 1:** Volume do canal direito de entrada. Valor por defeito - 0x0017.
- **Register 2:** Volume do canal esquerdo dos headphones. Valor por defeito - 0x01F9.
- **Register 3:** Volume do canal direito dos headphones. Valor por defeito - 0x01F9.
- **Register 4:** Controlo do path do áudio analógico. Valor por defeito - 0x0011.
- **Register 5:** Controlo do path do audio digital. Valor por defeito - 0x0000.
- **Register 6:** Controlo do interruptor on/off. Valor por defeito - 0x0000.
- **Register 7:** Formato do interface digital de áudio. Valor por defeito - 0x0043.
- **Register 8:** Controlo do sample rate. Valor por defeito - 0x0081.
- **Register 9:** Ativação do interface digital. Valor por defeito - 0x0001.

Os registos 0,1 e 8 foram alterados de forma a otimizar a performance da aplicação, pois os seus valores por defeito não produziam os efeitos desejados. Os valores passaram a ser:

- **Register 0:** 0x001F de forma a aumentar a potência de saída do canal esquerdo.
- **Register 1:** 0x001F de forma a aumentar a potência de saída do canal direito.
- **Register 8:** 0x001E de forma a maximizar o sample rate.

O sample rate foi configurado com o valor 0x001E para que tanto o DAC como o ADC funcionassem a 96Khz segundo tabela que se segue:

ADC	DAC	SR3	SR2	SR1	SR0	BOSR
96	96	0	1	1	1	1
48	48	0	0	0	0	1
32	32	0	1	1	0	1
8	8	0	0	1	1	1
48	8	0	0	0	1	1
8	48	0	0	1	0	1

Figura 44: Valores possíveis do registo 8

5.1.2-Interface do codec AIC23

De forma ao DSP poder realizar operações de input/output, o codec AIC23 disponibiliza um interface que contém funções para a sua inicialização, configuração e operação. A lista que se segue enumera as funções disponibilizadas pelo interface do codec. Estas funções estão declaradas no ficheiro "dsk6713_aic23.h"

- **DSK6713_AIC23_openCodec()** Obter um handle para o CODEC
- **DSK6713_AIC23_closeCodec()** Libertar um handle
- **DSK6713_AIC23_config()** Configuração do CODEC – freq. de amostragem
- **DSK6713_AIC23_read()** Ler 32 bits de dados
- **DSK6713_AIC23_write()** Escrever 32 bits de dados
- **DSK6713_AIC23_setFreq()** Definir a frequência de amostragem
- **DSK6713_AIC23_rset()** Escrever num registo de controlo
- **DSK6713_AIC23_rget()** Obter o valor de um registo de controlo
- **DSK6713_AIC23_outGain()** Definir o ganho de saída
- **DSK6713_AIC23_loopback()** Activar/desactivar o modo de loop-back
- **DSK6713_AIC23_mute()** Activar/desactivar o modo de mute
- **DSK6713_AIC23_powerDown()** Activar/desactivar o modo powerdown

Para se realizar uma aplicação em que as amostras não sofram nenhum tipo de processamento, depois de se inicializar o codec, vai-se recorrer à função DSK6713_AIC23_read() de forma a permitir que uma variável leia o valor presente no registo de entrada do codec e seguidamente à função DSK6713_AIC23_write() para escrever o valor da variável no registo de saída do codec. Todo o processamento de sinal é efetuado entre estas duas operações.

Por fim temos a representação interna do DSPcodec TI V320AIC23 que mostra a sua implementação interna, e os seus blocos constituintes.

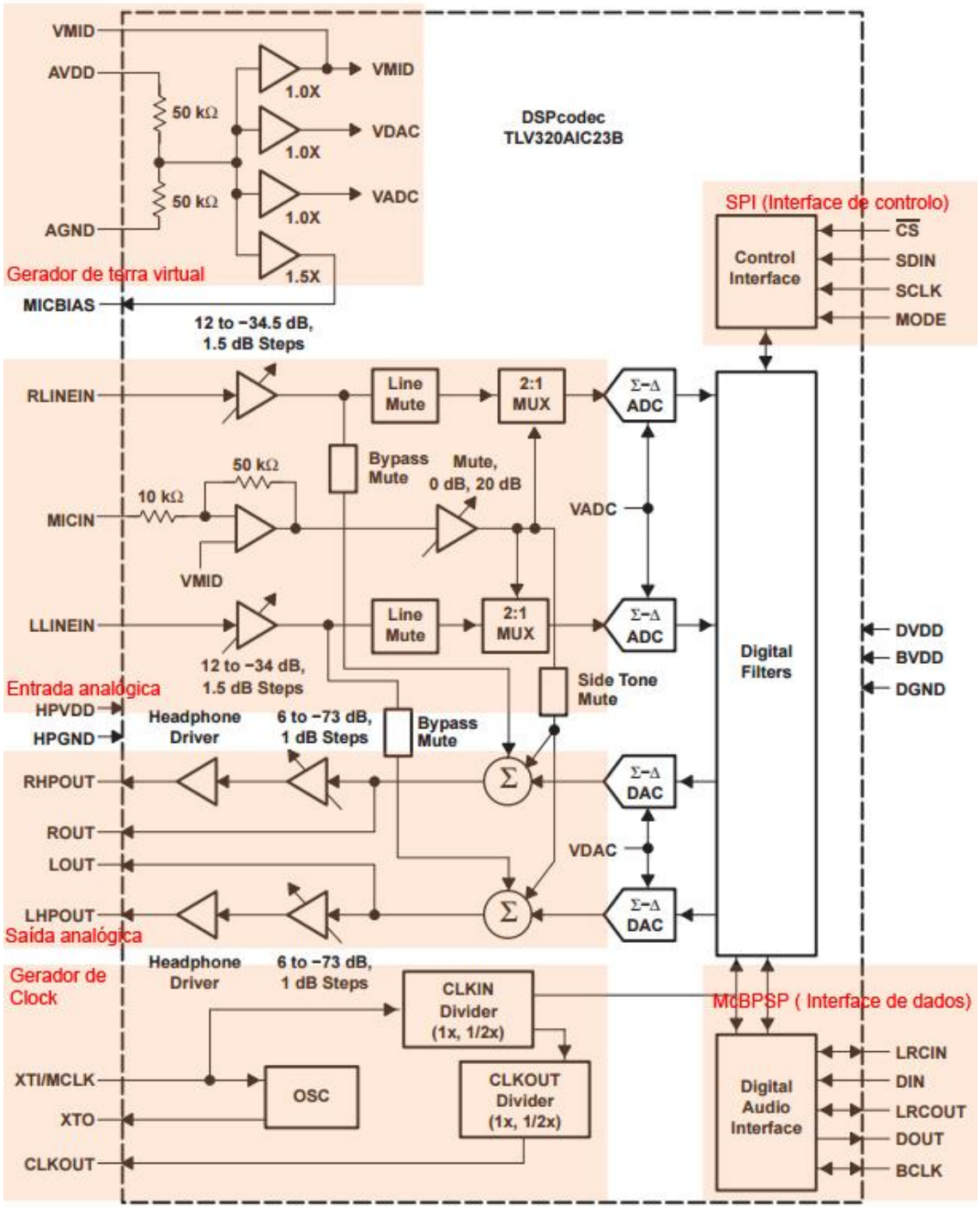


Figura 45: Blocos constituintes e implementação interna do codec AIC23

5.2- Implementação do código

A implementação do código no DSP vai ser realizada na linguagem C. Para tal, e como já foi referido anteriormente, vai ser usada a framework de desenvolvimento disponibilizada pela Texas Instruments designada por Code Composer.

Com as partes do planeamento concetual e simulação concluídas, é chegada a parte da implementação das arquiteturas em código. Esta parte acaba por envolver também o estado

debug da metodologia seguida como foi referido anteriormente, pois enquanto o código foi sendo desenvolvido, foram sempre tomadas precauções para efetuar debug ao código de forma a inferir se o caminho tomado era o mais indicado, e também de maneira a depurar o código com mais eficiência.

A estruturação do código divide-se modularmente de forma a estudar cada um dos casos isoladamente de forma a tirar conclusões acerca de cada um dos módulos mais pormenorizadamente. Os módulos são os seguintes:

1. Inicialização do DSP TMS320C6713
2. Portadora
3. Atraso
4. Filtro passa baixo
5. Filtro passa-alto
6. Filtro de Hilbert

A ter em conta que segundo a metodologia de trabalho descrita anteriormente a opção tomada foi efetuar a análise de cada um dos módulos no final da sua implementação, mas de forma a organizar o relatório de forma mais percetiva e eficaz, primeiro é referido o estudo da implementação de cada um dos módulos, para depois mostrar os resultados obtidos.

5.2.1-Inicialização do DSP TMS320C6713

Antes de se realizarem as operações necessárias para obter o sinal de áudio direcional à saída do sistema, é necessário inicializar o DSP de forma a este operar corretamente. Entre as configurações está a inicialização das bibliotecas de suporte da placa, a configuração do sample rate, a inicialização das interrupções, etc.

Para tal foram criadas as funções initialize() e sampling_rate (int rate) de forma a agrupar todas a ações que visam inicializar o DSP. A função sampling_rate tem somente por função configurar o sample rate do DSP, que no caso deste trabalho vai ser de 96Khz, que é o máximo permitido por esta placa.

```
void sampling_rate(int rate){
switch(rate){
    case 8  :  DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ); break;
    case 16  :DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_16KHZ); break;
    case 24  :DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_24KHZ); break;
    case 32  :DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_32KHZ); break;
    case 44  :DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_44KHZ); break;
    case 48  :DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ); break;
    case 96  :DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_96KHZ); break;
    default    DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_96KHZ);
}
```

Por sua vez a função initialize() tem à sua responsabilidade inicializar a placa ao integrar as bibliotecas de suporte do DSP, mapear as os vários tipos de interrupções, inicializar e configurar o codec AIC23.

```
void initialize()
{
	DSK6713_init();	// Inicializa as bibliotecas de suporte do DSP
	 hCodec = DSK6713_AIC23_openCodec(0, &config);
	MCBSP_FSETS(SPCR1, RINTM, FRM);
	MCBSP_FSETS(SPCR1, XINTM, FRM);
	MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
	MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
	IRQ_globalDisable();		// Desabilita as interrupções globalmente
	IRQ_nmiEnable();		// Ativa a interrupção NMI
	IRQ_map(IRQ_EVT_RINT1,15);// Mapear um evento para a interrupção
	IRQ_enable(IRQ_EVT_RINT1);// Habilita um evento
	IRQ_globalEnable();		// Habilita as interrupções globalmente
}
```

5.2.2-Portadora

Um ponto fundamental deste trabalho é a geração da portadora de ultrassons. Esta portadora consiste num cosseno com uma frequência de 40Khz. Para além disso também é necessário gerar também uma portadora consistida num seno também ele a 40Khz de forma a multiplicar pelo sinal filtrado com a transformada de Hilbert.

