FILTRO ANTI-ALIASING PARA SISTEMA DE AQUISIÇÃO SINCRONIZADA IMPLEMENTADO EM FPGA

Paulo Gentil Gibson Fernandes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: Maurício Aredes

Mariane Rembold Petraglia

Rio de Janeiro

Março de 2011

FILTRO ANTI-ALIASING PARA SISTEMA DE AQUISIÇÃO SINCRONIZADA IMPLEMENTADO EM FPGA

Paulo Gentil Gibson Fernandes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

_____

Prof. Maurício Aredes, Dr.-Ing.

_____

Prof. Mariane Rembold Petraglia, Ph.D.

_____

Prof. Luís Guilherme Barbosa Rolim, Dr.-Ing.

_____

Prof. José Eduardo da Rocha Alves Junior, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2011

Fernandes, Paulo Gentil Gibson

Filtro Anti-aliasing para Sistema de Aquisição Sincronizada Implementado em FPGA/ Paulo Gentil Gibson. – Rio de Janeiro: UFRJ/COPPE, 2011.

IX, 103 p.: il.; 29,7 cm.

Orientadores: Maurício Aredes

Mariane Rembold Petraglia

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia Elétrica, 2011.

Referências Bibliográficas: p. 74-76.

1. Filtro Anti-aliasing. 2. Aquisição Sincronizada. 3. FPGA I. Aredes, Maurício et al. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Dedicatória

Dedico a aqueles que acreditaram ser possível a realização deste trabalho, em especial a minha família.

Agradecimentos

Agradeço à minha família por acreditar em mim e fornecer apoio nos momentos de dedicação a esta obra. Minha mãe, meu pai e sua esposa, minha namorada, meus irmãos e amigos, todos são muito importantes por me apoiarem nesta obra.

Agradeço à minha orientadora Mariane Rembold Petraglia por ser sempre receptiva, atenciosa, me apoiar e cumprir seu papel com dedicação e carinho.

Agradeço ao meu orientador Maurício Aredes por fornecer as bases necessárias para criar este trabalho, seu apoio e ajuda nesta obra foram indispensáveis.

Agradeço aos professores Guilherme Rolim e José Eduardo da Rocha Alves Junior por contribuírem em manter a qualidade deste trabalho ajudando no processo de avaliação.

Agradeço ao meu amigo Daniel Mendes Fernandes por me ajudar e participar dos mais variados desafios que apareceram, não só na produção desta obra, mas também durante todo o desenvolvimento do Sistema SCADA Harmônico que realizamos juntos.

Agradeço ao amigo Juliano Freitas Caldeira, ao professor Juarez Bastos Monteiro, ao professor Jorge Leão e a toda a equipe do laboratório LEMT e da COPPE/UFRJ por contribuírem para a produção desta obra.

Por fim, agradeço à Deus por ser Ele o autor da realidade em que vivemos.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

FILTRO ANTI-ALIASING PARA SISTEMA DE AQUISIÇÃO SINCRONIZADA IMPLEMENTADO EM FPGA

Paulo Gentil Gibson Fernandes

Março/2011

Orientadores: Maurício Aredes
Mariane Rembold Petraglia

Programa: Engenharia Elétrica

Devido a problemas de sobrecarga nos filtros de harmônicos não-característicos da subestação (SE) de Ibiúna, um sistema de medição sincronizada das correntes necessita ser instalado. Neste sistema, o processamento das formas de onda das correntes digitalizadas em diferentes pontos da SE identifica as origens das componentes harmônicas. Nas unidades remotas de medição (UASs), as especificações do filtro *anti-aliasing* são bastante exigentes devido à necessidade de sincronismo das aquisições e à complexidade do processamento dos dados. Este trabalho propõe uma combinação de filtros analógico e digital e utiliza o *hardware* de um FPGA para formar uma solução eficiente do filtro *anti-aliasing*.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the Degree of Master of Science (M.Sc.)

ANTI-ALIASING FILTER FOR SYNCHRONIZED ACQUISITION SYSTEM IMPLEMENTED BY FPGA

Paulo Gentil Gibson Fernandes

March/2011

Advisors: Maurício Aredes
Mariane Rembold Petraglia

Department: Electric Engineering

Due to overloading problems on the non-characteristic filters of the Ibiúna substation (SE), a synchronized measurement system of the harmonic current needs to be installed. In this system, the processing of the current waveforms digitized in different points at the SE identifies the origins of harmonic components. In the remote measurement units (UASs), the specifications of the anti-aliasing filter are quite demanding due to the need of the acquisitions synchronicity and the data processing complexity. This paper proposes a combination of analog and digital filters and uses a FPGA hardware to form an efficient anti-aliasing filter.

Índice

Capítulo 1 Introdução

O avanço tecnológico vem permitindo a utilização de dispositivos eletro-eletrônicos cada vez mais sofisticados e eficientes no consumo dos recursos energéticos para desempenharem suas tarefas. Entretanto, estes circuitos necessitam, em geral, de uma complexidade maior e se apresentam para a rede de distribuição de energia elétrica como cargas não-lineares. Um exemplo são as lâmpadas incandescentes, cargas puramente lineares, que estão sendo substituídas pelas lâmpadas fluorescentes, não-lineares.

As cargas não-lineares têm como característica a introdução de harmônicos que podem ser entendidos como deformações periódicas nas formas de onda das tensões e correntes da rede elétrica. Assim, um aumento na utilização das cargas não-lineares provoca um aumento dos harmônicos que poluem a rede e reduzem a qualidade da energia dos consumidores.

Para mitigar este problema reduzindo a presença dos harmônicos, filtros devem ser utilizados em conjunto com estes equipamentos. Os filtros são circuitos que possuem impedâncias diferentes para freqüências diferentes e com isso tem a capacidade drenar de forma seletiva as correntes harmônicas nas freqüências desejadas.

Esses filtros podem ser construídos para funcionar tanto dentro dos equipamentos que emitem a poluição, quando fora deles. Por exemplo, as lâmpadas fluorescentes podem já ser fabricadas com os filtros. Por outro lado, filtros de maior potência podem ser construídos nas subestações para melhorar a qualidade da energia da rede elétrica. Estes filtros, em geral, estão em paralelo com a rede e não discriminam a origem dos harmônicos.

Será visto neste trabalho que a subestação de Ibiúna, responsável por receber a energia que vem de Itaipu através de uma transmissão em corrente-contínua (CC), vem sofrendo graves problemas com o aumento da poluição harmônica.

Nomeadamente, um aumento expressivo da corrente de 5º harmônico no sistema de 345 kV.

Como uma forma de monitorar e controlar este problema em Ibiúna, um projeto de P&D com Furnas S/A, responsável pela subestação, foi proposto. Este projeto, chamado de "Sistema SCADA para Monitoramento em Tempo Real da Propagação Harmônica em Estações Conversoras HVDC", está sendo desenvolvido no laboratório LEMT da COPPE/UFRJ e encontra-se em sua fase final.

Este sistema tem como objetivo medir os harmônicos em diferentes pontos da subestação para identificar as fontes geradoras da poluição. Essa tarefa é pouco trivial por exigir a necessidade de sincronizar as aquisições de corrente nos pontos de medições concentrados nas casas de relés dos filtros CA e salas de controle dos bipolos e controle principal da SE Ibiúna.

No Capítulo 4, veremos que a unidade digital de medição de harmônico, chamada de UAS, requer a presença de um filtro *anti-aliasing* no processo de digitalização das amostras de corrente. A exigência de sincronismo restringe os requisitos deste filtro, dificultando o seu projeto. Este trabalho tem como objetivo buscar uma solução eficiente para o filtro *anti-aliasing* destas unidades de digitalização.

No Capítulo 2, explicamos o problema da subestação e a origem do sistema "SCADA Harmônico". No Capítulo 3, apresentamos o funcionamento deste sistema e verificamos suas necessidades de projeto. No Capítulo 4, analisamos o filtro *anti-aliasing* analógico. No Capítulo 5, refletimos sobre as soluções para a filtragem digital. No Capítulo 6, a implementação do filtro em FPGA é apresentada. No Capítulo 7, propomos uma metodologia de validação do filtro digital desenvolvido e apresentamos os resultados obtidos. No Capítulo 8, concluímos e citamos algumas formas de continuidade deste trabalho.

Capítulo 2 Contexto

A hidrelétrica de Itaipu é uma usina binacional situada no rio Paraná que está na fronteira do Brasil com o Paraguai. Um acordo firmado entre os dois países na década de 70 permitiu a construção da usina que é a maior produtora de energia no mundo, com a potência de 14.000 MW. Das 20 turbinas que estão em funcionamento, 10 são do Brasil, funcionando na freqüência de 60 Hz, e 10 são do Paraguai que funcionam em 50 Hz. Como o Paraguai é um país pequeno, consumindo menos energia do que produz, o restante desta produção é comprada pelo Brasil. Itaipu é responsável por fornecer 90% da energia consumida pelo Paraguai e 19% da energia consumida pelo Brasil [1].

Para transmitir a energia produzida pela parte brasileira (7.000 MW), três linhas de transmissão em 765 kVac corrente-alternada (HVAC – *High Voltage Alternating Current*), são utilizadas. Para transmitir a energia comprada, o sistema em corrente contínua (HVDC – *High Voltage Direct Current*) foi adotado, pois os requisitos de conversão de freqüência e tamanho da linha (800 km) tornam a relação custo-benefício atraente, quando comparada às linhas em corrente alternada (CA).

A vantagem da transmissão em corrente-contínua, em relação à transmissão em corrente-alternada, esta na sua eficiência. As linhas de transmissão em CC são mais compactas e necessitam de um investimento menor.

Além disso, a transmissão HVDC facilita a conexão de sistemas assíncronos. Como o sistema de geração (50 Hz) está em uma freqüência diferente do sistema receptor (60 Hz), a conversão em CC permite a conexão de ambos os sistemas com facilidade. O sistema HVDC também é melhor para transmissão em longas distâncias, pois não há a necessidade de compensar reativos.

Por outro lado, uma transmissão em HVDC exige que conversores de potência e filtros estejam presentes para permitirem a compatibilidade com o sistema HVAC. Então, o custo dos conversores de potência e seus filtros devem ser considerados

também na construção do sistema CC. Para o caso de linhas de curta distância o aporte de investimento pode ser maior do que para o sistema de corrente-alternada. Assim, a linha de transmissão de um sistema HVDC deve ser longa o suficiente para os custos dos equipamentos de conversão e dos filtros associados seja atraente para o investimento. A Fig. 2.1 mostra um diagrama do sistema de transmissão de energia de Itaipu para o sudeste do Brasil, que é formado por dois bipolos de corrente-contínua e três linhas de transmissão em corrente-alternada.

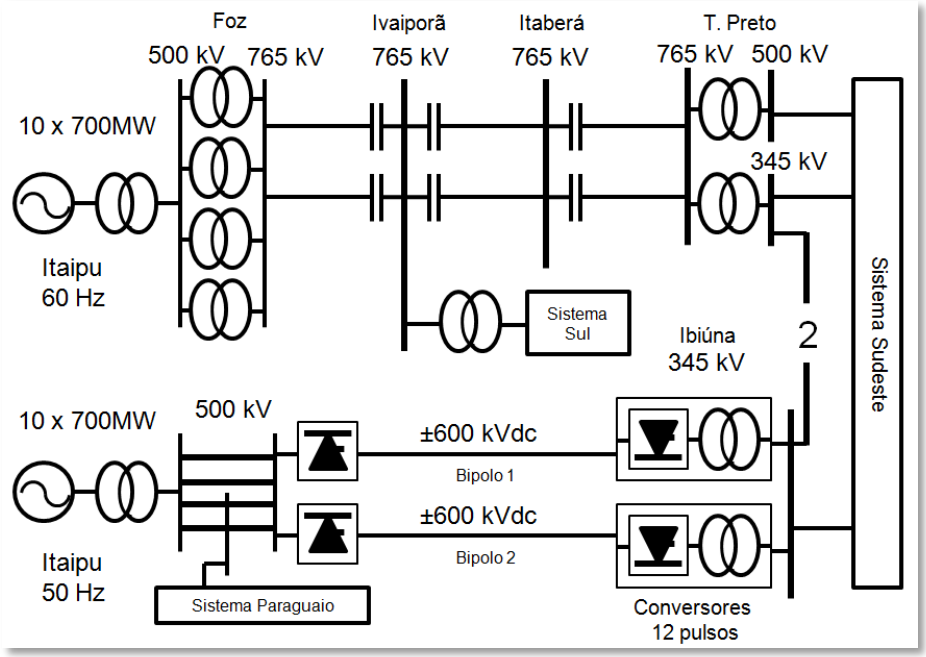


Figura 2.1 - Diagrama do sistema de distribuição de Itaipu para a Região Sudeste.