A portadora é gerada pela função wavGen que está listada de seguida:

```
float wavgen(int D, float *w, float A, float F, int *q)
{
	float y, c=D*F;
	y = A * w[*q];
	*q = qwrap(D-1, (int) (*q+c));
	return y;
}
```

A função wavGen recebe como parâmetros os seguintes campos:

```
int D   : Número de pontos da portadora.
float *w: Pontos definidos da portadora.
float A : sinal modulante que vai ser multiplicado pela portadora.
float F : Campo que vai determinar a frequência da portadora.
int *q  : Ponteiro que é incrementado através da função qwrap.
```

Tendo em conta os campos acima definidos, as variáveis que vão estar associadas à portadora que contêm o cosseno a 40Khz são as seguintes:

```
#define D 1000			// fmin = fs/D = 96000/1000 = 96 Hz
float coseno[D];	// wavetable buffer - coseno
short fs=96;
float fPortadora=40;
int wavPortadoraCoseno = 0;
```

Depois de definidas as variáveis foi criado o array de suporte para o cosseno:

```
//PORTADORA Coseno
for (i=0; i<D; i++) coseno[i] = cos(2*PI*i/D);
```

Finalmente temos a declaração da função que gera o cosseno. Neste caso, e de forma a visualizar o sinal no osciloscópio, o sinal de entrada a multiplicar pelo cosseno, é o inteiro estático 32768, para que se observe a portadora pura.

```
sigWavGenCos = wavgen(D, coseno, 32768, fPortadora/fs, &wavPortadoraCoseno);
```

Para gerar a portadora que contêm o seno os parâmetros que se alteram é o array seno que vai conter a sinusoide e o inteiro a ser incrementado de forma a percorrer o array.

```
float seno[D];            // wavetable buffer - seno
int wavPortadoraCoseno = 0;

//PORTADORA Seno
for (i=0; i<D; i++) seno[i] = sin(2*PI*i/D);
```

A declaração que vai gerar a portadora que contêm é seno é a segunte:

```
sigWavGenSen = wavgen(D, seno, 32768, fPortadora/fs, &wavPortadoraSeno);
```

A imagem que se segue ilustra a portadora gerada de 40Khz. Como se pode verificar o sinal está bastante distorcido, dado ao facto de estarmos com um sample rate de somente 96Khz.
A partir deste ponto surgem duas opções:

1. Baixar a frequência até atingir um sinal perfeito.
2. Continuar com este sinal e deixar a filtragem solucionar o problema, pelo menos a nível espectral.

5.2.3-Atraso

Existem duas formas de gerara o atraso necessário para sincronizar os samples:

- Buffer linear
- Buffer circular

No buffer linear a operação a realizar é a seguinte:

$$for\ i = D\ down\ to\ i = 1\ do:$$

$$w[i] = w[i-1]$$

Que em traduzido na linguagem c reflete a seguinte função:

```
// ----------------------------

void delay(int D, float *w)
{
int i;
for (i=D; i>=1; i--)
        w[i] = w[i-1];
}
// ----------------------------
```

Mas para valores muito elevados de D, este método torna-se inconveniente, pois envolve mudar grandes porções de dados de um ponto de memória para o outro, pelo que a melhor opção

nestes casos é ao invés de alterar os pontos de armazenamento de memória dos dados, somente alterar o endereço inicial do buffer em memória uma slot de memória para a esquerda.

Este conceito é a base do buffer circular e é este que vai ser usado de forma a otimizar os parcos recursos em memória presentes no DSP.

As slots de memória são percorridas por um ponteiro movível que devolve os seus estados através do seguinte algoritmo:

$$S_i = *(\mathrm{p} + \mathrm{i}), \mathrm{i} = 0, \dots, \mathrm{D}$$

Caso o ponteiro exceda os limites do array à direita, este dá a volta e retorna ao ponto inicial.

A imagem que se segue ilustra ambos os casos (buffer linear e circular), para dois estados temporais consecutivos, em que se tem um D=3. Como se pode reparar, ambas as slots de memória são acedidas através do algoritmo $S_i = *(\mathrm{p} + \mathrm{i}), \mathrm{i} = 0, 1, 2, 3$ sendo que no caso linear o ponteiro mantém-se fixo no endereço inicial do buffer, ou seja, p = w, e são os dados que são alterados de uma slot de memória para a outra, enquanto que no caso do buffer circular, é o ponteiro p que altera a sua posição para a esquerda, mantendo os dados na sua posição inicial.

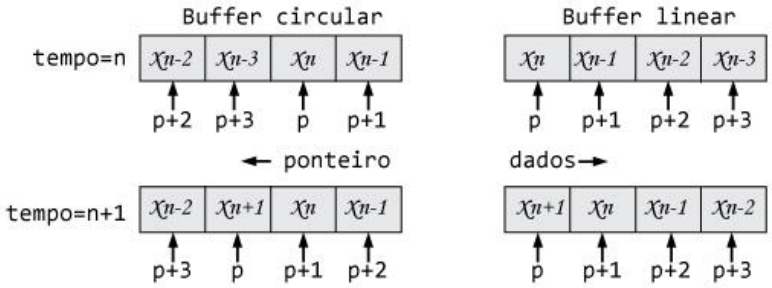


Figura 46: Arquiteturas dos buffers linear e circular

```
// ------------------------------------------------
float *bufferCircular(int D, float *w, float *p)
{
        if (p > w+D)
                p -= D+1;
        if (p < w)
                p += D+1;
        return p;
}
// ------------------------------------------------
```

Neste caso o estado da slot de memória e a atualização do ponteiro movível podem ser calculados da seguinte forma:

$$S_i = * \mathbf{bufferCircular}(\mathrm{D}, \mathrm{w}, \mathrm{p} + \mathrm{i}), \qquad \mathrm{i} = 1,2, \dots, D$$

$$P_{next} = \mathbf{bufferCircular}(\mathrm{D}, \mathrm{w}, --\mathrm{p})$$

Tendo em conta este tipo de mecanismo circular, é fácil traduzir um algoritmo de processamento de amostras em pseudo-código, para código C. A imagem que se segue contêm um diagrama de blocos e um pseudo-código que vão ser traduzidos para a linguagem C tendo em conta o buffer circular, e em especial a função bufferCircular.

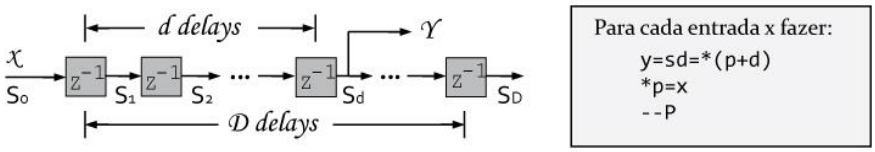


Figura 47: Diagrama de blocos e pseudo-código de processamento de amostras.

Passando para a linguagem C usando a função bufferCircular vamos ter:

```
y = * bufferCircular (D,w,p+d);        // output
*p = x;                                // input
p = bufferCircular (D,w,--p);          // Atualização do ponteiro
```

Resumidamente, e depois de terem sido efetuado estudos, testes e verificações, esta foi a forma mais otimizada e funcional de implementar todos os mecanismos que necessitassem de realizar atrasos na sua implementação, especialmente os filtros digitais presentes na aplicação, mas também na sincronização de amostras no filtro de Hilbert.

5.2.4-Filtragens

Como foi referido anteriormente, o sistema tem na sua arquitetura três filtros, que vão ajudar a limar o sinal, e a prepara-lo para a saída.

A ter em conta que no trabalho se está a lidar com ultrassons, o que vai fazer com que as frequências em jogo sejam mais elevadas que o normal, sendo necessário ter em atenção o tipo de filtros necessários para ter o resultado desejado na saída. Há um compromisso entre o resultado desejado, e as especificações do DSP TMS320C6713, especialmente ao nível da memória disponível. Os coeficientes dos filtros são todos gerados através de scripts realizados em MATLAB, e depois exportados para um C Header, através doutro script somente realizado para esse propósito.

As filtragens são realizadas por um filtro FIR ou seja, de resposta ao impulso finita. A principal característica deste tipo de filtro é que é um tipo de filtro digital caracterizado por uma resposta ao impulso que se torna nula após um tempo finito.

O filtro apresenta a seguinte relação de convolução:

$$y[k] = \sum_{n=0}^{N-1} h[n]x[k-n],$$

Onde:

$X(z) = TZ[x[n]] = \sum_{n=-\infty}^{+\infty} x[n]z^{-n}$ - é o sinal de entrada e

$H(z) = TZ[h[n]]$ - é a resposta impulsiva do filtro de duração finita igual a N amostras.

A saída também poderá ser dada pela fórmula:

$$y(n) = h_0x(n) + h_1x(n-1) + \ldots + h_Px(n-P)$$

Onde P é a ordem do filtro, x(n) o sinal de entrada, $y(n)$ o sinal de saída e h_i são os coeficientes do filtro.