O sistema HVDC é composto pelas pontes retificadoras em Itaipu, que transformam a energia gerada nas turbinas (500 kVac) em 50 Hz para $\pm$ 600 kVDC. Para transmitir a energia, duas linhas de transmissão em corrente-contínua de 800 km com capacidade de 2625 amperes cada uma, são utilizadas [1]. Na estação receptora do HVDC, em Ibiúna, inversores de 12 pulsos são utilizados para transformar a energia em corrente alternada e distribuí-la para o restante do Brasil. Um total de 18432 tiristores compõe todo o sistema HVDC. Como será visto, este sistema introduz harmônicos característicos (11º e 13º) e não-característicos (3º e 5º) na rede de

distribuição, gerando a necessidade de utilizar filtros na subestação. A Fig. 2.2 mostra o diagrama unifilar do sistema de 345 kVac da subestação (SE) de Ibiúna.

Ao longo do tempo, percebeu-se que a poluição harmônica está crescendo, o que provoca problemas de sobrecarga nos filtros causando o seu desligamento. Neste capítulo iremos verificar como sistema HVDC insere harmônicos na rede e propor uma alternativa para monitorar em tempo real sua propagação pela subestação de Ibiúna. A Fig. 2.2 mostra o diagrama unifilar da subestação de Ibiúna.

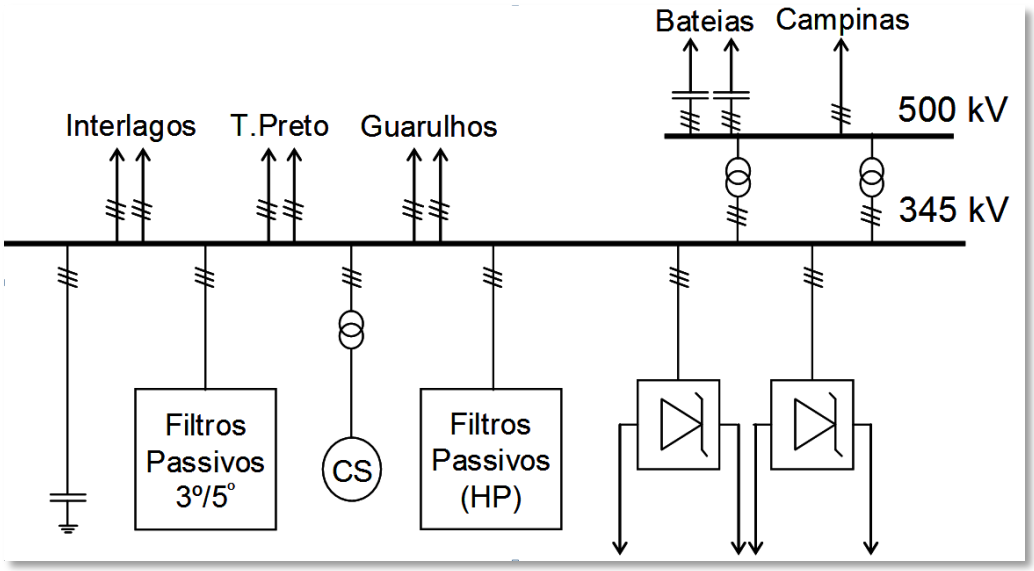


Figura 2.2 - Diagrama unifilar do sistema de 345 kVAC da subestação de Ibiúna [2].

2.1 O Conversor 12 Pulsos

Cada linha de transmissão do sistema HVDC de Itaipu possui um retificador 12 pulsos, que transforma a corrente CA em CC, e um inversor de 12 pulsos que recebe a energia em CC e transforma para CA. Após retificar e transmitir a energia, os conversores de 12 pulsos convertem a energia para uma forma de onda aproximadamente senoidal, através de disparos seqüenciais das suas válvulas a

tiristores. A Fig. 2.3 (b) mostra o esquema completo das duas linhas HVDC e seus filtros.

Um inversor de corrente de 12 pulsos é formado por dois conversores de seis pulsos conectados através de um banco de transformadores como mostra o esquema da Fig. 2.3 (a).

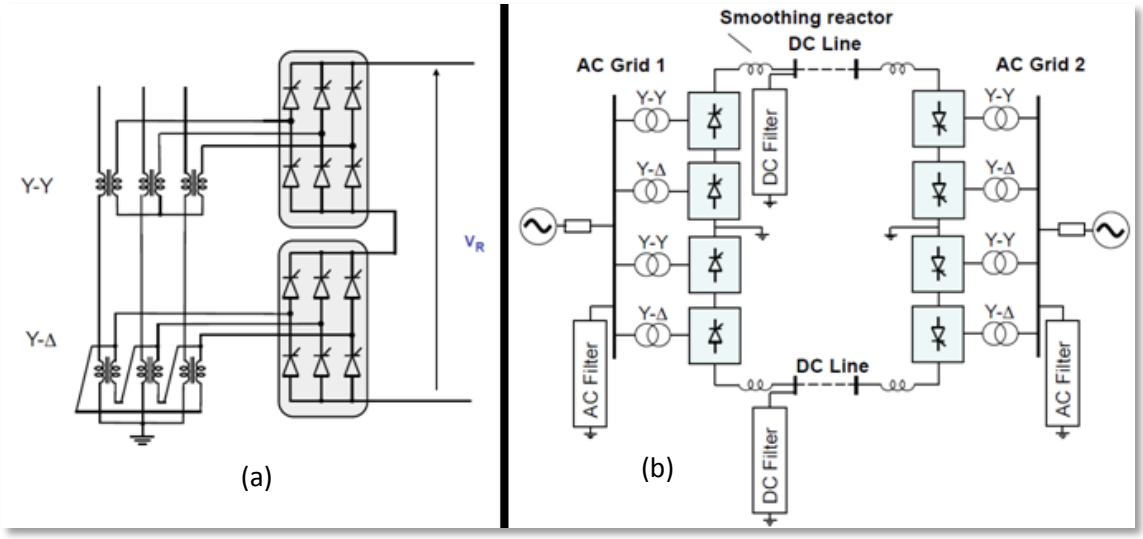


Figura 2.3 – Esquema de um conversor 12 pulsos.

Para sintetizar uma forma de corrente 12 pulsos, a tensão de disparo das seis chaves do conjunto superior da Fig. 2.3(a) deve ser feito por formas de onda quadradas com o período de 60 Hz (mesmo da rede de distribuição) defasadas entre si de π/3 radianos. As tensões de disparo do conjunto de seis chaves inferiores, da mesma figura, devem estar todas defasadas de π/6 em relação ao primeiro conjunto.

Para expressar matematicamente as formas de onda de corrente geradas pelo conversor de 12 pulsos, podemos utilizar as séries de Fourier tal que [3]:

$$I_1 = 2I\left(\cos\Omega t + \frac{1}{11}\cos 11\Omega t + \frac{1}{13}\cos 13\Omega t + \frac{1}{23}\cos 23\Omega t + \ \ldots\right) \qquad (2.1)$$

$$I_2 = 2I\left(\cos\left(\Omega t - \frac{\pi}{3}\right) + \frac{1}{11}\cos 11\left(\Omega t - \frac{\pi}{3}\right) + \frac{1}{13}\cos 13\left(\Omega t - \frac{\pi}{3}\right)\ldots\right) \qquad (2.2)$$

$$I_3 = 2I\left(\cos\left(\Omega t + \frac{\pi}{3}\right) + \frac{1}{11}\cos 11\left(\Omega t + \frac{\pi}{3}\right) + \frac{1}{13}\cos 13\left(\Omega t + \frac{\pi}{3}\right)\ldots\right) \qquad (2.3)$$

em que *I* é uma constante correspondente à amplitude da forma de onda da corrente de 12 pulsos, como mostrada na Fig. 2.4.

Com isso, podemos escrever um *script* em Matlab para verificar visualmente a forma de onda na saída do inversor. O código encontra-se no apêndice A1 deste trabalho e a Fig. 2.4 mostra a forma de onda das correntes de 12 pulsos do sistema trifásico.

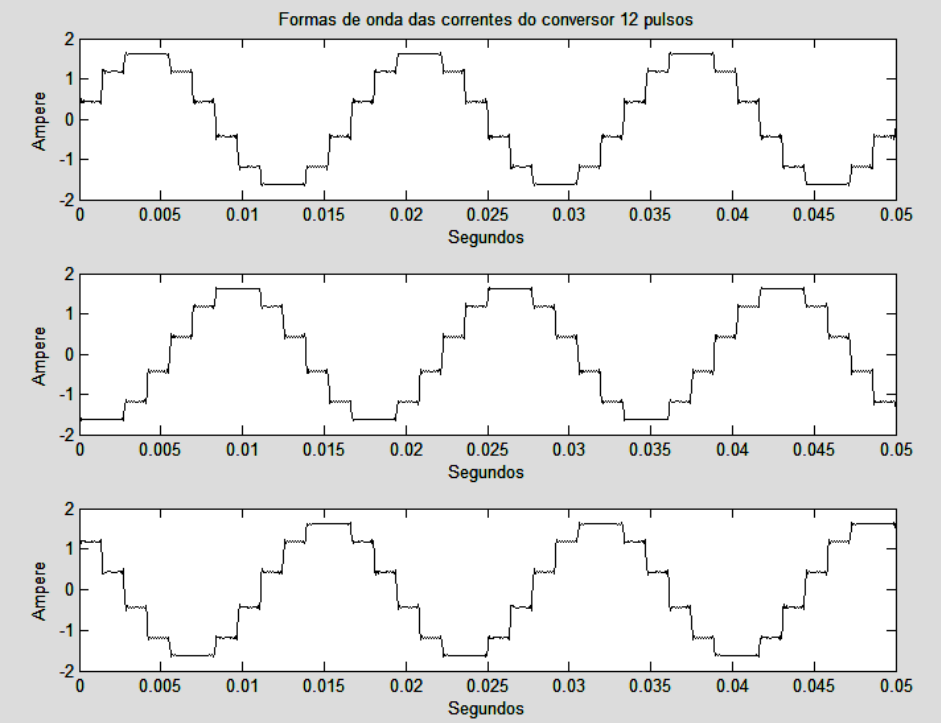


Figura 2.4 – Formas de onda corrente na saída do inversor 12 pulsos.

2.2 Filtros do Conversor

Através da análise das correntes do conversor de 12 pulsos, podemos verificar que estas formas de onda introduzem os harmônicos de ordem $12n \pm 1\ (11º, 13º, 23º, 25º, \ldots)$ na rede de distribuição e por este motivo são chamados de harmônicos característicos. Por outro lado, devido a imperfeições intrínsecas ao funcionamento dos conversores, bem como tolerâncias nas relações de espiras dos trafos conversores do sistema HVDC, uma pequena injeção de correntes harmônicas de 3º e 5º é observada. Estes harmônicos são chamados de não-

característicos, pois não são introduzidos diretamente pelo pela forma de onda ideal de 12 pulsos (Fig. 2.4).

Para resolver os problemas dos harmônicos, foram construídos filtros passivos para estes dois tipos de harmônicos (característicos e não-característicos). Estes filtros são associações de capacitores e indutores que são conectados em paralelo com a rede de 345 kVac e têm suas freqüências de sintonia igual à do harmônico a ser filtrado. Assim, na subestação de Ibiúna, temos filtros rejeita-faixa que se comportam como curto-circuito para as correntes nas freqüências de 180 Hz e 300 Hz (3º e 5º) e para freqüências de 660 Hz e 780 Hz (11º e 13º), além de filtros passa-altas e banco de capacitores.

Os filtros não realizam nenhuma discriminação das origens destas correntes harmônicas e eliminam a poluição harmônica proveniente tanto do sistema HVDC quanto de fontes externas à subestação, via suas interligações por linhas de transmissão em 345 KVac e 500 kVac.

Os harmônicos não-característicos vêm aumentando sua energia ao longo do tempo e provocando problemas de excesso de corrente nos filtros passivos dos harmônicos não-característicos. Este fato já gerou a instalação de dois novos filtros em Ibiúna, alterando o projeto original que possuía dois filtros sendo que apenas um deveria estar em funcionamento. Hoje, apesar dos novos filtros, o sistema funciona no limite da sua operação e problemas de sobrecarga continuam ocorrendo. Vale ressaltar que a indisponibilidade total dos filtros 3º e 5º obriga a redução da potência transmitida pelo sistema HVDC, o que acarreta enorme perda financeira e redução da confiabilidade global do sistema interligado Nacional (SIN) [2].

2.3 Conclusões

Embora não se saiba ao certo a origem dos harmônicos não-característicos, supõe-se que a topologia da rede e seu sistema de carga sofreram alterações ao longo dos anos e que isto pode ter causado o aumento desta energia. Acredita-se que as

distorções podem estar relacionadas ao aumento das cargas não-lineares utilizadas pelos distribuidores e consumidores conectados à rede, mas encontrar os responsáveis é uma tarefa complexa.

De qualquer forma, existe a necessidade de contornar este problema e estudos vêm sendo realizados para descobrir a natureza destes harmônicos e propor soluções para esta questão.

Uma solução seria a introdução de filtros ativos de maior eficiência para aumentar a potência de redução dos harmônicos, ou até bloquear a injeção de correntes harmônicas provenientes de fontes externas a subestação. Entretanto, se os harmônicos continuarem a crescer, o limite da capacidade do filtro será alcançado e outra solução deverá ser buscada.