A sua estrutura básica é:

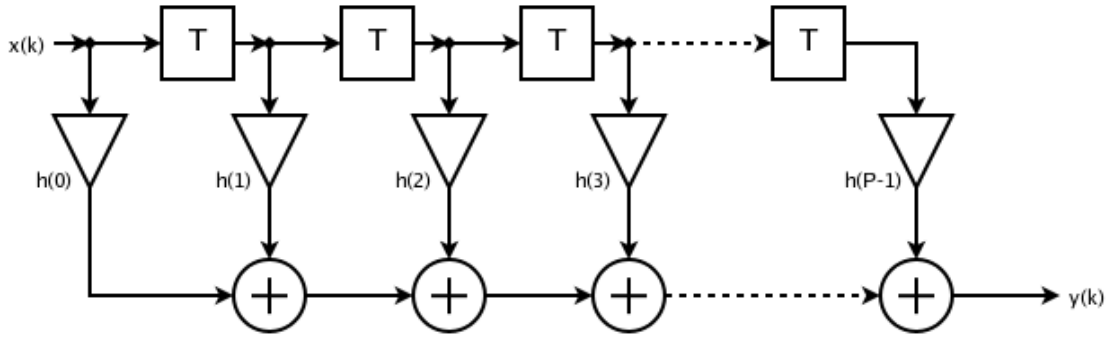


Figura 48: Estrutura interna do filtro FIR

Na estrutura anterior, pode-se visualizar as características internas do filtro FIR implementado, onde os termos h são os coeficientes e os T são os elementos de atraso.

Foram usados vários tipos de filtros, e foram testadas as performances de cada um deles, e por fim o escolhido foi o filtro que implementa um índex circular, ou seja, é um filtro que se aproxima mais da implementação de filtros FIR em hardware, em que evita que existam longas interações dentro do ciclo de repetição for, pois ao chegar ao final do buffer retorna o índex para q=0.

A implementação do filtro está listado a seguir:

```
float firq(int M, float *h, float *w, int *q, float x)
{
    int i, Q;
    float y;
    Q = M - (*q);               // number of sates to end of buffer
    w[*q] = x;                  // ler a amostra x
    for (y=0, i=0; i<=Q; i++)   // ciclo desde até ao fim do buffer
    y += h[i] * w[(*q)++];
    (*q) = 0;                   // apontar para o início do buffer
    for (i=Q+1; i<=M; i++)      // ciclo até q-1
    y += h[i] * w[(*q)++];
    (*q)--; if (*q == -1) *q = M;// andar o index uma posição para trás
    return y;
}
```

Os seus parâmetros são os seguintes:

- `int` M: Número de coeficientes.
- `float` *h: Vector com os coeficientes.
- `float` *w: Vector que vai conter as amostras.
- `int` *q: Index que vai permitir percorrer o vector com as amostras.
- `float` x: Variável que contém a amostra presente.

5.3 - Análise dos resultados dos módulos programados

Esta parte do relatório consiste na análise da performance e eficiência dos módulos programados. Como foi referido anteriormente, esta analise foi sendo feita à medida que os

módulos eram concluídos, mas de forma a juntar todos os resultados obtidos, foi criado este capítulo para somente servir esse propósito. Os módulos que vão ser analidados vão ser os seguintes:

1. Frequência de amostragem
2. Portadora
3. Sinal modelado
4. Filtro passa-baixo
5. Filtro passa-alto

5.3.1-Frequência de amostragem

De forma a compreender o ruído introduzido pelo DSP ao aumentar a frequência de amostragem, foram capturada visualizações do osciloscópio com diferentes frequências de amostragem. A frequência escolhida para efetuar a captura foi de um quarto da frequência de amostragem, ou seja, metade da frequência de Nyquist. Como se pode verificar, à medida que se vai aumentando a frequência de amostragem, vai sendo introduzido ruído, pelo que naturalmente, nos resultados finais vai haver a presença de algum ruído residual.

Fs=8Khz

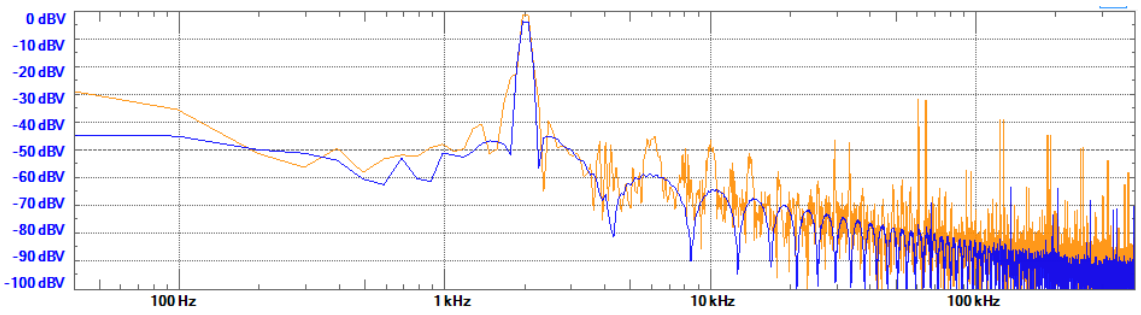


Figura 49: Código - Fs = 8Khz

Fs=32Khz

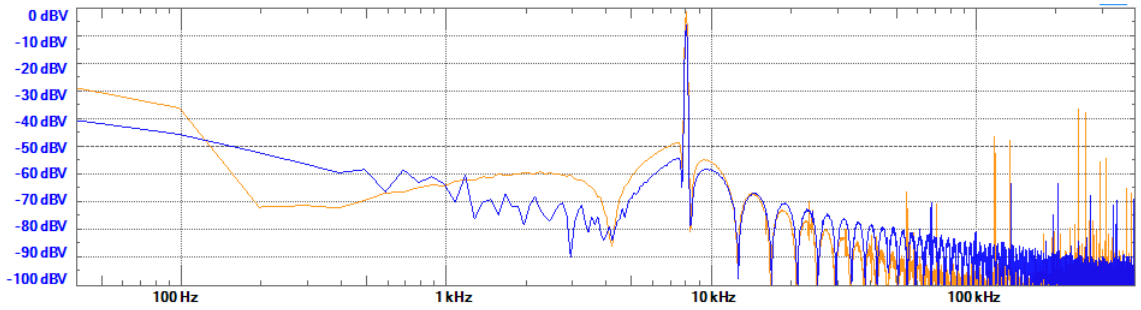


Figura 50: Código - Fs = 32Khz

Fs=44Khz

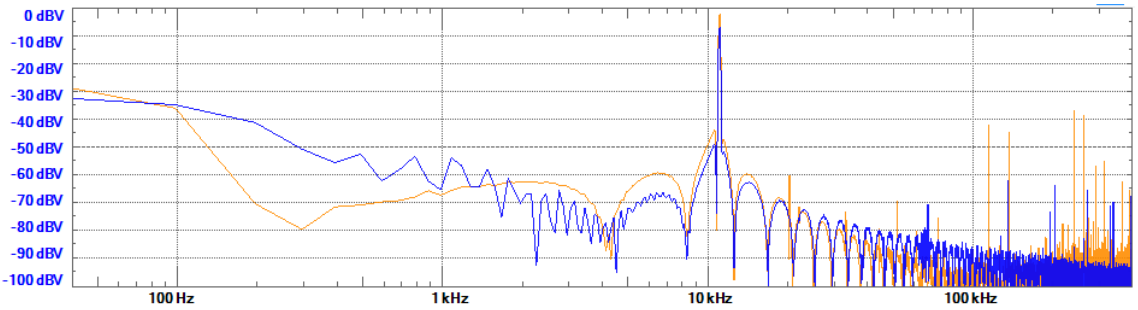


Figura 51: Código - Fs = 44,1Khz

Fs=48Khz

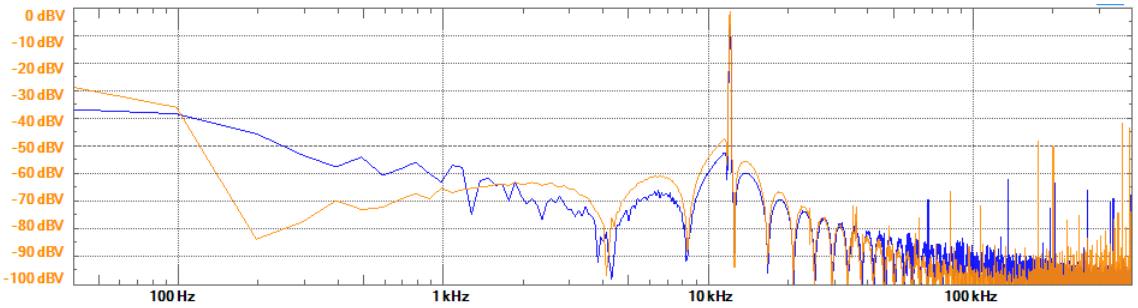


Figura 52: Código - Fs = 48Khz

Fs=96Khz

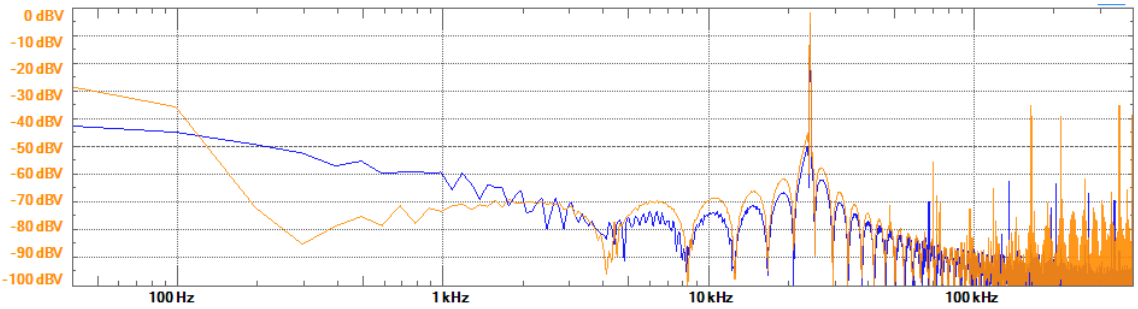


Figura 53: Código - Fs = 96Khz

5.3.2-Portadoras de ultrassons

Depois de efetuado o estudo sobre os efeitos da frequência de amostragem, foi realizada a análise da portadora isoladamente de forma a compreender o comportamento desta no DSP. O código realizado em C para efetuar este estudo foi o seguinte:

```
interrupt void isr()
{
    float sig, sigAmp, portadora;

    read_inputs(&xL, &xR);
    sig = (float) xL;
    sigAmp = sig;
    portadora = wavgen(D, coseno, 16384, fPortadora/fs, &wavPortadoraSeno);
    yL = (short) portadora;
    write_outputs(yL,yL);
    return;
```

}

A função wavegen é usada também para a modulação, pelo que o parâmetro de entrada 16384 corresponde ao parâmetro que fornece o sinal a ser modelado. Neste caso o valor é uma constante de forma a se ter somente a portadora com um valor constante.
A imagem que se segue ilustra a captura da portadora pelo osciloscópio:

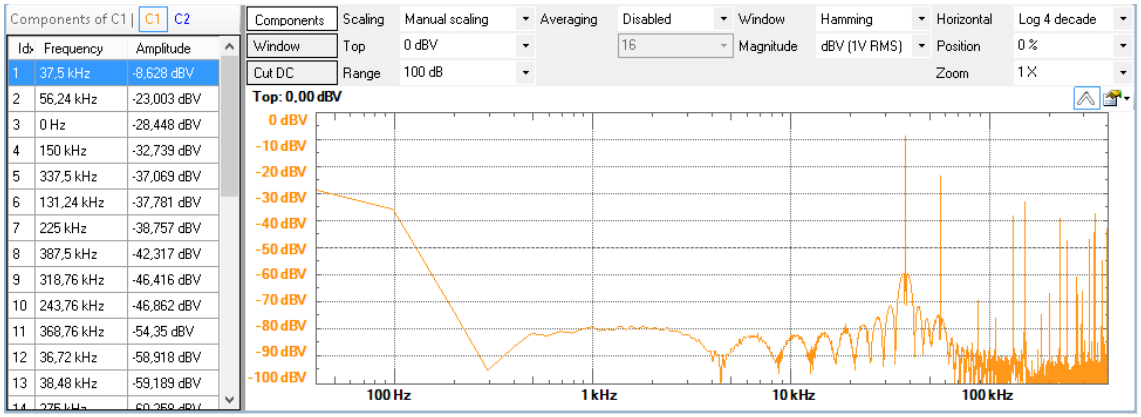


Figura 54: Geração da portadora a 40Khz.

Como se pode verificar existem dois componentes espectrais que se sobrepõem sobre os outros, e dois outros componentes mais baixos:

1	**37,5Khz**	**-8,628 dBV**
2	56,24Khz	-23,003 dBV
3	0Khz	-28,448 dBV
4	150Khz	-32,739 dBV

Tabela 1:Componentes espectrais da portadora

A segunda componente com um valor mais elevado apresenta uma frequência centrada nos 56Khz. O gráfico que se segue explica o fenómeno:

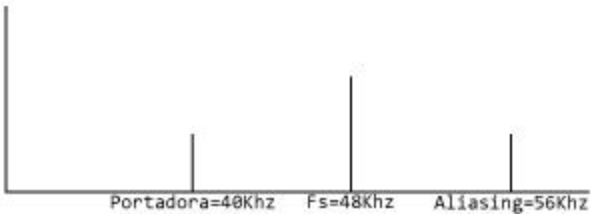


Figura 55: Aliasing

De notar que a frequência central da portadora é de 37,5Khz apesar de ter sido declarada uma variável com um valor de 40Khz. O fenómeno deve-se ao facto de termos poucos pontos a representar a portadora. Esse fenómeno é explicado de seguida.
Ao se visualizar a portadora no domínio do tempo verifica-se que esta apresenta um comportamento irregular. Tal deve-se ao facto desta ser somente representada por dois pontos, pois a frequência de amostragem é de 96Khz e a frequência da portadora é de 40Khz.
Para calcular o número de pontos específico pode-se efetuar o seguinte cálculo:

$$\frac{Frequência\ de\ amostragem}{Frequência\ da\ portadora} = Número\ de\ amostras\ da\ portadora$$

$$\frac{96000}{40000} = 2{,}4$$

Como se pode verificar o número de amostra é de 2,4 em cada ciclo, o que indica que em alguns dos ciclos vamos ter 2 pontos enquanto em outros ciclos vamos ter 3 pontos.

5.3.3-Sinal modelado

Depois de analisado o comportamento e eficiência da portadora, passou-se à análise do sinal modelado. Inicialmente o sinal de teste foi uma sinusoide gerada pelo gerador de sinais disponibilizado pelo Analog discovery. O sinal tem uma amplitude de 1V e uma frequência de 10Khz. O código realizado para efetuar este teste foi o seguinte:

```
interrupt void isr(){
    float sig, sigAmp, sigPbd;

    read_inputs(&xL, &xR);
    sig = (float) xL;
       sigAmp = sig;
       sigPbd = firq(Mpbd,hpbd,passaBanda, &q,sigAmp);
    yL = (short) sigPbd;
    write_outputs(yL,yL);
    return;
}
```

A imagem que se segue representa a captura do sinal modelado com a sinusoide:

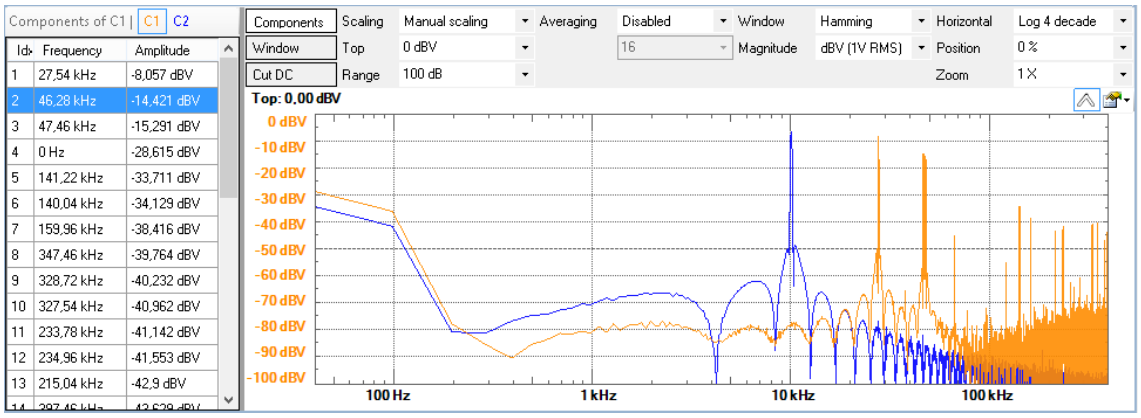


Figura 56: Captura do sinal modelado em DSB.

1	**27,54Khz**	**-8,057 dBV**
2	46,28Khz	-15,291 dBV
3	0Khz	-28,615 dBV
4	141,22Khz	-33,711 dBV

Tabela 2:Componentes espectrais do sinal modelado

Como se pode verificar, as componentes espectrais mais fortes são as centradas nas frequências de 27 e 46Khz. Os valores esperados seriam de 30 e 40Khz, mas este aspeto não é de estranhar, atendendo ao facto da portadora tomar o valor de 37Khz como foi verificado anteriormente. A imagem que se segue traduz estes resultados:

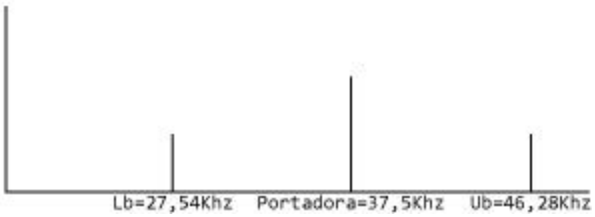


Figura 57: Modulação DSB.

A imagem que se segue ilustra a modulação DSB com um sinal de composto por ruído branco gerado no programa Adobe Audition.

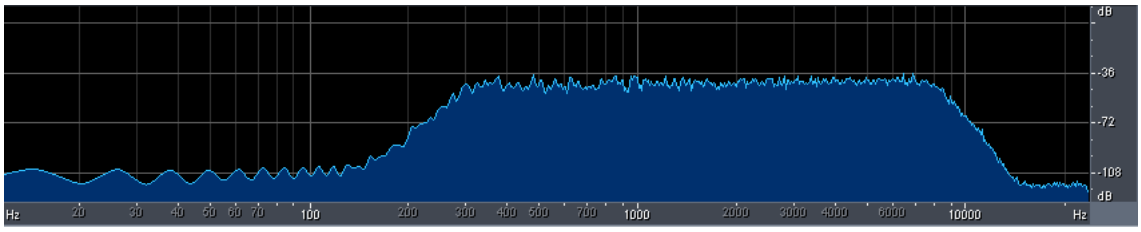


Figura 58: Sinal de teste de ruído branco gerado no Adobe Audition

Como se pode verificar o sinal está filtrado com um filtro passa-banda com frequências de corte compreendidas entre os os 300Hz e os 8Khz e com uma potência de -36dB.

A imagem que se segue representa a captura no osciloscópio do sinal modelado com este sinal de teste:

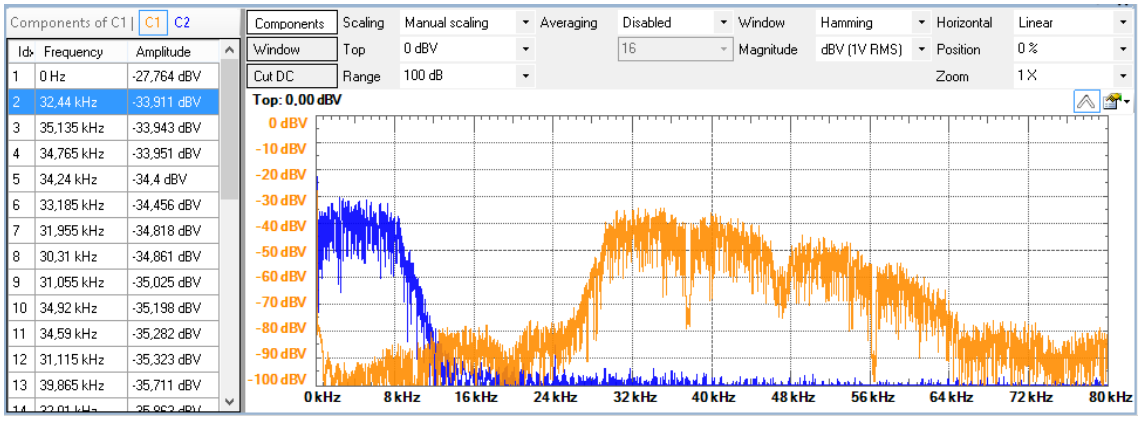


Figura 59: Modulação DSB com ruído branco com Fc1=300Hz e Fc2=8Khz como sinal de teste

Como se pode verificar na imagem, os resultados obtidos estão de acordo com o esperado. Tem-se a portadora centrada nos 37,5Khz e ambas as componentes espectrais com 8Khz de largura de banda. Também é visível o fenómeno de aliasing, pois o sinal é espelhado nos 48Khz, que é a frequência de Nyquist do sistema. De notar que a banda lateral superior sofre uma ligeira atenuação ao se aproximar da frequência de Nyquist, o que é de todo esperado, dado ao facto do parte do sinal se sobrepor à frequência limite. Esta captura também revelou um dado importante, que indica que o sinal quando proveniente da fonte sonora do computador apresenta sinais de potência ligeiramente mais baixos, atendendo ao facto que tanto o volume do computador como o volume da aplicação estavam no máximo. Esta constatação revelou que era necessário introduzir ou um sistema de amplificação do sinal no código, ou um sistema de pré-amplificação do sinal antes deste entrar no DSP.