É de extrema importância a instalação de um sistema de monitoramento em tempo real dos harmônicos no barramento da subestação de Ibiúna, na tentativa de identificar as fontes geradoras desta poluição [2] e assim identificar se estas são externas ou internas, provocadas pelos conversores de potência, à subestação. A base desta solução é a utilização da lei de *Kirchhoff* das correntes [4], que diz que o somatório das correntes injetadas em um nó é igual a igual a soma das correntes que saem do mesmo. Assim, como é possível separar as freqüências, os somatórios do 3º e 5º harmônicos injetados neste nó também seguem a mesma regra.

Para que a lei de *Kirchhoff* seja aplicada de forma correta, é necessário que as medições de correntes nos diferentes pontos do barramento sejam feitas de forma sincronizada, dando origem ao "Sistema SCADA para Monitoramento em Tempo Real da Propagação Harmônica em Estações Conversoras HVDC". Este sistema é um projeto desenvolvido pelo laboratório LEMT, da COPPE/UFRJ, em parceria com Furnas S/A, e será discutido no próximo capítulo.

Capítulo 3 O Sistema SCADA Harmônico

Como mencionado no final do Capítulo 2, o "Sistema SCADA (Supervisory Control and Data Acquisition) para Monitoramento em Tempo Real da Propagação Harmônica em Estações Conversoras HVDC" é um projeto de P&D com Furnas S/A que foi concebido com o objetivo de identificar as fontes de injeção de correntes harmônicas (3º e 5º), no barramento da SE de Ibiúna, que provocam sobrecarga nos filtros desta subestação (SE Ibiúna), local em que estão situados os conversores de 12 pulsos do sistema HVDC. Daqui em diante, o título deste projeto será abreviado por "SCADA Harmônico".

A idéia da identificação da propagação harmônica se baseia na lei de *Kirchhoff* das correntes. Segundo esta lei, a soma das correntes que entram em um nó deve ser igual à soma das que saem do mesmo [2]. Assim, podemos entender que as correntes que trafegam na subestação são vetores cuja magnitude é definida pela amplitude da corrente e a direção é definida pela fase da corrente. Podemos expressar matematicamente o somatório fasorial das correntes que entram tal que:

$$\sum_{n=1}^{N} I_n = I_F \qquad (3.1)$$

em que I_n são as correntes no nó e N é o número de "ramos" que este nó possui, excluindo I_F, que é a corrente que vai para os filtros. A Fig. 3.1 mostra uma aplicação da lei de Kirchoff das correntes para a Equação (3.1).

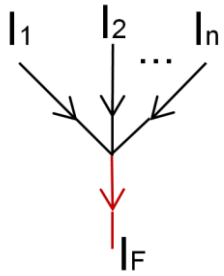


Figura 3.1 – Exemplo de aplicação da lei de Kirchoff para as correntes.

Para ser possível realizar este somatório vetorial devemos conhecer as componentes harmônicas e suas fases para cada corrente medida. Para isso, aplicamos a transformada discreta de Fourier (DFT) e realizamos o somatório das diferentes componentes de correntes nas freqüências de 180 e 300 Hz separadamente, referentes ao 3º e 5º harmônicos, respectivamente.

Para identificar quais correntes estão sobrecarregando o filtro, devemos analisar o ângulo e a magnitude dos fasores das componentes harmônicas destas correntes em relação ao ângulo do fasor de tensão. Assim, quando o ângulo do fasor de uma corrente $I_n\,(\theta_n)$ for tal que $-90° < \theta_n < 90°$, podemos entender que esta corrente está entrando no filtro e, conseqüentemente, contribuindo para sobrecarregar os filtros. Da mesma forma, se este ângulo for tal que $90° < \theta_n < 270°$, esta componente harmônica está saindo do filtro. Isto significa que ao realizarmos o somatório, iremos verificar que as componentes harmônicas que saem do filtro estão contribuindo para aliviar o filtro, pois reduzem o valor resultante da componente harmônica entrando no filtro ao somarem valores negativos ao somatório.

Por outro lado, como os pontos de medição das correntes estão distantes entre si, o ângulo do fasor da tensão de referência poderá sofrer uma variação devido à propagação da mesma até estes pontos. Isto provocaria um erro na medição das fases das componentes harmônicas da corrente. Calculamos, então, o comprimento de onda da freqüência da rede para verificar a influência destas distâncias relativas no erro de fase dos fasores de corrente. Considerando que a corrente trafega aproximadamente na velocidade da luz (c), temos que:

$$\lambda_{60} = \frac{c}{f} \cong \frac{300000\ km/s}{60\ Hz} = 5000\ km \qquad (3.2)$$

Como as distâncias entre os pontos de medição não ultrapassam 1 km, podemos calcular na fundamental (60 Hz), uma defasagem (ϕ) máxima de:

$$\phi_{60} = 360° \frac{1\ km}{5000\ km} = \ 0{,}072° \qquad (3.3)$$

Como este valor é muito pequeno, para efeitos de praticidade, podemos desprezá-lo e assumir que o barramento da subestação é formado por um único nó onde entram e saem correntes.

Outro ponto importante a ser considerado nesta abordagem é o sincronismo das aquisições das correntes. Como várias unidades monitoram diferentes pontos, para que a lei de *Kirchhoff* funcione corretamente devemos amostrar as correntes no barramento de forma sincronizada. Assim, os conjuntos de amostras de corrente de todas as UASs devem estar todos referenciados ao mesmo momento inicial. A Fig. 3.2 mostra o exemplo de um esquema para uma aquisição sincronizada no barramento da subestação.

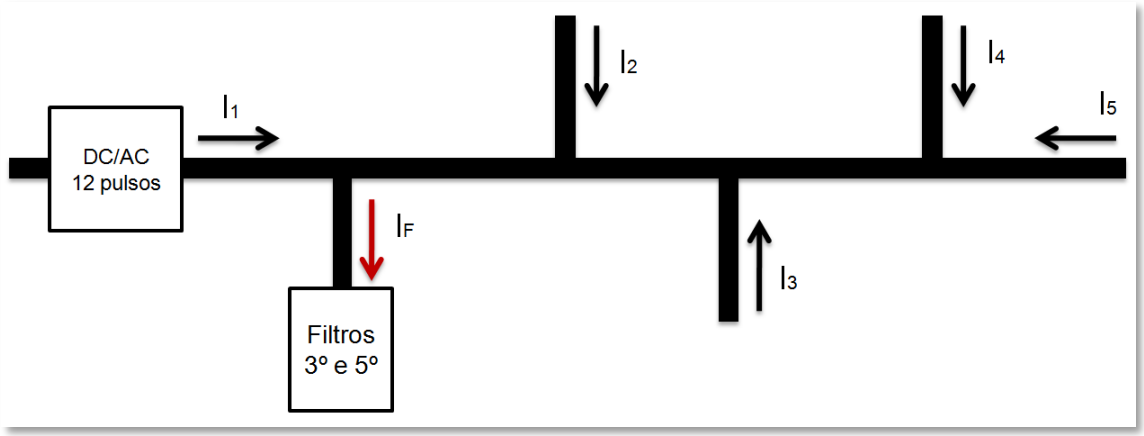


Figura 3.2 – Exemplo de aquisição sincronizada das correntes da subestação de Ibiúna.

3.1 Sincronismo das Aquisições

O sincronismo das aquisições é feito através de um *trigger* único que dispara simultaneamente os conversores A/D (analógico para digital) de todas as unidades de aquisição de sinal (UASs). Este *trigger* é formado por um sinal digital referenciado a uma base de tempo estável.

Um atraso no disparo das unidades digitalizadoras provocaria uma diferença de fase entre o valor correto e o valor medido, alterando o ângulo dos fasores e resultando em um erro no somatório das correntes.

Para resolver este problema, foram desenvolvidos circuitos baseados em GPS. Dispositivos GPS são muito utilizados nos dias de hoje para localização e sincronismo

de data/hora. Quando escolhido corretamente, o GPS pode fornecer uma base de tempo que tem sua estabilidade comparada aos relógios atômicos. Neste projeto utilizamos um GPS que nos fornece uma base tempo com um erro menor que 10 ns [6], quando entra em um modo chamado de "*surveying*", o que corresponde a um atraso de fase menor que $0{,}001°$, para o 5º harmônico.

Entretanto, como as unidades estão distantes umas das outras e são disparadas pelo mesmo circuito GPS, devemos levar em consideração o atraso de propagação deste pulso até as unidades de aquisição.

Considerando que as distâncias entre os circuitos GPS e as unidades de aquisição podem variar de 0 a 300 m, podemos então calcular o erro de fase provocado pelo atraso no disparo das medições. Lembrando que o comprimento de onda para o 5º harmônico é dado por:

$$\lambda_{300} = \frac{300000\ km/s}{300\ Hz} = 1000\ km \qquad (3.4)$$

e considerando que a corrente trafega na velocidade da luz, podemos calcular a variação de fase causada por uma distância de 300m, que é dada por:

$$\phi_{300} = 360° \frac{0{,}3\ km}{1000\ km} = \ 0{,}1° \qquad (3.5)$$

3.2 Arquitetura

Para compensar estes atrasos, foi desenvolvido um circuito baseado em FPGA, denominado Concentrador, que mede o tempo de propagação do sinal de *trigger* do GPS até as unidades de medição. Uma vez conhecido este valor, o circuito também é capaz de compensar o sinal de sincronismo para que seja possível garantir o disparo de todos os conversores A/D no mesmo instante.

Além disso, o Concentrador tem por função receber os dados das unidades via fibra ótica e retransmiti-los através de rede ethernet para o Servidor. Por questões de

topologia da SE e limitações de circuito, cada Concentrador possui seu próprio GPS e pode receber até oito UASs (Unidades de Aquisição de Sinal).

Nas unidades de aquisição, um circuito baseado em FPGA também foi desenvolvido para receber o sinal de *trigger* e disparar os A/Ds, garantindo o sincronismo das aquisições. A comunicação entre um Concentrador e as UASs é feita através de fibra ótica e 34 pontos da SE de Ibiúna serão monitorados. A Fig. 3.2 mostra a arquitetura do Sistema SCADA Harmônico.

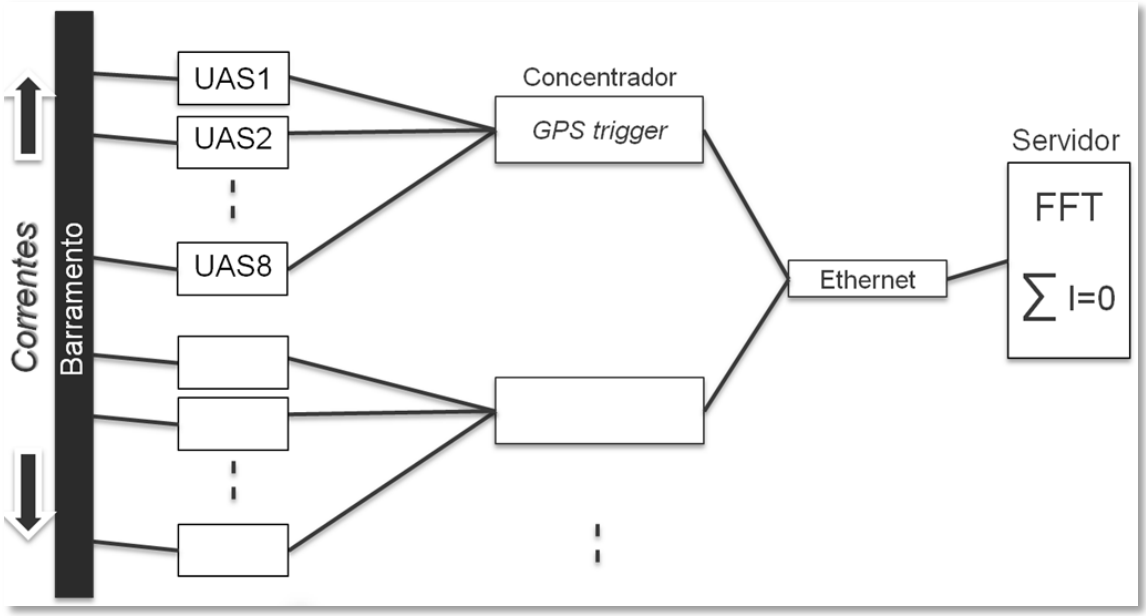


Figura 3.2 – Arquitetura do Sistema SCADA Harmônico.

3.3 Pós-aquisição

O Servidor é um computador que estará executando o *software* também desenvolvido para o Sistema SCADA Harmônico. Este software recebe os dados de todas as unidades de aquisição através da rede *ethernet* e realiza as transformadas de Fourier para cada corrente medida com o intuito de analisar os espectros de freqüência e encontrar as fontes geradoras de correntes harmônicas. Este serviço também realiza o somatório das correntes para validar a Lei de *Kirchoff* e verificar o

funcionamento correto de todo o sistema. Caso seja detectada uma sobrecarga nos filtros, o *software* informa ao operador do sistema quais “ramos” estão contribuindo com ou aliviando o fluxo de 3º e 5º harmônicos para o filtro.