Finalmente, e dado ao facto que este é um sistema DSB-WC somou-se a portadora de forma a verificar o resultado. A imagem que se segue representa uma modulação com um sinal de 5Khz conjuntamente com a adição da portadora.

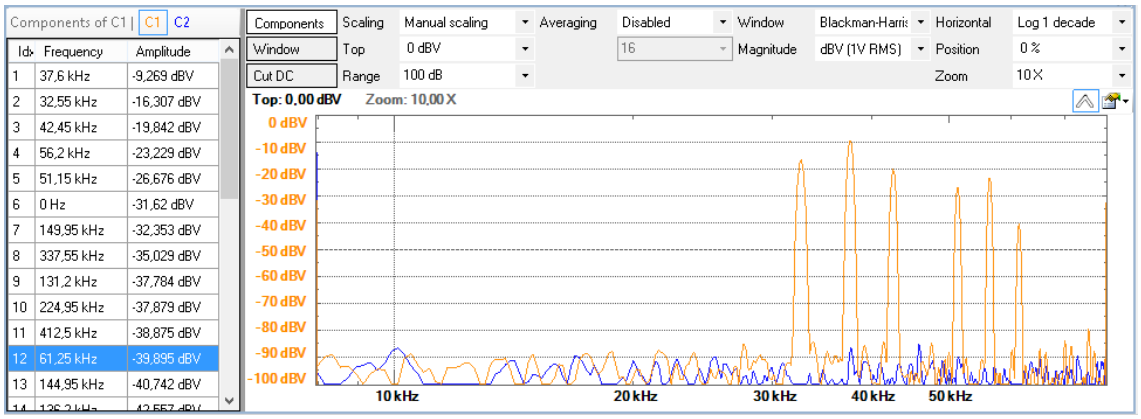


Figura 60: Modulação com um sinal de 5Khz conjuntamente com a adição da portadora.

Como se pode verificar os resultados estão de acordo com o esperado. Como já foi referido anteriormente convém que a portadora tenha sensivelmente mais 10dBs que o sinal, o que se confirma.

5.3.4-Filtro passa-baixo

As filtragens são uma parte crucial do sistema, pelo facto de serem uma peça fundamental na redução do ruído e da distorção, como também pelo facto de serem os módulos mais sensíveis a destabilizações devido a este sistema estar a trabalhar no seu limite.
O primeiro filtro a ser abordado é o passa-baixo com uma frequência de corte de 8Khz para que no limite somente deixe permitir que passem as frequências relativas a voz.
O código da interrupção para este teste foi o seguinte:

```
interrupt void isr(){
    float sig, sigAmp, sigPb;

    read_inputs(&xL, &xR);
    sig = (float) xL;
       sigAmp = sig;
       sigPb = firq(M,h,passaBaixo, &t,sigAmp);
    yL = (short) sigPb;
    write_outputs(yL,yL);
    return;
}
```

De forma a obter os coeficientes foram criados dois scripts em Matlab que conjuntamente geram um ficheiro com os coeficientes conforme os parâmetros de entrada [Frequência de amostragem, Número de coeficientes]. O código do script que gera os coeficientes foi o seguinte:

```
function Cn = coeficientesPassaBaixo(Fs,fc)
    N = 20;
    Q = (N -1)/2;
    C0=fc/(Fs/2);
```

```
    Cn=zeros(0,length(N));
    j=1;
    for n=-Q:Q
        Cn(j) = (C0)*sinc((C0)*n);
        j = j+1;
    end
    GerarFicheiroCoeficientes(Cn)
    figure(1);
    plot(Cn)
    freqz(Cn)
end
```

Como se pode verificar é chamada a função GerarFicheiroCoeficientes(Cn) que tem como parâmetro de entrada o array que contém os coeficientes. Este ficheiro foi realizada de forma a simplificar o trabalho, pois retira ao utilizador a necessidade de criar o ficheiro manualmente, declarando as variáveis necessárias e colocando as vírgulas entre os termos do array. O formato devolvido pela função é o seguinte.

```
#define N 15
float h[N] = {
1.2480E-004,-8.3333E-005,1.2514E-004,1.3787E-001,-2.5000E-004,-
1.2514E-004,-8.3333E-005,1.2480E-004
};
```

Número coeficientes = 15 Fc=15Khz

```
#define N 15
float h[N] = { 2.5263E-002,-2.0302E-002,-6.2439E-002,-5.6270E-002,2.0700E-
002,1.4704E-001,2.6466E-001,3.1250E-001,2.6466E-001,1.4704E-001,2.0700E-002,-
5.6270E-002,-6.2439E-002,-2.0302E-002,2.5263E-002};
```

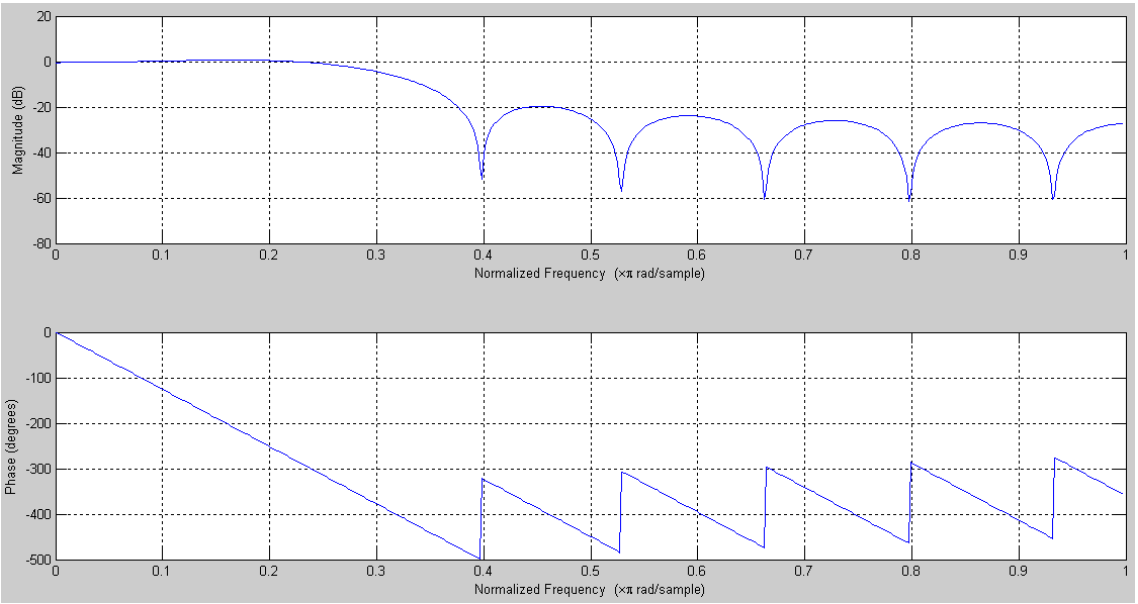


Figura 61: Fase e magnitude do filtro passa-baixo de 15 coeficientes.

Número de coeficientes=15 Fc=15Khz V=735mV

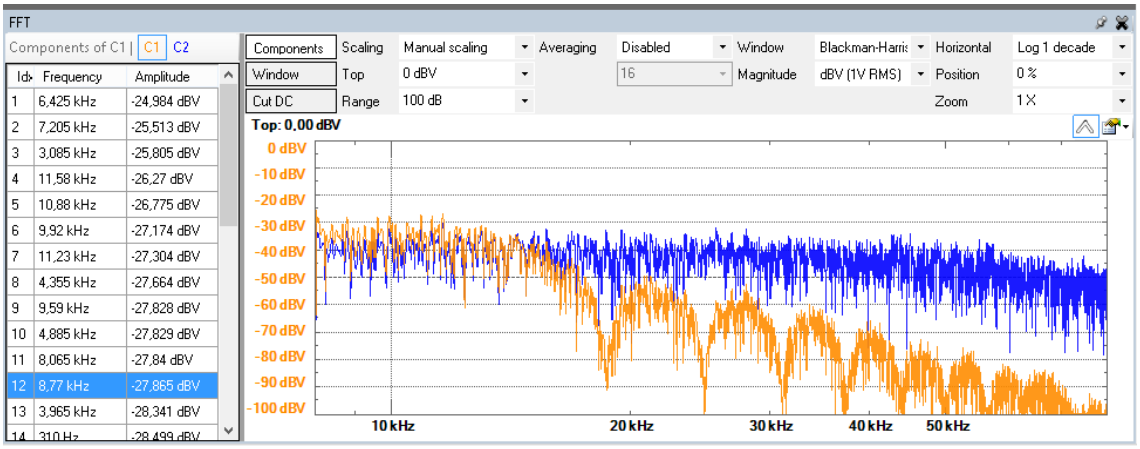


Figura 62: Captura do filtro passa-baixo de 15 coeficientes.

5.3.5-Filtros passa-banda

Quanto aos filtros passa-banda, foram usados dois distintos conforme a arquitetura usada necessitasse de ficar somente com a banda lateral superior ou com a banda lateral inferior.

No caso de ser usada a banda lateral inferior, as frequências de corte foram as seguintes:

- Fc1: 32Khz
- Fc2:40Khz

Caso a arquitetura recorre-se à banda lateral superior, as frequências de corte foram:

- Fc1: 40Khz
- Fc2:48Khz

De seguida está listado o código usado na interrupção de forma a somente processar a filtragem:

```
interrupt void isr(){
    float sig, sigAmp;
    read_inputs(&xL, &xR);
    sig = (float) xL;
       sigAmp = sig;
       sigPbd = firq(Mpbd,hpbd,passaBanda, &q,sigAmp);
    yL = (short) sigPbd;
    write_outputs(yL,yL);
    return;
}
```

À imagem do filtro passa-baixo, também aqui foram dois scripts desenvolvidos em Matlab de forma a gerar os coeficientes. O script listado em baixo foi o usadopara gerar os coeficientes. COmo se pode verificar o script chama a função GerarFicheiroCoeficientes(Cn) que como foi referido anteriormente tem o objetivo de criar o C Header com os coeficientes. Esta função pode ser consultada nos anexos.

```
function Cn = coeficientesPassaBanda(fc , Fs)
    N = 15;
    Q = (N -1)/2;
    C0=fc/(Fs/2);
    C1=(fc*2)/(Fs/2);
    Cn=zeros(0,length(N));
    j=1;
    for n=-Q:Q
        Cn(j) = (C1)*sinc(C1*n)-(C0)*sinc(C0*n);
        j = j+1;
    end
    GerarFicheiroCoeficientes(Cn)
    figure(1);
    plot(Cn)
    freqz(Cn)
end
```

Número de coeficientes=15 Fc1=25Khz Fc2=40Khz

```
#define Mpbd 15
float hpbd[Mpbd] = { 1.5586E-002,2.2861E-002,-3.1644E-002,-8.6980E-
002,2.1027E-001,-1.1952E-001,-1.5565E-001,3.1000E-001,-1.5565E-001,-1.1952E-
001,2.1027E-001,-8.6980E-002,-3.1644E-002,2.2861E-002,1.5586E-002
}
```

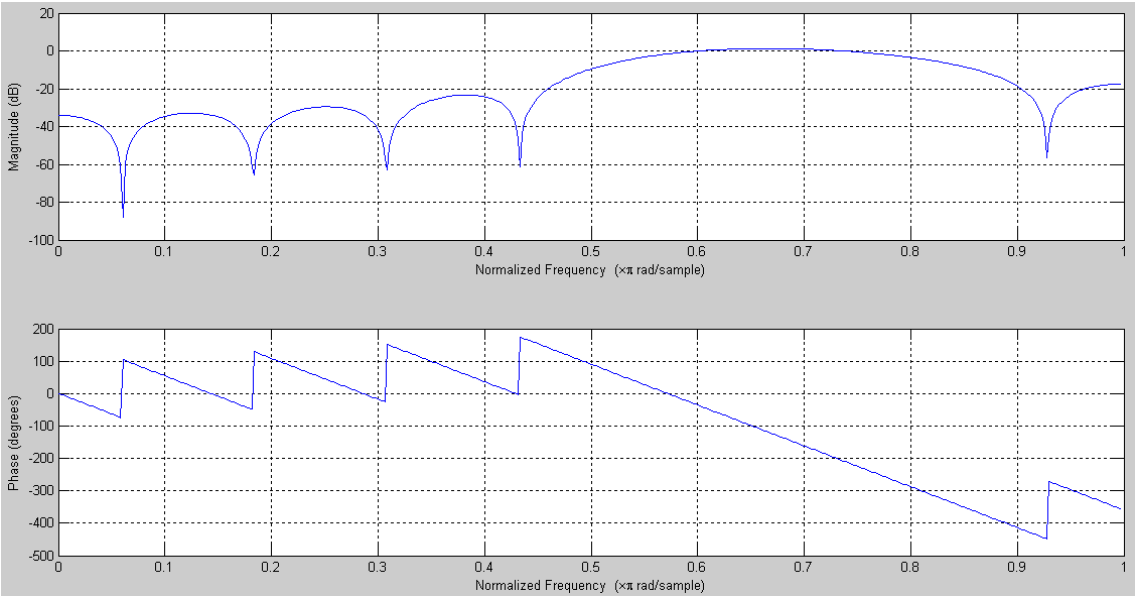


Figura 63: Fase e magnitude do filtro passa-banda com Fc1=25Khz Fc2=40Khz de 15 coeficientes.

Número de coeficientes=15 Fc1=25Khz Fc2=40Khz

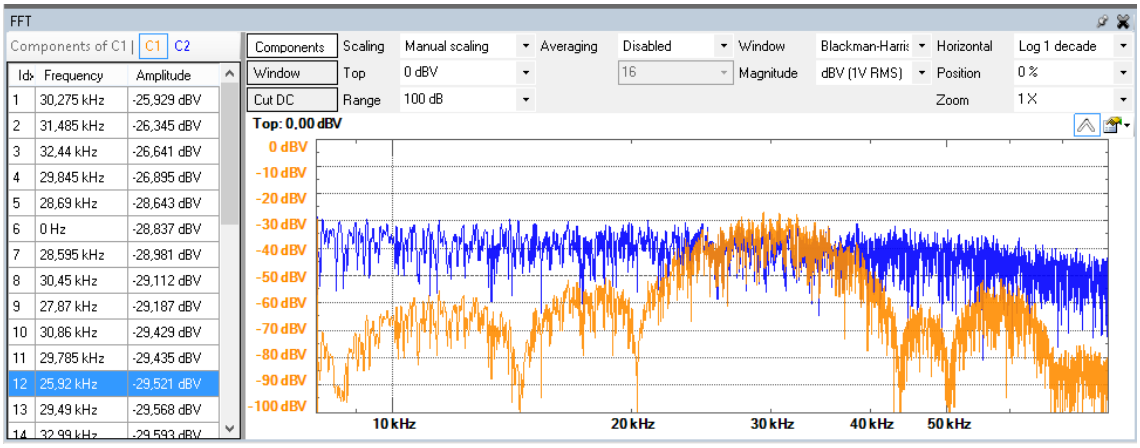
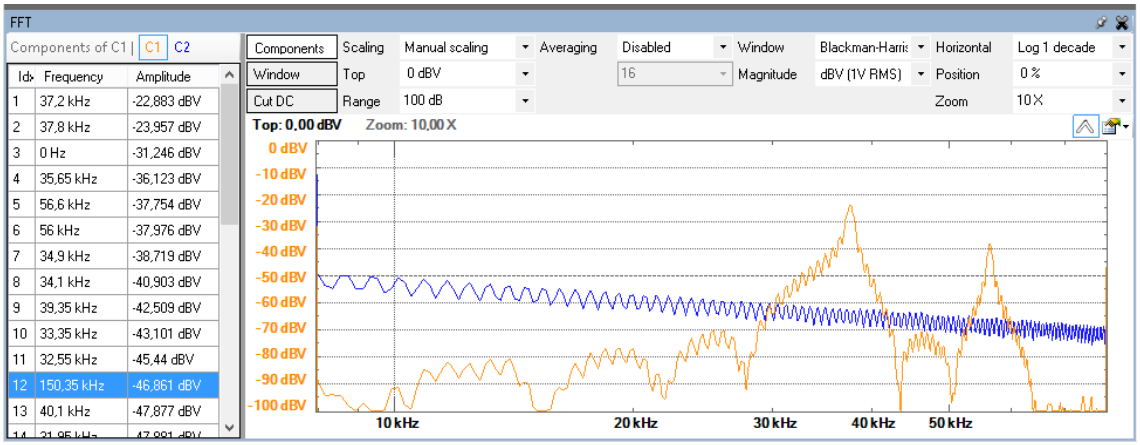


Figura 64: Captura do filtro passa-banda com Fc1=25Khz Fc2=40Khz de 15 coeficientes.

5.4-Sistema de ultrassons

Por fim o sistema foi testado com todos os seus módulos ativos de forma a ver qual o seu comportamento. Nesta parte do trabalho importa dizer que os teste já são com a presença da coluna de parametric arrays de forma a inferir os resultados a nível sonoro. Num primeiro passo os sinais de teste introduzidos foram sinusoides e ruído branco.

```
interrupt void isr() {
        float sig, sigAmp, sinalModelado;
        float sigPb, sigPbd, portadora;
        float DSSWC;
        float output;
        float sigPreProc;

        read_inputs(&xL, &xR);
        sig = (float) xL;
        sigAmp = sig * ganho1;
        sigPb = firq(M, h, passaBaixo, &t, sigAmp);
        portadora = wavgen(D, coseno, 16384, fPortadora / fs, &wavPortadoraSeno);
        sinalModelado = wavgen(D, coseno, sigPb, fPortadora / fs, &wavPortadoraCoseno);
        sigPbd = firq(Mpbd, hpbd, passaBanda, &q, sinalModelado);
        DSSWC = portadora + sigPbd;
        output = DSSWC * ganho2;
        yL = (short) output;
        write_outputs(yL, yL);
        return;
}
```



Como se pode verificar o sinal apresenta uma boa resposta em frequência relativamente à introdução de ruído. Na imagem também aparece a imagem espelhada derivada do aliasing. Ao testar o sistema com a coluna, este garantiu uma boa diretividade, especialmente com sinusoides como sinal de teste. Também esteve presente algum ruído derivado não só ao próprio tipo de sistema, mas também derivado ao facto de como foi referido anteriormente somente se ter dois pontos para representar a sinusoide em cada um dos ciclos. Na imagem também aparece a portadora no topo do sinal. Esta tem sensivelmente 10dB a mais do que o sinal a transmitir o que é esperado, dado ao facto de estar na norma e deste ser um sistema DSB-WC com uma das bandas filtradas.
Apesar do nível sonoro não ter um valor muito elevado, o sinal diretivo ouve-se claramente a sensivelmente oito metro de distância quando tem uma sinusoide como sinal de teste; para além disso, o sinal apresenta a propriedade de se refletir ao atingir um obstáculo, como seria de esperar, pois esta é uma característica deste tipo de sistemas.