3.4 O Problema

Como foi visto neste capítulo, todas as UASs do Sistema SCADA devem ser disparadas simultaneamente por um *trigger* para realizar as aquisições das correntes do barramento da SE de Ibiúna. O sincronismo é importante, pois nos permite encontrar a direção das correntes e assim identificar quais linhas de transmissão estão contribuindo para sobrecarga dos filtros. Desta forma, manter a integridade da fase no processo de aquisição torna-se fundamental, caso contrário corrompemos a direção dos vetores de corrente e não podemos mais realizar a Lei de Kirchoff das correntes.

Como será visto no próximo capítulo, todo o sistema de digitalização de sinal deve possuir um filtro *anti-aliasing*. Este filtro é responsável por impedir que ocorre o efeito de sobreposição em freqüência e com isso evitar um erro no procedimento de aquisição das correntes e seu projeto requer cuidados específicos para cada caso. Para suprir os requisitos do Sistema SCADA Harmônico este filtro *anti-aliasing* não deve distorcer a fase da corrente amostrada a ponto de corromper os cálculos de pós-aquisição realizados no Servidor. No próximo capítulo apresentaremos o projeto do filtro *anti-aliasing* analógico e no Capítulo 5 estudaremos o projeto do filtro digital.

Capítulo 4 Filtro Anti-aliasing Analógico

Como mencionado no Capítulo 3, o objetivo do Sistema SCADA é monitorar as correntes que passam no barramento da subestação para proteger os filtros de harmônicos não-característicos do sistema HVDC. Através do processamento pós-aquisição das correntes, é possível identificar os "ramos" que carregam componentes com freqüências que sobrecarregam o filtro (180 Hz e 300 Hz).

Nas unidades remotas de medição, que são denominadas por UASs, transdutores de corrente convertem o valor analógico da corrente em um sinal de tensão analógico. Em seguida, conversores A/D (Analógico/Digital) convertem este sinal de tensão para um sinal discreto no tempo e em amplitude (sinal digital), correspondente ao valor medido.

O processo de digitalização é composto por um filtro *anti-aliasing*, um amostrador e um circuito que converte o valor analógico para o seu correspondente numérico (A/D) e envia este valor para o microprocessador [5]. Este sistema é exemplificado no diagrama de blocos da Fig. 4.1.

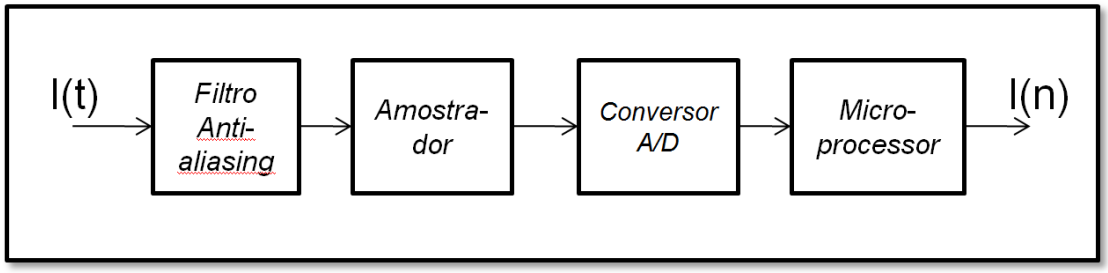


Figura 4.1 - Diagrama de blocos do sistema de digitalização.

O *Amostrador* é o circuito responsável por manter o valor de tensão analógico durante o processo de conversão para digital e normalmente encontra-se dentro do circuito integrado (chip) do conversor A/D [6].

O processo de amostragem de um sinal analógico $x_c(t)$ realizado por um amostrador ideal pode ser definido como [7]:

$$x_c[n] = x_c(nT), -\infty < n < \infty \qquad (4.1)$$

em que *T* é o intervalo de tempo entre as amostras discretizadas a partir do sinal contínuo no tempo, denominado período de amostragem. Para calcular o espectro de freqüências do sinal amostrado $x_c(t)$, aplicamos a transformada de Fourier contínua no tempo, que é dada por [7]:

$$X_c(j\Omega) = \int_{-\infty}^{\infty} x_c(t) e^{-j\Omega t} dt \qquad (4.2)$$

Desenvolvendo este resultado e utilizando o modelo em que o sinal amostrado de $x_c(t)$ é formado pelo sinal $x_S(t)$ tal que:

$$x_S(t) = x_c(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \qquad (4.3)$$

podemos verificar que o sinal $x_S(t)$ é composto pelo sinal original $x_c(t)$ multiplicado por uma seqüência periódica de impulsos de *Dirac* $\left(\delta(t)\right)$. Utilizando a transformada de Fourier contínua, podemos deduzir a expressão do espectro $X_S(j\Omega)$ tal que [7]:

$$X_S(j\Omega) = \frac{1}{T} \sum_{n=-\infty}^{\infty} X_c\left(j(\Omega - n\Omega_T)\right) \qquad (4.4)$$

em que $\Omega_T = 2\pi/T$ é definida como a freqüência de amostragem (*sampling frequency* $\Omega_T = 2\pi F_S$*)*, a qual é proporcional ao inverso do intervalo de tempo entre amostras de $x_c[n]$. Ao analisarmos a função $X_S(j\Omega)$, podemos verificar que o espectro do sinal amostrado é uma função periódica em relação a Ω e com período igual a Ω_T [7]. A Fig. 4.2(a) mostra um exemplo de espectro X_S de 0 até F_S.

Entretanto, para calcular o espectro do sinal digitalmente e obter uma seqüência discreta correspondente ao espectro X_S (contínuo), utilizamos a DTFT (*Discrete-Time Fourier Transform*). Para achar a relação entre a transformada discreta

e contínua de Fourier, utilizamos a relação entre a freqüência analógica (Ω) e digital (ω), $\omega = \Omega\, T$, e obtemos que [7]:

$$X(e^{j\omega}) = \frac{1}{T} \sum_{n=-\infty}^{\infty} X_C(j(\frac{\omega - 2\pi}{T})) \qquad (4.5)$$

Com isso, verificamos que o espectro discreto de X_C é periódico com período igual a 2π, que é o período correspondente a freqüência analógica F_S, porém normalizado tal que $F_S = \frac{\omega}{2\pi T}$, em que ω é a freqüência digital em $rad/amostra$.

Para realizarmos uma aquisição bem sucedida, devemos respeitar o teorema de amostragem de *Nyquist* [8] que determina que um sinal amostrado a uma taxa F_S não deve conter componentes de freqüência maior ou igual a $F_S/2$. Caso contrário, o efeito de *aliasing* pode ocorrer, distorcendo o sinal digitalizado e impedindo sua reconstrução.

O *aliasing* ocorre devido à periodicidade do espectro X_S, conforme descrito anteriormente, as freqüências que estavam acima de $F_S/2$ aparecem como freqüências abaixo de $F_S/2$. A Fig. 4.2(b) exemplifica este efeito mostrando como uma alta freqüência de valor F_2 interfere no sinal digitalizado na freqüênciaF_1.

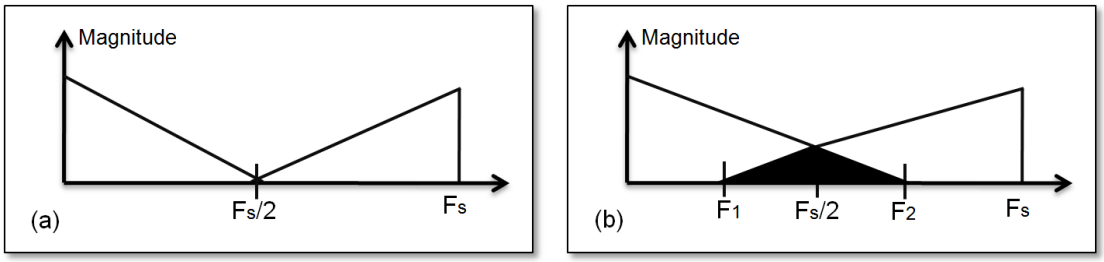


Figura 4.2 – (a) Efeitos da amostragem a F_S, sem *aliasing* e (b) com *aliasing*.

Para garantir que este efeito não irá ocorrer, devemos filtrar o sinal desejado antes do processo de conversão para digital e retirar as altas freqüências indesejadas. Assim, um filtro analógico, filtro *anti-aliasing*, deve ser utilizado e projetado para que

as componentes acima de $F_S/2$ sejam atenuadas o suficiente para não interferir no sinal digitalizado.

4.1 Características da Aquisição de Corrente

A corrente que se deseja medir é composta por componentes de harmônicas de 60 Hz, como visto no Capítulo 2 e Capítulo 3. No Sistema SCADA, o objetivo é observar as freqüências do 3º e 5º harmônicos, ou seja, 180 Hz e 300 Hz. Por especificação, foi definido que a máxima freqüência que pode existir neste sinal é de 1800 Hz. Outra especificação do projeto requer que a amostragem do sinal seja feita de modo que cada ciclo da componente fundamental do sinal (60 Hz) possua 128 pontos. Podemos assim calcular a taxa de amostragem que é dada por:

$$F_S = 128 * 60 = 7680\ Hz \qquad (4.6)$$

A última especificação do processo de aquisição do sinal de corrente exige que o sinal digitalizado possua uma resolução de 16 bits. Como isso, podemos calcular a razão sinal ruído (SNR), em decibéis, que é dada por:

$$SNR_{dB} = 20 * log_{10}\left(2^{16\ bits}\right) = 96{,}3\ dB \qquad (4.7)$$

O circuito A/D utilizado no projeto SCADA Harmônico é o AD7687 [6]. Ao consultarmos o seu manual, verificamos que o valor referente à sua SNR é de $95\ dB$. É natural que este valor seja menor que o calculado, pois no funcionamento real dos dispositivos temos a presença de ruído que nos impede de alcançar o valor teórico ideal.

4.2 Aproximações do Filtro Analógico

Como visto no Capítulo 3, o sistema de aquisição é composto por várias unidades que digitalizam correntes em diferentes pontos do barramento da subestação de forma simultânea e sincronizada. Para ser possível utilizar a lei de *Kirchoff* e verificar que a soma das correntes que entram no nó, representado pela subestação, é igual à soma das que saem do mesmo, devemos garantir que o erro de fase provocado pelo filtro a*nti-aliasing* não distorça o resultado. Com isso, o ponto crítico do sistema de aquisição sincronizada é garantir um erro de fase conhecido e controlado para não corromper os resultados pós-aquisição.

Existem algumas aproximações de filtros analógicos e cada uma possui uma característica que o projetista deve levar em consideração para escolher o filtro mais apropriado à sua aplicação.

Os filtros *Butterworth* [9] possuem uma resposta de magnitude maximamente plana em sua banda passante. Isto significa que o nível de ondulações (*ripple*) é muito baixo e aproxima-se de zero tanto na faixa passante quanto na de rejeição. Este filtro é muito utilizado para aplicações em que a distorção de magnitude na banda de passagem deve ser mínima.

Os filtros do tipo *Chebyshev* [10] são característicos por possuírem uma atenuação na banda de transição bastante íngreme, em detrimento de um maior *ripple* na banda de passagem. Estes filtros são muito utilizados quando o ponto chave do projeto é expulsar as freqüências não desejadas do sinal, ao mesmo tempo em que a distorção da magnitude na banda passante não é tão relevante. Quando comparado aos filtros do tipo *Butterworth*, possuem uma ordem menor para uma mesma atenuação na faixa de rejeição. Na aproximação *Chebyshev* inversa, ondulações aparecem na banda de rejeição e a banda de passagem torna-se mais plana.

Outra aproximação bastante utilizada são os filtros do tipo elíptico [11]. Estes filtros possuem ondulações ajustáveis de forma independente, tanto na banda de passagem, quanto na banda de rejeição. São muito utilizados por possuírem uma rápida transição de ganhos para um menor *ripple*, quando comparado às outras topologias.

Por último, os filtros do tipo Bessel [12] são característicos por serem maximamente planos na resposta de fase, mas a resposta de magnitude possui um declive suave e gradual na banda de passagem. Este filtro foi escolhido para servir como filtro *anti-aliasing* analógico deste projeto porque é o que menos distorce a resposta de fase.

4.3 Projeto do Filtro Analógico

Considerando as características de corrente mencionadas na Seção 4.1 e a freqüência de amostragem calculada na Equação (4.1), o filtro *anti-aliasing* analógico deve atenuar em pelo menos 95 dB as freqüências acima de 3840 Hz ($F_S/2$). Como as freqüências que se deseja observar são as componentes de 3º e 5º harmônicos definimos que a máxima freqüência observável do sinal correspondente à corrente é de 1800 Hz. Com isso, freqüências abaixo deste valor (F_c, *freqüência de corte*) devem possuir ganho de aproximadamente 0 dB. Na faixa de 1800 a 3840 Hz, chamada faixa de transição, deve ocorrer um ganho que varia gradualmente de 0 dB a -95 dB e a partir de 3840 Hz todas as freqüências devem ser atenuadas de no mínimo 95 dB. A largura da faixa de transição é o ponto crítico do projeto do filtro analógico, como veremos mais adiante, uma vez que sua inclinação define a ordem do filtro (tamanho do filtro) e filtros analógicos de ordem muito elevadas tornam-se inviáveis de serem implementados na prática. A Fig. 4.3 mostra o gráfico da resposta em freqüência desejada para o filtro *anti-aliasing* analógico.