5.5-Processos de redução da distorção

Finalmente foram efetuados os testes relativos à redução de distorção do sistema. Como foi referido anteriormente, à que ter em conta que a operação quadrática é o factor que mais contribui para a distorção harmónica, pelo que foi aplicado o sistema SRAM (Square Root Amplitude Modulation) que consiste em realizar a operação de raiz quadrada ao sinal modulante de forma a pré-distorcer a envolvente no sistema DSB-WC. A imagem que se segue representa a arquitetura do sistema seguido:

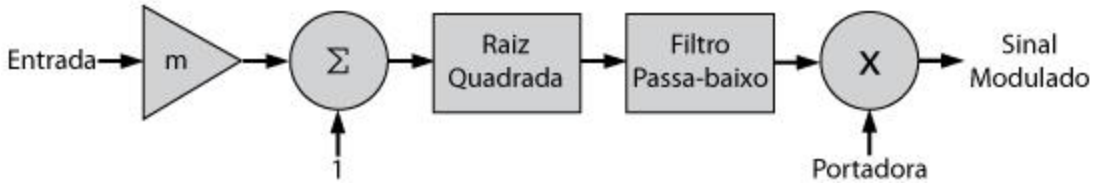


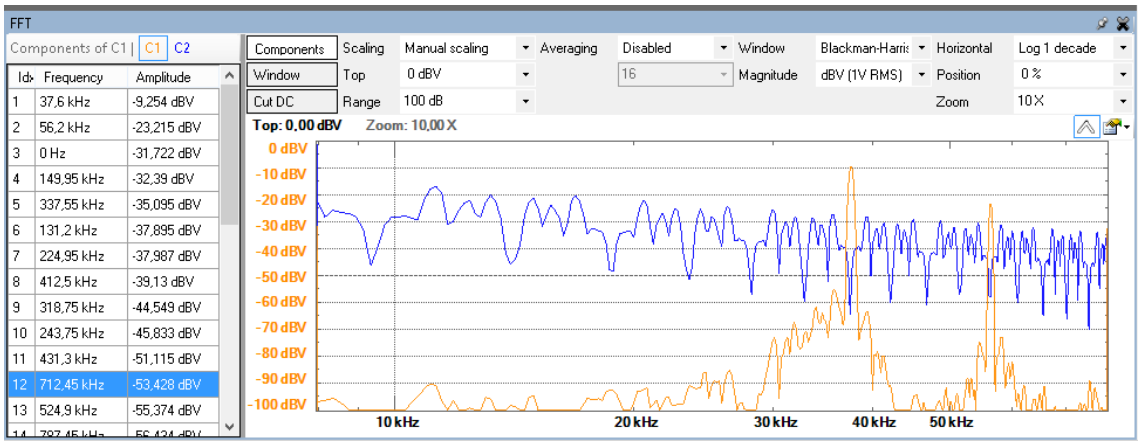
Figura 65: Arquitetura SRAM

O Código implementado para a arquitetura SRAM foi o seguinte:

```
interrupt void isr() {
        float sig, sigAmp, sinalModelado;
        float sigPb, sigPbd, portadora;
        float DSSWC;
```

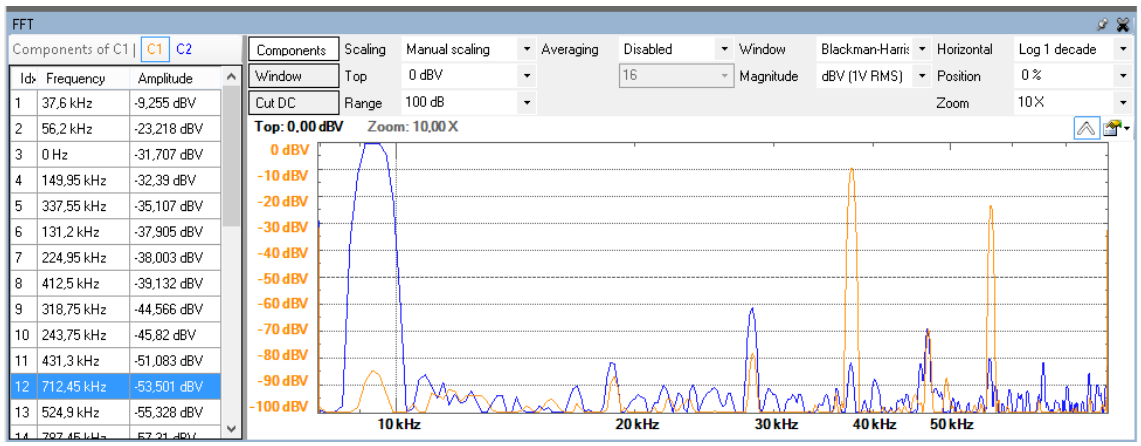
```
        float output;
        float sigPreProc;

        read_inputs(&xL, &xR);
        sig = (float) xL;
        sigAmp = sig * ganho1;
        sigPb = firq(M, h, passaBaixo, &t, sigAmp);
        sigPreProc = sqrt(sigPb + 1);
        portadora = wavgen(D, coseno, 16384, fPortadora / fs, &wavPortadoraSeno);
        sinalModelado = wavgen(D, coseno, sigPreProc, fPortadora / fs, &wavPortadoraCoseno);
        sigPbd = firq(Mpbd, hpbd, passaBanda, &q, sinalModelado);
        DSSWC = portadora + sigPbd;
        output = DSSWC * ganho2;
        yL = (short) output;
        write_outputs(yL, yL);
        return;
}
```

A distorção melhorou muito ligeiramente, mas como se pode verifica pela figura, o nível do sinal baixou significativamente, fazendo com que a portadora tivesse um valor de potência muito acima do sinal, bem mais dos que os 10dBs definidos pela norma. O resultado esteve de acordo com o esperado, mas teve de se aumentar a amplitude do sinal, de forma a que a potência sonora seja idêntica a quando não se aplica o sistema SRAM, pelo que foi necessário adicionar uma camada de amplificação externa, pois ao se efetuar a operação internamente, o ruído e a distorção aumentavam também significativamente.



Id	Frequency	Amplitude
1	37,6 kHz	-9,254 dBV
2	56,2 kHz	-23,215 dBV
3	0 Hz	-31,722 dBV
4	149,95 kHz	-32,39 dBV
5	337,55 kHz	-35,095 dBV
6	131,2 kHz	-37,895 dBV
7	224,95 kHz	-37,987 dBV
8	412,5 kHz	-39,13 dBV
9	318,75 kHz	-44,549 dBV
10	243,75 kHz	-45,833 dBV
11	431,3 kHz	-51,115 dBV
12	712,45 kHz	-53,428 dBV
13	524,9 kHz	-55,374 dBV

Figura 66: Aplicação do algoritmo SRAM com ruído branco



Id	Frequency	Amplitude
1	37,6 kHz	-9,255 dBV
2	56,2 kHz	-23,218 dBV
3	0 Hz	-31,707 dBV
4	149,95 kHz	-32,39 dBV
5	337,55 kHz	-35,107 dBV
6	131,2 kHz	-37,905 dBV
7	224,95 kHz	-38,003 dBV
8	412,5 kHz	-39,132 dBV
9	318,75 kHz	-44,566 dBV
10	243,75 kHz	-45,82 dBV
11	431,3 kHz	-51,083 dBV
12	712,45 kHz	-53,501 dBV
13	524,9 kHz	-55,328 dBV

Figura 67: Aplicação do algoritmo SRAM com uma sinusóide de 10Khz

Capítulo 6: Comparação de resultados, Limitações e Investigação Futura

Por fim, é chegado o capítulo de desfecho da dissertação que tem como objetivo concluir o trabalho apresentando as conclusões obtidas relativas às várias etapas de execução da dissertação a nível teórico e prático. Como primeiro ponto é realizada uma comparação de resultados, comparando essencialmente os resultados obtidos em casa uma das plataformas onde o sistema foi implementado. Também são abordadas as limitações encontradas ao longo do trabalho. Estas foram sendo descritas ao longo da execução prática do sistema, mas agora, com uma visão mais ampla do sistema na sua totalidade, é mais fácil perceber quais os pontos que precisam de ser mais trabalhados, de maneira a se formular como as Investigações futuras devem ser inicializadas, de forma a se dar continuação a este trabalho.

6.1-Comparação de resultados

A comparação de resultados aqui abordados referem-se à comparação dos testes realizados na FPGA Spartan 3E e no DSP TSM320C6713.

O sistema funcionou de acordo com o esperado em ambos os sistemas, sendo que no DSP não foi possível implementar:

- Arquitetura 1 com o filtro passa-banda com uma banda passante dos 40 aos 60kHz.
- Arquitetura 2, referente à filtragem de Hilbert.

Em ambos os casos, não foi possível implementar o sistema dado ao facto de se estar limitado pela frequência de amostragem, de 96kHz. Atendendo que a portadora tem uma frequência de 40kHz, ficam somente a sobrar 8khz de largura de banda para a banda superior.

Relativamente à implementação da arquitetura 1 com o filtro passa-banda com uma banda passante dos 20 aos 40kHz o sistema implementado no DSP apresenta uma ligeira diminuição na potência sonora emitida relativamente ao implementado na FPGA.

Relativamente aos custos de implementação temos dois fatores a ser comparados:

- O tempo de execução
- Custos do hardware envolvido

Ao nível do tempo de execução, pode-se afirmar co toda a certeza que a implementação no DSP é significativamente mais rápida e acessível tanto ao nível de projeto como de implementação. A implementação na FPGA mostrou ser bastante mais morosa, com muitos pormenores técnicos a ser abordados de forma a se conjeturar a melhor forma de se implementar um sistema robusto e com o menor custo possível.

6.2-Limitações

A nível do DSP a maior limitação foi ter o sistema restrito a uma frequência de amostragem de 96kHz. O facto de ser ter somente uma largura de banda de 48kHz, vai fazer com que somente se possa usar a banda lateral inferior da modelação DSB-WC, pois a portadora tem uma frequência de 40kHz o que nos deixa somente 8kHz de largura de banda na banda lateral superior. Na FPGA uma das limitações mais relevante é o facto do ADC e do DAC serem periféricos SPI, tendo de partilhar várias portas entre si, sendo umas delas o relógio, tendo este de ser sincronizado de forma a fornecer o sinal de relógio a ambos os componentes. Para agravar a situação já de si preocupante desta limitação, o relógio do ADC e do DAC funcionam a frequências diferentes. Enquanto o relógio do DAC tem uma frequência de 25MHz O ADC tem uma frequência bastante mais baixa, de 12,5MHz. O facto destes dois componentes importantíssimos para a arquitetura do sistema terem clocks diferentes, é em si só um gerador de ruído, mas de facto a intenção aqui era de ter a frequência de amostragem o mais elevada possível, e ponderando ambos os aspetos, optou-se pela frequência de amostragem um pouco mais elevada, em detrimento do ADC e do DAc terem uma velocidade de relógio idêntica.

A imagem que se segue ilustra o relógio SPI em três estados diferentes. O primeiro está associado à frequência mais baixa, de configuração do amplificador, de seguida está o estado de captura no ADC e por ultimo o estado de amostragem do DAC. Como se pode verificar todos eles apresentam velocidades distintas.

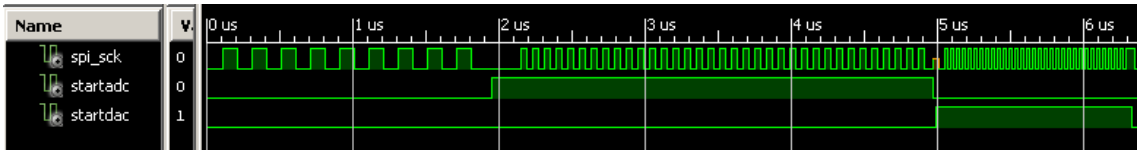


Figura 68: Relógio SPI

6.3-Investigação futura

Para investigação futura, é necessário investigar no mercado qual a melhor plataforma para desenvolver o sistema, de forma a não ser necessário ter a preocupação de ter de se implementar um sistema de captura e amostragem robusto, que no fundo foi onde foi empregue a maior parte do tempo de implementação do sistema, e mesmo assim no fim continuou-se com alguns problemas de ruído devido a estes módulos não estarem desacoplados do desenvolvimento do processamento propriamente dito. Na pesquisa pode-se enveredar por dois caminhos distintos, um francamente menos dispendioso monetariamente do que o outro. Optar por implementar o sistema na FPGA Spartan 3E e procurar ADC's e DAC's externos. Esta opção é válida no sentido em que se pode colocar ambos os componentes a correr paralelamente, cada um deles com uma velocidade de clock distinta, caso seja necessário, mas preferencialmente com a mesma velocidade. Neste sistema tem de se ligar o ADC e o DAC à FPGA pelas portas externas, o que pode ser um fator gerador de ruído.

A outra opção é procurar tanto FPGA's como DSP's de forma a encontrar a plataforma ideal. Esta forma, apesar de mais dispendiosa, é a que no final vai apresentar melhores resultados pelo facto da plataforma ter já os periféricos incorporados, pelo que não é necessário recorrer ás portas externas.

Capítulo 7 – Conclusão

O propósito deste trabalho foi alcançado, pois o objetivo principal, era o de implementar um sistema Áudio Spotlight que processasse áudio em tempo real. Inicialmente começou a ser implementado na FPGA Spartan 3E, sendo que quando se chegou à parte da implementação das filtragens, o recursos começaram a esgotar-se por completo, especialmente os multiplicadores, e a determinada altura o trabalho começou a ser implementado na DSP TMS320C6713 de forma a salvaguardar a implementação do sistemas numa das plataformas disponibilizadas, pois, nesta plataforma o número de multiplicadores não é problema, e a gestão destes muito mais abstrata. Posteriormente chegou-se à conclusão que se pode projetar produtos e operações MAC (multiply and accumulate) através da técnica Distributed Arithmetic (DA) que permite alcançar os mesmos resultados sem recorrer a multiplicadores, o que permitiu implementar também o sistema Áudio Spotlight na FPGA.

Conclui-se então que o que provou inicialmente ser um contratempo deveras limitador, tornou-se no fim uma grande vantagem, dado ao facto que foi possível aferir qual o melhor sistema para implementar o Áudio Spotlight, assim como foi possível comparar os resultados em dois sistemas completamente distintos, em vez de um. Porém na implementação no DSP, houve um problema que foi transversal a todas as fases de implementação, que foi a frequência de amostragem de 96Khz. Esta frequência, apesar de estar dentro da frequência de Nyquist para a portadora, limitou o uso de reprodução de áudio até aos 8Khz, para além de se ter na melhor das hipóteses somente dois pontos para representar o sinal à saída já modelado, o que fez com que o sinal apresenta-se algum ruído adicional, e a presença de harmónicos indesejáveis. Já na FPGA foi possível alcançar uma frequência de amostragem de 220Khz, que se revelou bastante mais satisfatório, especialmente depois das filtragens, pois regra geral, verificou-se neste tipo de sistemas que é os filtros são os locais em que é gerado um nível de ruído e harmónicos mais elevado.

A implementação no DSP mostrou-se bastante menos complexa, sendo a parte da filtragem a mais trabalhosa, mas a curva de aprendizagem é neste sistema bastante mais acessível, pelo que o número de horas para alcançar o sistema final foi bastante mais curto do que no período alcançado na FPGA, exatamente com o mesmo sistema. Em ambas as plataformas foram implementadas 3 arquiteturas distintas:

- SSB através de uma filtragem passa-banda da componente espetral superior
- SSB através de uma filtragem passa-banda da componente espetral inferior
- SSB através de uma filtragem de Hilbert

Enquanto na FPGA os melhores resultados ao nível da perceção e amplitude do sinal se encontraram na filtragem de Hilbert, esta demonstrou-se um grande desafio na implementação realizada no DSP. A ter em conta que para se alcançar uma boa filtragem de Hilbert, é necessário usar um número bastante considerável de coeficientes de forma a ser ter um deslocamento de fase nítido e com uma boa definição. Foi por esta última razão que não houve não se realizou a arquitetura com a filtragem de Hilbert no DSP TMS320C6713 DSK.

Bibliografia

[1] *Joel Preto Paulo, "Parametric Loudspeaker array", ISEL - Instituto Superior de Engenharia de Lisboa, 2 de setembro de 2011*
[2] *F.A. Karnapi * , W.S. Gan, Y.K. Chong, "FPGA implementation of parametric loudspeaker system", in Microprocessors and Microsystems, Volume 28, no. 5–6, pp. 261–272, 2 de agosto de 2004*
[3] *Woon-Seng Gan, Jun Yang b , Tomoo Kamakura, "A review of parametric acoustic array in air", 3 de abril de 2012, in Applied Acoustics, Volume 73, no. 12, pp 1209-1302, dezembro de 2012*
[4] *Pompei, F. Joseph (Frank Joseph), "Sound from ultrasound : the parametric array as an audible sound source", Massachusetts Institute of Technology. Dept. of Architecture. Program in Media Arts and Sciences, 27 de junho de 2003*
[5] *Chuang Shi, Woon-Seng Gan, "A preprocessing method to increase high frequency response of a parametric loudspeaker", "Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific", pp: 1 - 5, DOI: 10.1109/APSIPA.2013.6694373, outubro de 2013*
[6] *Martin Kumm, "Hilbert Transformator IP Cores", In International Conference on Field Programmable Logic and Applications 2008, pp. 251–256, 27 de dezembro de 2008*
[7] *V. Angelov, "Digital Signal Processing (VHDL Vorlesung)", 2009*
[8] *Marthworks, "Filter Design HDL Coder™ User Guide 2", 2010*
[9] *Mark S. Manalo and Ashkan Ashrafi, "Implementing Filters on FPGAs" - Department of Electrical and Computer Engineering Real-Time DSP and FPGA Development Lab*
[10] *Mathwork Release 10.1, "System Generator for DSP Getting Started Guide", março de 2008*
[11] *Ee-Leng Tan, Woon-Seng Gan, Peifeng Ji e Jun Yang, "Distortion Analysis and Reduction for the Parametric Array", School of Electrical & Electronic Engineering, Nanyang Technological University, 639798, Singapore, Conference: Audio Engineering Society 124th Convention, maio de 2008*
[12] *Yishu Wang, "Implementation of digital filter by using FPGA", CURTIN University of Technology Department of Electrical and Computer System Engineering, 10 de junho de 2005*
[13] *OmkarCK CAD Lab, "Using the ChipScope Pro for Testing HDL Designs on FPGAs", Dept of E&ECE, IIT Kharagpur*
[14] *Satyaki Mascharak, Arghyapriya Choudhuri, "Implementation of the onboard ADC and DAC on the Spartan 3E FPGA platform", Department of Electronics and Communication Engineering National Institute of Technology, Rourkela*
[15] *XILINX SPECS, "ISE In-Depth Tutorial" - UG695 (v14.1) 24 de abril de 2012*
[16] *Thomas D. Kite*, John T. Posty, and Mark F. Hamiltonz, "Parametric Array in Air: Distortion Reduction by Preprocessing",The Journal of the Acoustical Society of America, Vol. 103, no. 5, junho de 1998*
[17] *Wen-Kung Tseng, "Design of a Beam-forming System in the Vehicle Cabin", International Journal of Engineering Science and Innovative Technology (IJESIT),Volume 2, Issue 6, novembro de 2013*
[18] *Wen-Kung Tseng, "Acoustic Beam Forming Using Ultrasonic Transducers", Advanced Engineering Forum, Vol. 4, pp. 238-242,junho de 2012*
[19] *P. Hong, "The Audio Spotlight: An Alternative Approach" - IMTC, Georgia Institute of Technology, 2 de fevereiro de 2006*
[20] *Martin Kumm, M. Shahab Sanjari, "DIGITAL HILBERT TRANSFORMERS FOR FPGA-BASED PHASE-LOCKED LOOPS", Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on Heidelberg, pp. 251 - 256, 8 a 10 de setembro de 2008*

[21] *Peter Westervelt, "Parametric Acoustic Array", Journal of the Acoustical Society of America, Vol. 35, No. 4 (535-537), 1963*
[22] *Tianwen YANG, Dagui HUANG, Min CHEN, "Influences of Amplitude of Input Signal Frequency Components on Self-demodulated Signal of Parametric Sound Source", Journal of Computational Information Systems 9: 20 (2013)*
[23] *Helmholtz, "Combination_tones_in_violins", [in German: Uber Combinationst¨one ¨], Ann. Phys. Chem., 99, 497–540, 1856*