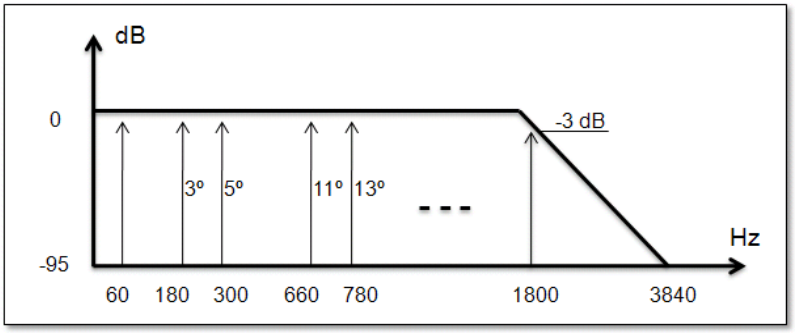


Figura 4.3 – Resposta em freqüência do circuito analógico *anti-aliasing*.

Para avaliar a ordem resultante da especificação do filtro em questão, a função do *Matlab Besself* foi utilizada [13]. Detalhes do *script* que permite gerar o gráfico de resposta em freqüência deste filtro analógico podem ser verificados no Apêndice A2. A partir das simulações de diferentes ordens, foi possível verificar que uma ordem maior que 25 é necessária para cumprir os requisitos da Fig. 4.3. Pelo fato desta ordem ser muito elevada, o filtro torna-se muito grande (com muitos estágios de amplificação) e assim muito sensível à variação dos componentes eletrônicos que formam o filtro. Portanto este filtro é inviável de ser realizado na prática.

4.4 Alternativa para o Filtro Analógico

Como alternativa para este problema sugerimos aumentar a taxa de amostragem através de um *upsampler* (super amostragem). Fazemos então uma amostragem a uma taxa 16 vezes maior do que a taxa de amostragem original, isto é a 122880 Hz, o que reduz a ordem do filtro analógico a um valor viável de se realizar na prática. Com isso as freqüências que provocam o efeito de *aliasing* passam a ficar acima do valor de 61440 Hz. Dessa forma, podemos projetar um filtro *anti-aliasing* em que a freqüência de corte tem o mesmo valor de F_c do filtro anterior (1800 Hz), porém as freqüências de atenuação de 95 dB passam a estar acima de 61440 Hz. Assim, a inclinação da atenuação na faixa de transição torna-se muito mais suave e a complexidade do filtro analógico se reduz a quarta ordem (com dois amplificadores). O primeiro bloco da Fig. 4.4 mostra a especificação da magnitude da resposta em freqüência para o filtro *anti-aliasing* analógico projetado.

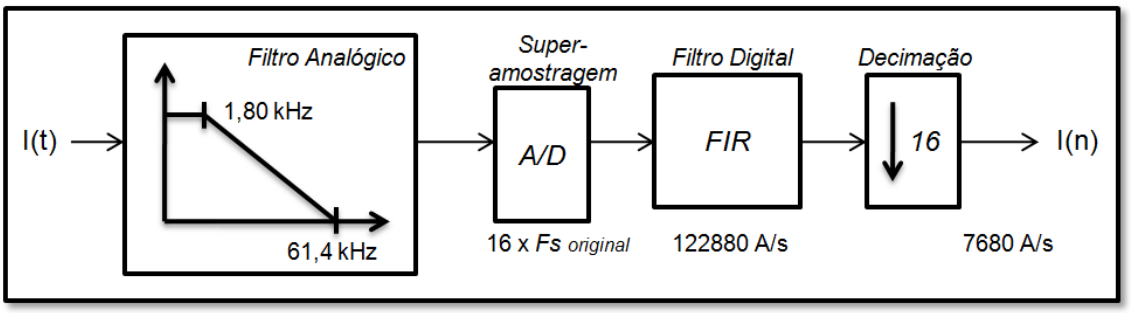


Figura 4.4 – Diagrama de blocos da filtragem *anti-aliasing* com filtro analógico, filtro digital e *super-amostragem*.

Em contrapartida, no mundo digital ocorre um aumento da complexidade dos circuitos. Uma vez que o período de amostragem diminui, mais dados são produzidos para os microprocessadores, exigindo que mais contas sejam realizadas em um menor espaço de tempo. Isto sobrecarregaria o sistema SCADA Harmônico e aumentaria o seu custo de cálculos pós-aquisição. Para resolver este problema, uma combinação de filtro digital e decimação *(downsamplig)* é realizada para converter a taxa de amostragem para a freqüência inicialmente desejada (7680 Hz) e permitir uma complexidade menor nos cálculos pós-aquisição. A Fig. 4.4 mostra o diagrama de blocos completo do processo de amostragem.

No Capítulo 5 sugerimos topologias com filtros FIR que aproveitam o efeito do *down-sampler* para reduzir a ordem do filtro *anti-aliasing* digital. No final do Capítulo 7 comprovamos empiricamente que o filtro otimizado é mais eficiente, mostrando-se nove vezes mais rápido do que o filtro FIR com o projeto convencional.

4.5 Implementação do Filtro Analógico

Para implementar o filtro analógico do primeiro bloco da Fig. 4.4, a ferramenta para projeto de filtros analógicos da *Texas Instruments* [14], chamada de *Filter-Pro*, foi utilizada. Com esta ferramenta é possível configurar as bandas de freqüência para as diferentes topologias de filtros e gerar gráficos de resposta em freqüência com seus esquemas eletrônicos correspondentes. A Fig. 4.5 mostra o gráfico da resposta em

freqüência quando se deseja implementar uma topologia Bessel para as freqüências em questão e a sugestão do circuito eletrônico que realiza o filtro desejado.

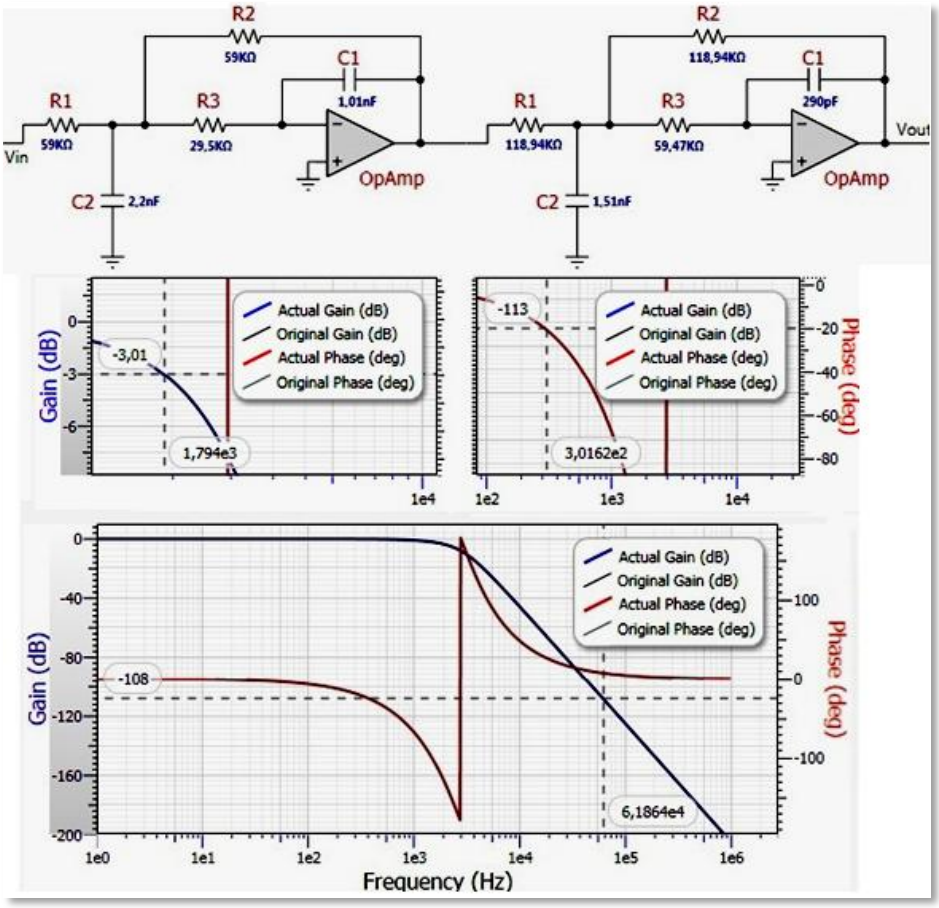


Figura 4.5 – Ferramenta *Filter-Pro* para projeto de filtros analógicos (*Texas Instruments*).

4.6 Simulação no SPICE

Para verificar o funcionamento do filtro projetado, foi utilizada uma ferramenta de simulação de circuitos eletrônicos do tipo *SPICE* [2]. A simulação *SPICE* é o padrão utilizado pelas indústrias de circuitos integrados para verificar o correto funcionamento dos sistemas desenvolvidos [15]. A Fig. 4.6(a) e 4.6(b) apresentam os resultados obtidos durante a simulação do circuito do filtro analógico sugerido na Seção 4.4. O esquemático completo do mesmo circuito pode ser consultado no apêndice A3.

Ao analisarmos o gráfico da magnitude da resposta em freqüência do filtro simulado (Fig. 4.6(a)), notamos que o ponto atenuação de 3dB está entre 1.0 kHz e 3.0 kHz, se aproximando do valor de 1800 Hz que é a freqüência de corte. Ainda nesta figura, podemos observar uma atenuação de 100 dB próxima aos 60 kHz. A Fig. 4.6(b) mostra a fase da resposta em freqüência e podemos verificar uma diferença de fase em torno de 20 graus para as freqüências próximas ao 5º harmônico e uma curva próxima da linearidade, como era esperado para a fase. Assim, concluímos que o filtro encontra-se dentro das especificações.

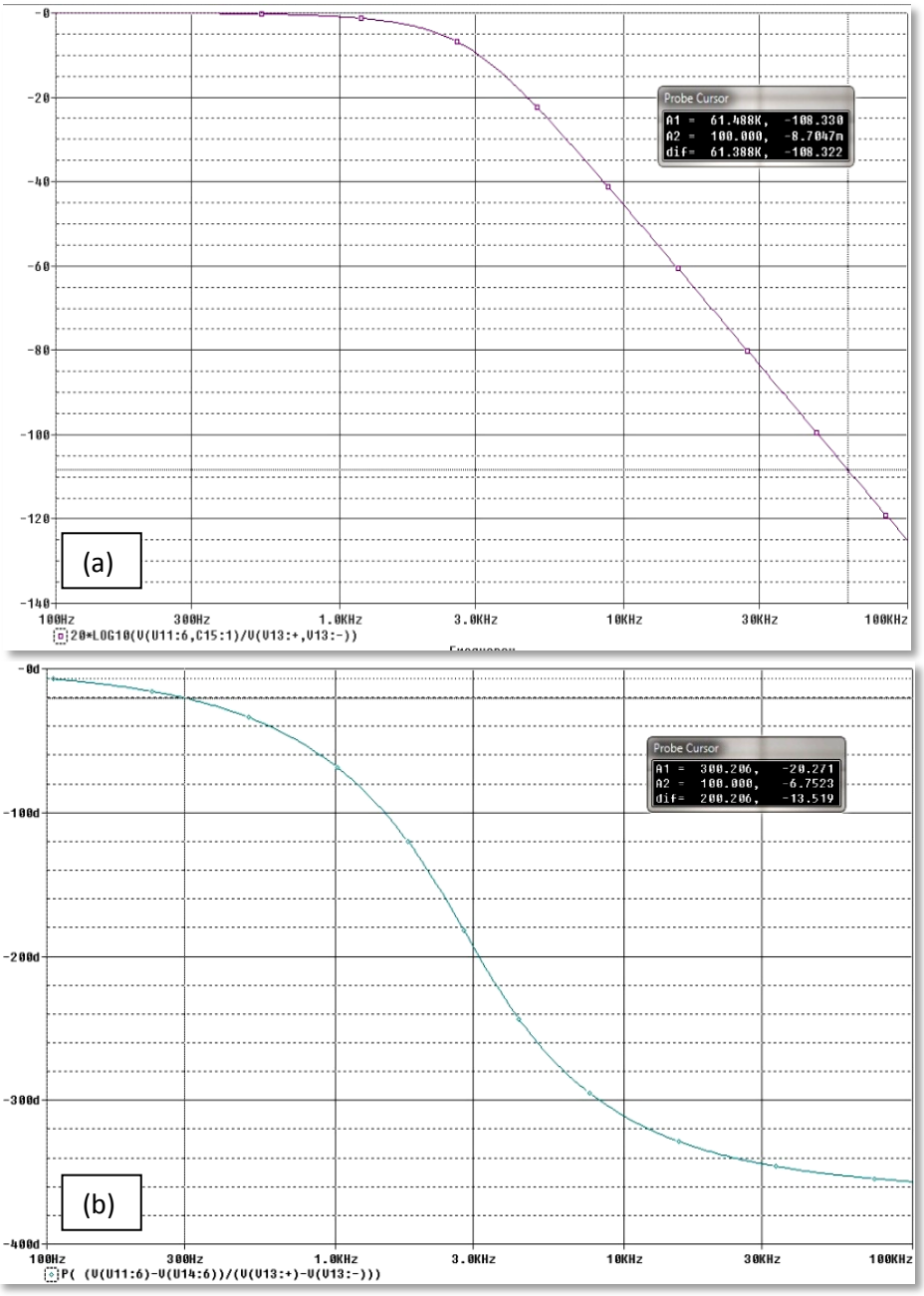


Figura 4.6 – Simulação da resposta em freqüência do circuito do filtro analógico: (a) magnitude (b) fase.

4.7 Função de transferência do filtro analógico

Na seção anterior o filtro analógico foi projetado e verificamos que ele encontra-se dentro das especificações. Entretanto, para que o sistema de aquisição sincronizada chegue a resultados satisfatórios, é necessário que o erro de fase devido à filtragem *anti-aliasing* nos pontos de medição sejam todas, nas freqüências de interesse (180 Hz e 300 Hz), muito próximas entre si. Isto é relevante porque, quando ocorre uma distorção de fase, os fasores de corrente modificam suas direções alterando o valor do somatório. Entretanto, no caso de todas as unidades de aquisição distorcerem a fase da corrente do mesmo valor, o resultado final do somatório das correntes não será modificado.

Para isto, devemos garantir que o circuito projetado possa ser repetido, quando utilizamos valores de componentes comerciais, evitando uma grande distorção de fase relativa entre os circuitos. Como visto no Capítulo 3, a variação das fases devido à distância entre os pontos de medição é de no máximo $0{,}1°$. Assim, especificamos que a maior distorção de fase relativa permitida entre as UASs é de $0{,}1°$.

Para realizar uma análise de sensibilidade da fase em relação aos resistores e capacitores, devemos encontrar a função de transferência $H(s)$ do filtro em relação a estes componentes. De posse desta equação, poderemos utilizar um método estatístico e traçar um histograma da variação da fase.

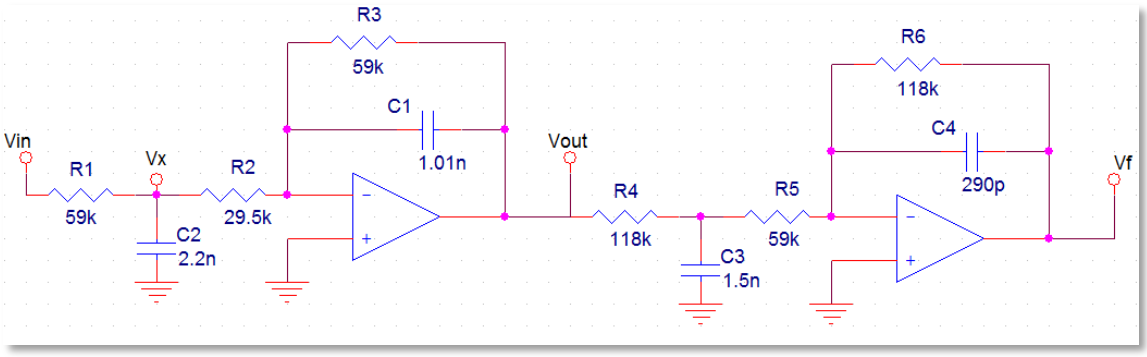


Figura 4.7 – Esquemático do filtro analógico de ordem 4.

Assim, analisando o primeiro estágio do filtro mostrado na Fig. 4.7, podemos aplicar o teorema de Norton [16] e entender que o sinal de entrada e saída são fontes de tensão V_{in} e V_{out}, respectivamente. Com isso, para analisar o valor da tensão no ponto V_x, curtamos este ponto para o terra, verificamos as correntes entrando e as condutâncias no nó, tal que:

$$V_x = \frac{\sum correntes\ entrando}{\sum condutâncias} = \frac{\frac{V_{in}}{R1} + \frac{V_{out}}{R3}}{\frac{1}{R1} + sC2 + \frac{1}{R2} + \frac{1}{R3}} \tag{4.8}$$

Além disso, lembramos que o ganho do circuito, $\frac{V_{out}}{V_x}$, é definido por:

$$\frac{V_{out}}{V_x} = -\frac{\frac{1}{sC1}}{R2} \tag{4.9}$$

Resolvendo este sistema de equações podemos encontrar $H_I(s) = \frac{Vout(s)}{Vin(s)}$, tal que:

$$H_I(s) = \frac{-R3}{R1 + sC1(R2R3 + R1R3 + R2R1) + s^2C1C2R1R2R3} \tag{4.10}$$

Podemos utilizar este resultado e calcular $H_{II}(s) = \frac{V_f(s)}{V_{out}(s)}$, tal que:

$$H_{II}(s) = \frac{-R6}{R4 + sC3(R5R6 + R4R6 + R5R4) + s^2C3C4R4R5R6} \tag{4.11}$$

Assim calculamos a função de transferência final do filtro dada por:

$$H_f(s) = H_I(s)\, H_{II}(s) \tag{4.12}$$

Entretanto, este modelo matemático considera que todos os componentes são ideais e com isso haverá uma diferença entre a resposta de $H_f(s)$ e a obtida com o circuito simulado no *SPICE*. Para avaliar esta diferença, um *script* no Matlab foi realizado para calcular a fase do modelo matemático nas freqüências utilizadas pelo *SPICE* e traçar um gráfico referente ao erro relativo entre os dois sistemas. Este gráfico é mostrado na Fig.4.8, e o *script* completo pode ser consultado no Apêndice A4.

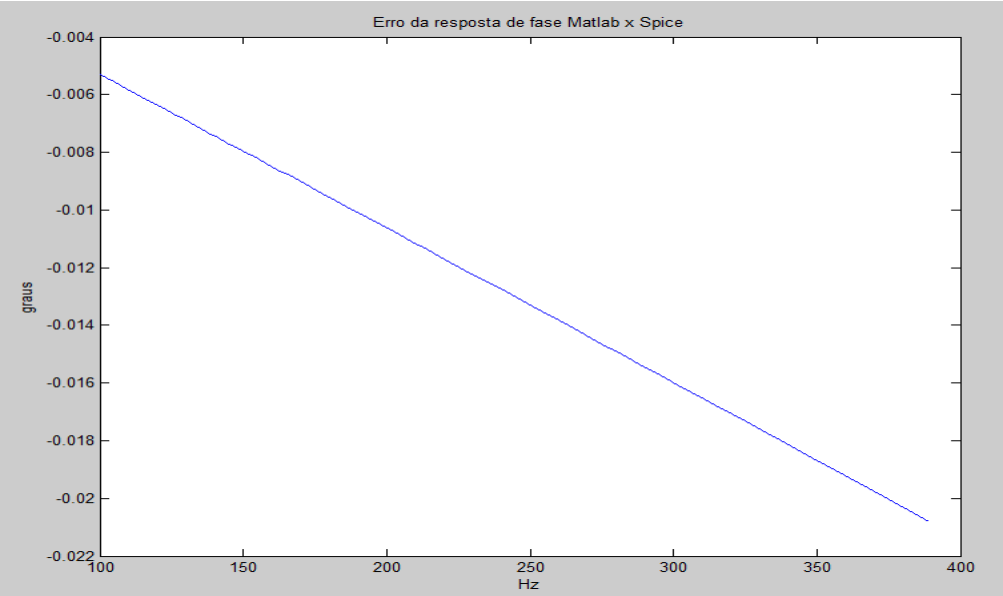


Figura 4.8 – Erro relativo das respostas de fase entre o modelo matemático e a simulação no *SPICE*.

Observando esta figura, podemos verificar que o erro entre os dois modelos para as freqüências abaixo de 300 Hz é menor do que $0{,}017°$. Como este valor é muito pequeno, para efeito de simplicidade, iremos considerar que os dois modelos são idênticos nestas freqüências. Assim, a análise estatística da resposta de fase em relação à variação dos componentes para o modelo matemático, também será idêntica ao modelo da simulação.

4.8 Estatística da variação da fase

Para avaliar a variação da fase em função da variação dos componentes, vamos utilizar o método de Monte Carlo [17] para o modelo matemático ($H_f(s)$) e considerar que os resistores têm precisão de 0,1% e os capacitores tem precisão de 0,5%, pois estes valores são valores típicos mais precisos encontrados no mercado ainda com uma boa relação custo-benefício. Com isso, realizamos um algoritmo para variar aleatoriamente a incerteza dos componentes segundo uma distribuição uniforme. Em seguida, calculamos a fase de $H_f(s)$ para as freqüências do 3º e 5º harmônicos para cada conjunto de componentes, com um total de 10.000 grupos diferentes de

componentes. Ao final, separamos as amostras de saída em grupos de resposta de fase e traçamos um histograma do número de amostras encontrado para cada tipo de resposta de fase. A Fig. 4.9(a) e Fig. 4.9(b) mostram os histogramas referentes à variação da fase do 3º e 5º harmônicos, respectivamente, e o *script* deste algoritmo pode ser consultado no Apêndice A5.

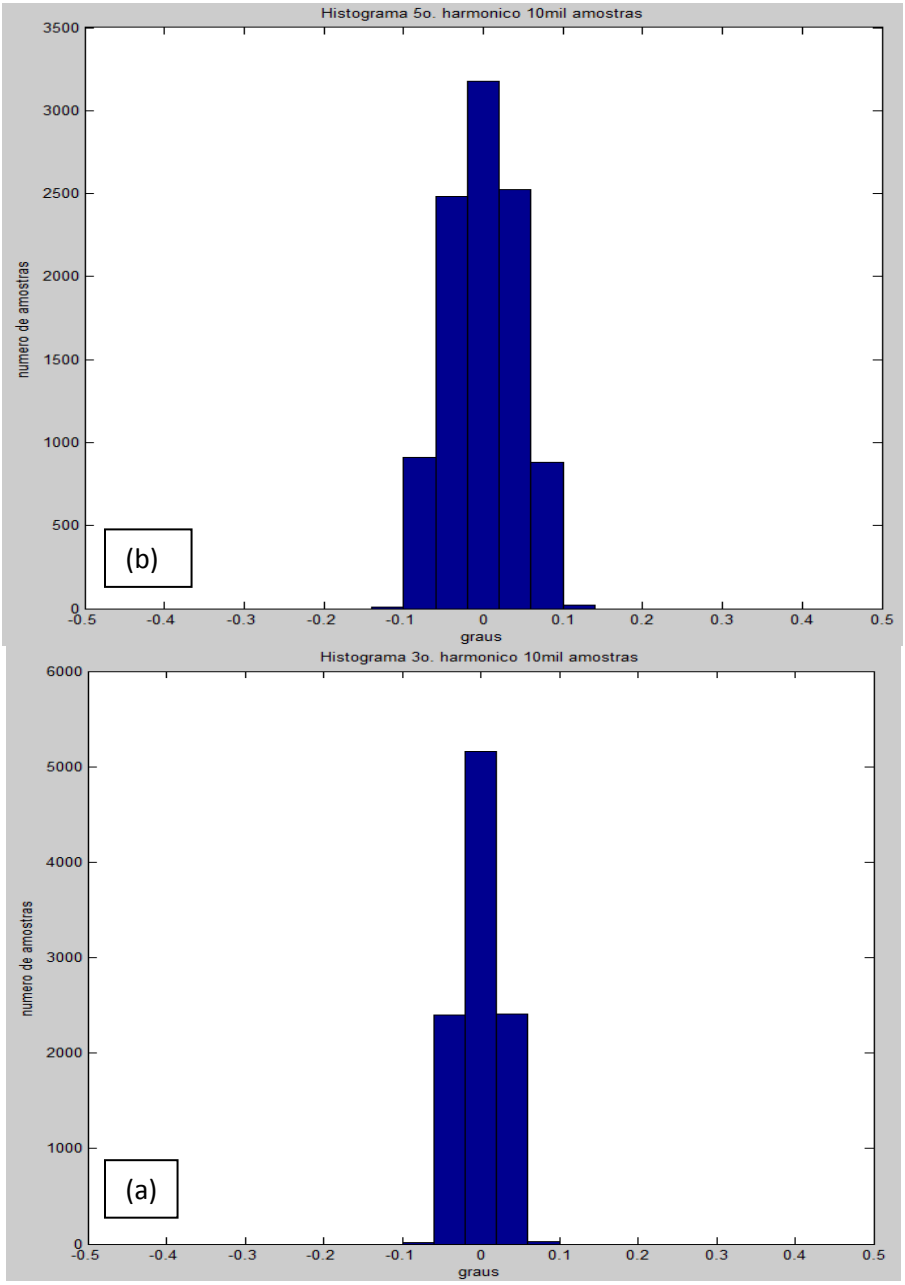


Figura 4.8 – Histogramas de variação da fase em função dos componentes para o 3º harmônico (a) e 5º harmônico (b).

Ao observamos os histogramas, verificamos que a fase do 5º harmônico é mais sensível à variação dos componentes do que a fase do 3º harmônico (180 Hz). Isto ocorre porque, como o 5º harmônico corresponde a uma freqüência maior (300 Hz), sua fase varia mais para um mesmo período de tempo do que o 3º harmônico. Assim, como uma variação nos valores dos componentes provoca uma alteração nas constantes de tempo RC do circuito do filtro, a fase do 5º harmônico irá sofrer uma variação maior.

Além disso, podemos observar que para o caso mais crítico, o 5º harmônico, em aproximadamente 99,7% das respostas de fase não sofreu uma variação maior do que $\pm 0{,}1°$. Como a variação está dentro do especificado no inicio da Seção 4.7, podemos concluir que este filtro é suficiente para o projeto.

No próximo capítulo iremos estudar os detalhes relativos ao projeto do filtro digital e veremos que no mundo digital o problema de variação da resposta de fase está relacionado à resolução em bits dos coeficientes do filtro e dos sinais de entrada e saída, o que torna os sistemas digitais mais fáceis de serem repetidos. Entretanto, devido ao tempo de processamento dos sistemas digitais, um atraso maior será verificado na resposta dos mesmos a um estímulo de entrada.

Capítulo 5 Filtro *Anti-aliasing* Digital

Diferente dos filtros analógicos que processam sinais contínuos no tempo, os filtros digitais são sistemas que realizam operações matemáticas em sinais discretos no tempo [18], isto é, em sinais que foram amostrados como visto no início do Capitulo 4. Dessa forma, para todo o filtro digital existe uma taxa de amostragem associada aos sinais e com isso uma freqüência máxima de resposta. Como os filtros digitais computam os sinais no mundo discreto, uma latência é observada entre a entrada e a resposta do sistema, o que não ocorre com os filtros analógicos que apresentam uma resposta praticamente instantânea.

Ao observamos os diferentes *softwares* que utilizamos no nosso cotidiano, percebemos que o avanço da eletrônica permitiu a existência de circuitos digitais programáveis. Estes programas foram desenvolvidos para serem processados em um computador pessoal, que possui sempre a mesma arquitetura de dispositivo eletrônico.

Por outro lado, esta evolução também deu origem aos circuitos digitais programáveis que provocam modificações físicas nos dispositivos eletrônicos (*hardware*). Isto significa que é possível sintetizar componentes digitais e realizar conexões entre eles, baseado em um código de descrição de *hardware*. Para descrever o funcionamento da eletrônica digital, os projetistas normalmente utilizam códigos escritos em linguagem *VHDL*, *Verilog* ou *System Verilog*. Todos eles são descrições de como uma matriz virgem de portas lógicas, como os FPGAs, CPLDs ou ASICs, se interligam para realizar um determinado circuito eletrônico.

Com isso, a vantagem dos filtros digitais em relação aos analógicos é a capacidade de serem programado, facilitando as atividades relacionadas com o seu desenvolvimento, produção e repetição. Como os filtros digitais são definidos por um conjunto de coeficientes numéricos, não dependem diretamente de um circuito específico, podendo ser repetidos por um mesmo *hardware* sem que haja variações. Assim, sistemas digitais são circuitos mais precisos, com precisão finita, imunes a ruídos e permitem que estruturas mais complexas sejam realizadas. Além disso, os filtros digitais podem ser projetados para que apresentem uma resposta de fase

exatamente linear durante toda a banda de interesse, o que não ocorre com os analógicos.

5.1 Filtros Digitais

Existem dois tipos de filtros digitais, os do tipo FIR e os IIR. Os Filtros do tipo IIR são estruturas recursivas e possuem uma resposta ao impulso de duração infinita, enquanto os filtros FIR são, em geral, implementados por estruturas não-recursivas e possuem uma resposta ao impulso de duração finita.

O projeto de filtros do tipo IIR é normalmente realizado a partir de um filtro passa-baixas normalizado, sendo a sua resposta em freqüência dada por aproximações como as de *Butterworth*, *Chebyshev* e elíptico, como nos projetos de filtros analógicos. Estes filtros não apresentam uma resposta de fase linear.

A resposta de fase linear ocorre quando o filtro possui um atraso constante da saída em relação à entrada. Isto significa que, embora a resposta do filtro não seja instantânea, o que é natural nos sistemas digitais, seu atraso permanece o mesmo para todas as freqüências de entrada. Assim, um atraso constante permite simplificar o projeto do filtro especificado na Seção 4.6, em que se exige um atraso de fase relativo entre as UASs menor que $0{,}1°$. Logo os filtros FIR com fase linear foram escolhidos para este projeto. Um filtro IIR com aproximação *Bessel* poderia ter sido escolhido, mas este não possui uma resposta de fase exatamente linear.

O projeto dos filtros FIR não possui nenhuma conexão com o projeto de filtros analógicos, como no caso do IIR, e é baseado em uma aproximação direta da magnitude da resposta em freqüência desejada, na maioria das vezes com o requisito de ter uma resposta de fase linear. Os filtros FIR de ordem *N* têm uma função de transferência $H(z)$ expressa por [19]:

$$H(z) = \sum_{n=0}^{N} h[n] z^{-n} \qquad (5.1)$$

em que $h[n]$ é o sinal discreto que representa a resposta ao impulso do filtro FIR e a resposta em freqüência correspondente é dada quando $z = e^{j\omega}$ e assim temos que [19]:

$$H(e^{j\omega}) = \sum_{n=0}^{N} h[n]\, e^{-j\omega n} \qquad (5.2)$$

A resposta $y[n]$ do filtro FIR a um sinal de entrada $x[n]$ no domínio do tempo é expressa pela convolução deste sinal com os coeficientes do filtro formado pela resposta ao impulso $h[k]$, tal que [19]:

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k] \qquad (5.3)$$

Esta equação pode ser representada por um diagrama de blocos facilitando o entendimento do sistema discreto, como mostra a Fig 5.1. Nesta figura, os blocos que contêm z^{-1} são componentes que realizam o atraso de uma unidade da amostra e os coeficientes são representados pelos blocos de ganhos $b_0, b_1, b_2, \ldots, b_M$.

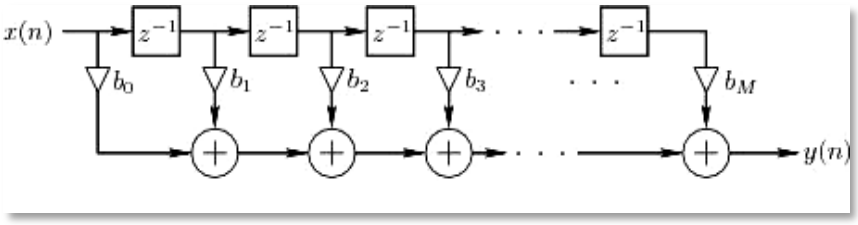


Figura 5.1 – Filtro digital FIR em sua forma direta.

5.1.1 Projeto convencional do FIR com MATLAB

O projeto do filtro FIR pode ser baseado no janelamento da resposta ao impulso ideal, na amostragem da resposta em freqüência ou na otimização de uma função custo.

Para o projeto de filtros FIR existem várias fórmulas como Kaiser, Bellanger e Hermann [20], que ajudam o projetista a estimar a ordem mínima do filtro que atenda as especificações de freqüência desejada. Uma importante propriedade dessas

fórmulas é que a complexidade do filtro é sempre inversamente proporcional a faixa de transição. Assim, quanto mais estreita for a transição da banda de passagem para a banda de atenuação, maior será a ordem do filtro. Se o projeto não alcançar os requisitos de freqüência impostos pela especificação, o projetista deve incrementar a ordem do filtro até que as condições sejam satisfeitas.

Uma abordagem mais exata para o projeto do filtro FIR é baseada em algoritmos iterativos de otimização, que minimizam os erros entre a reposta em freqüência desejada e a alcançada pela manipulação dos parâmetros do filtro. Este tipo de projeto requer que um computador seja utilizado.

Para o projeto do filtro *anti-aliasing* digital, o algoritmo de *Parks-McClellan* foi utilizado para obter um filtro *equirriple* com o Matlab [21]. Este método é o mais utilizado em projetos de filtros FIR. Como mencionado na Seção 4.4, o objetivo do filtro *anti-aliasing* digital é atenuar em no mínimo 95 dB as freqüências acima de 3840 Hz $(F_a = F_S/32)$. Assim, torna-se possível descartar 15 amostras a cada 16 (procedimento de decimação de fator 16) evitando o efeito de sobre-posição (*aliasing*) em freqüência. A freqüência de corte F_c se mantém a mesma em 1800 Hz. Estas especificações e sua normalização são apresentadas na Tabela 5.1.

	F_c	F_a	$F_S/2$	F_S
Freq. (Hz)	1800 Hz	3840 Hz	61440 Hz	122880 Hz
Freq. (Norm.)	0,0146	0,0312	0,5	1

Tabela 5.1 – Projeto do filtro digital simples, freqüências normalizadas e em Hz.

5.1.2 Projeto convencional do Filtro FIR

A Fig. 5.2 mostra a interface para projeto do filtro FIR *equirriple* com o *fdatool* do Matlab. Este ambiente permite definir as especificações de freqüências do filtro FIR, verificar graficamente a resposta de magnitude e de fase, definir a precisão em bits para os coeficientes e para o sinal de entrada. Obtemos, então, um arquivo de saída

com a configuração do filtro. Estes arquivos serão utilizados para realizar a configuração dos filtros no hardware do FPGA (Capítulo 6) e para validação das respostas em freqüência dos filtros implementados no FPGA e pós-processadas no Matlab (Capítulo 7).

Como será visto no Capítulo 6, este filtro será implementado em uma máquina (*hardware*) de ponto fixo com 18 bits de resolução para os coeficientes. A conversão dos coeficientes de ponto flutuante (teórico) para ponto fixo (implementado) causou um aumento do *ripple* na banda de rejeição. Para resolver este problema, o filtro foi re-projetado com uma especificação de atenuação maior: 120 dB na banda de rejeição. A resposta em freqüência deste filtro é mostrada novamente na Fig. 5.6(a) da Seção 5.2.3, na qual uma comparação com a resposta em freqüência deste filtro com a do filtro otimizado é realizada.

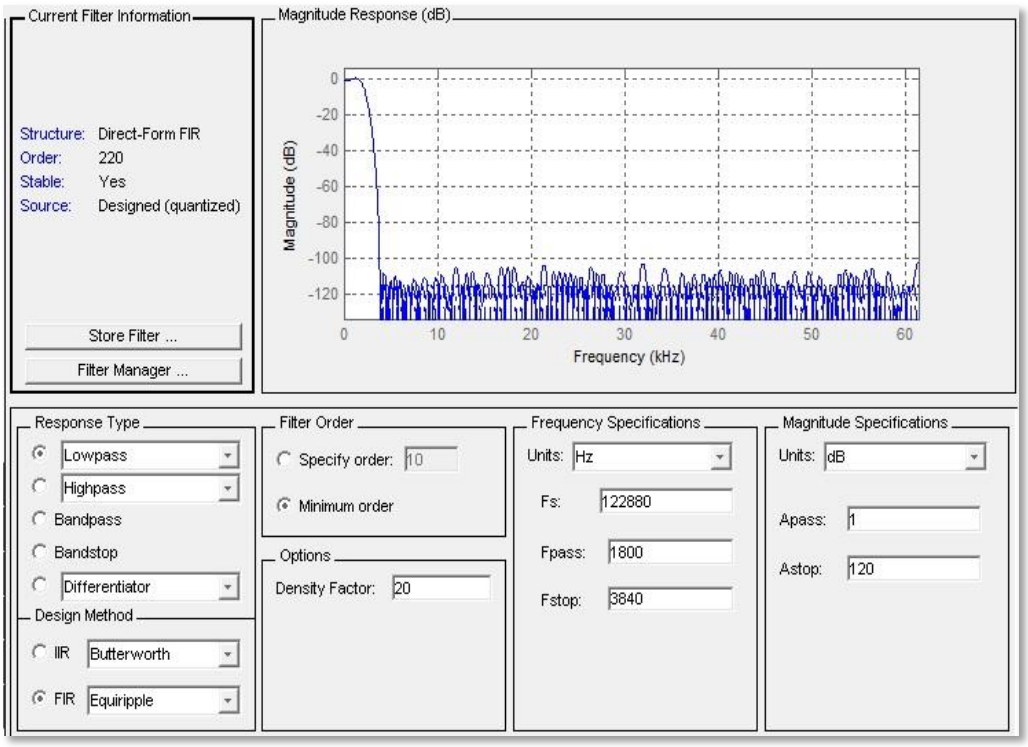


Figura 5.2 – Filtro digital FIR em sua forma direta.

5.2 Otimização

Na Seção 5.1 o filtro digital foi projetado para evitar o efeito de *aliasing* após o *down-sampling* (decimação) do fator de 16. Nesta seção, será proposta uma topologia que melhor aproveita a alteração da taxa de amostragem para reduzir a complexidade computacional do filtro e assim diminuir o tempo de processamento.

Sistemas que alteram a freqüência de amostragem são chamados de sistemas multi-taxas (*Multirate Systems*) e são utilizados quando há necessidade de converter os sinais para uma taxa de amostragem diferente. Por exemplo, os sinais de áudio em um CD têm uma taxa de 44,1 kHz, enquanto uma estação de rádio transmite a uma taxa de 32 kHz [22]. Para passar de uma fonte para outra, é necessário fazer uma conversão de taxa. Outra área que utiliza as técnicas de alteração de taxa é a compressão de sinais. A maneira mais elementar de se comprimir um sinal é jogar fora algumas amostras e armazenar outras, assim realizando um *down-sampling.*

Os componentes básicos para alterar a taxa de um sinal são os módulos chamados de *up-sampler* (aumento da taxa) e *down-sampler* (decimação). Para utilizarmos estes componentes nos sistemas digitais, devemos primeiro realizar uma análise na freqüência dos sinais para evitar que durante a conversão a informação carregada seja corrompida devido ao efeito de *aliasing*. Para evitar este efeito, a alteração da taxa deve ser realizada em conjunto com filtros *anti-aliasing.*

Por outro lado, devemos estar cientes de que ao reduzirmos a taxa de um sinal, estamos limitando a máxima freqüência e assim perdendo parte da informação carregada no sinal. Caso tenhamos que retornar a taxa original, a informação que foi perdida não mais estará presente naquele sinal. Devemos então avaliar a informação carregada pelo sinal antes de manipular sua taxa. Voltando ao exemplo do áudio de CD e do rádio, quando fazemos um *down-sampling* do CD para o rádio, estamos jogando fora as freqüências altas do sinal que são os sons enriquecidos pelo agudo. Ao ouvir uma música, temos a capacidade de perceber que parte da informação foi perdida

neste processo, pois houve uma redução da qualidade. Como ainda conseguimos identificar a música, sabemos que a informação principal foi mantida.

Para alterar a taxa do sinal por um fator que não é um número inteiro, devemos utilizar uma combinação de *up-sampler* e *down-sampler* que realize a conversão pelo fator da taxa de amostragem desejada.

5.2.1 *Down-sampler* e *Up-sampler*

Para verificar o que acontece quando ocorre um *up-sampling* do sinal vamos lembrar a Equação (5.2) que calcula a resposta em freqüência do filtro digital:

$$H(e^{j\omega}) = \sum_{n=0}^{N} h[n]\, e^{-j\omega n} \qquad (5.4)$$

O sinal $h[n]$ representa a resposta ao impulso do filtro $H(z)$ de ordem *N*. Supondo que ocorra uma interpolação do sinal por um fator de 2, gerando o sinal $h_{2u}[n]$ dado por:

$$h_{2u}[n] = \begin{cases} h[n/2], & n = 0, \pm 2, \pm 4 \\ 0\,, & caso\ contrário \end{cases} \qquad (5.5)$$

e substituindo $h_{2u}[n]$ na equação acima, temos que:

$$H_{2u}(e^{j\omega}) = \sum_{n=0}^{N} h_{2u}[n]\, e^{-j\omega n} = \sum_{n=0,\ n\ par}^{N} h[n/2]\, e^{-j\omega n} \qquad (5.6)$$

Substituindo $m = n/2$, temos que [23]:

$$H_{2u}(e^{j\omega}) = \sum_{m=0}^{N} h[m]\, e^{-j\omega 2m} = H(e^{2j\omega}) \qquad (5.7)$$

Analisando a equação $H_{2u}(e^{j\omega})$ e lembrando que o espectro de $H(e^{j\omega})$ é periódico com período 2π, verificamos que o efeito de interpolação (*up-sampling*) por 2, no caso da interpolação, replica o espectro de $H(e^{j\omega})$ em 2 vezes dentro do mesmo

período, ou seja, o período do espectro passa a ser π. A Fig. 5.3(a) mostra graficamente o resultado do efeito *up-sampling* na freqüência.

Analogamente e partindo do mesmo princípio, podemos concluir que o efeito devido a uma decimação (*down-sampling*) provoca um alargamento do espectro de $H(e^{j\omega})$ por um fator de 2, demonstrado pela expressão dada por [24]:

$$H_{2d}(e^{j\omega}) = \frac{1}{2}\left(H\left(e^{\frac{j\omega}{2}}\right) + H\left(e^{j\left(\frac{\omega}{2}+\pi\right)}\right)\right) \qquad (5.8)$$

Caso o sinal $h[n]$ contenha originalmente freqüências acima de $\pi/2$ $(\mathrm{F_S}/4)$, e não for devidamente filtrado, o efeito de *aliasing* irá ocorrer. Este efeito é mostrado na Fig. 5.3(b). As linhas tracejadas definem o limite de freqüências permitido para evitar o *aliasing* (0 a π/2 e 3π/4 a 2π).

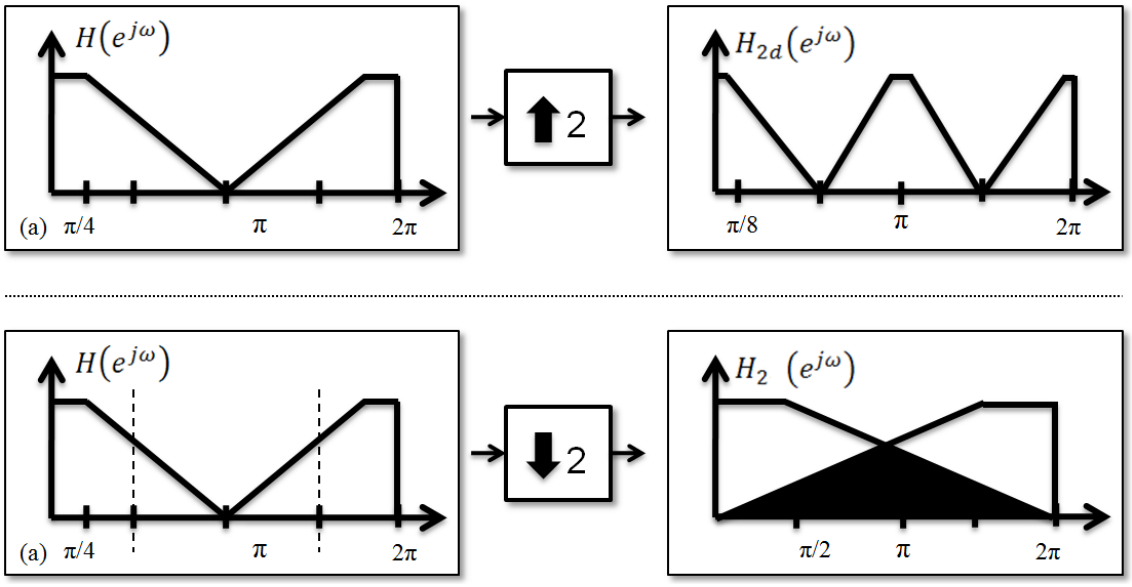


Figura 5.3 – (a) Efeito espectral devido a interpolação de fator 2. (b) Efeito decimação de fator de 2 com *aliasing*.

5.2.2 Projeto com 4 filtros

Para realizar o projeto do filtro *anti-aliasing* otimizado, vamos utilizar dois conceitos importantes. O primeiro é que quanto menor for a faixa de transição do filtro digital, maior será a sua ordem. E o segundo, que ao realizarmos uma redução da

taxa de amostragem devemos antes realizar uma filtragem para evitar o efeito de *aliasing*.

Dessa forma, foi proposto reduzir a taxa de amostragem para depois realizar a filtragem *anti-aliasing* digital sugerida na Seção 4.4. Entretanto, devido à redução da taxa, surge a necessidade de realizar outra filtragem *anti-aliasing*. Como a especificação do projeto exige uma decimação de 16, dividimos esta conversão em dois estágios como mostra a Fig. 5.4.

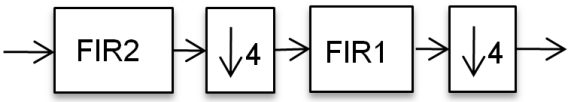


Figura 5.4– Topologia em multi-estágios com *down-sampling* de 4

As freqüências do filtro FIR1 devem ser calculadas para possuir uma resposta em freqüência, na nova taxa, equivalente a do filtro projetado na Seção 5.1. Entretanto, como foi realizado o *down-sampling*, a freqüência de amostragem associada a este filtro torna-se 1/4 da freqüência original. A Tabela 5.2 mostra as freqüências para o filtro FIR1 e sua normalização.

Para calcular as freqüências do filtro FIR2 devemos lembrar que este filtro é projetado para evitar que o efeito de *aliasing* ocorra ao realizar o primeiro *down-sampling*. Assim, devido à ocorrência do alargamento espectral, devemos projetar este filtro para atenuar as freqüências que estão acima de $F_S/8$ de um valor mínimo de $95\ dB$. Como a freqüência de corte também será esticada em quatro vezes, devemos considerar esta freqüência como sendo a F_c para o FIR2. A Tabela 5.3 apresenta os valores e sua normalização para o FIR2.

FIR1	$\boldsymbol{F_c}$	$\boldsymbol{F_a}$	$\boldsymbol{F_S/2}$	$\boldsymbol{F_S}$
Freq. (Hz)	1800 Hz	3840 Hz	15360 Hz	30720 Hz
Freq. (Norm.)	0,0586	0,1172	0,5	1

Tabela 5.2 – Projeto do filtro FIR1 com *down-sampling* de 4, freqüências normalizadas e em Hz.

FIR2	$\boldsymbol{F_c}$	$\boldsymbol{F_a}$	$\boldsymbol{F_S/2}$	$\boldsymbol{F_S}$
Freq. (Hz)	7200 Hz	15360 Hz	61440 Hz	122880 Hz

Freq. (Norm.)	0,0586	0,1172	0,5	1

Tabela 5.3 – Projeto do filtro FIR2 com *down-sampling* de 4, freqüências normalizadas e em Hz.

Comparando as tabelas, é possível perceber que os filtros FIR1 e FIR2 possuem as mesmas freqüências normalizadas e a mesma banda de transição e com isso os coeficientes que implementam os filtros digitais tornam-se os mesmos. Isto ocorre porque o fator de decimação é o mesmo para os dois casos.

Ao compararmos as bandas de transição do filtro simples da Tabela 5.1 com as dos filtros das Tabelas 5.2 e 5.3, podemos verificar que houve um aumento da banda de transição em quatro vezes. Assim, os filtros FIR1 e FIR2 serão quatro vezes menores em sua ordem que o filtro projetado na Seção 5.1. Associado ao fato que após o *down-sampler* a taxa de dados diminui, esta estrutura torna-se mais rápida do que o filtro simples. Os cálculos de complexidade computacional para os diferentes filtros serão apresentados nas Equações (5.18), (5.19) e (5.20).

Utilizando este conceito, podemos avançar mais um passo na otimização re-utilizando a estrutura multi-estágios da Fig. 5.4. Se esta estrutura é mais rápida que a decimação em um estágio de 16, então a estrutura apresentada na Fig. 5.5 será mais rápida do que a apresentada na Fig. 5.4. Podemos comprovar este fato nos cálculos de complexidade computacional da Seção 5.3.

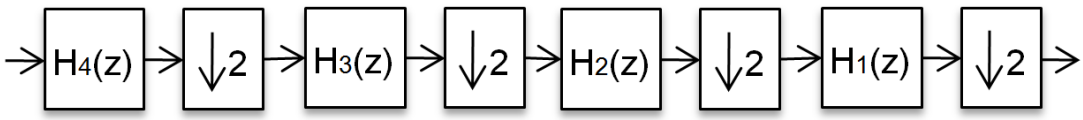


Figura 5.5 – Topologia multi-estágios com *down-sampling* de 2.

Utilizando a mesma idéia, podemos calcular as freqüências para o caso da Fig. 5.5. Os resultados dos cálculos são apresentados nas Tabelas 5.4 e 5.5. Observando as freqüências dos quatro filtros, percebemos que todos eles possuem a mesma freqüência normalizada, como no caso anterior, resultando em filtros com os mesmos

coeficientes. Novamente, isto ocorre porque o fator de decimação é o mesmo para os quatro filtros.

	H_4	H_3	H_2	H_1
F_S (Hz)	122880	61440	30720	15360
F_c (Hz)	14400	7200	3600	1800
F_a (Hz)	30720	15360	7680	3840

Tabela 5.4 – Projeto dos 4 filtros, freqüências em Hz.

	H_4	H_3	H_2	H_1
F_S (norm.)	1	1	1	1
F_c (norm.)	0,117	0,117	0,117	0,117
F_a (norm.)	0,250	0,250	0,250	0,250

Tabela 5.5 – Projeto dos 4 filtros, freqüências normalizadas.

5.2.3 Comparação das Respostas em Freqüência

Para comparar a resposta em freqüência do projeto convencional do filtro FIR com a do projeto do FIR otimizado, foi realizado um script no Matlab que calcula as respostas em freqüência dos filtros considerando as decimações. Para achar a resposta ao impulso resultante do diagrama da Fig. 5.5 podemos entender que o filtro $H_1(z)$ teve a sua resposta ao impulso $h_{H_1}[n]$ decimada por um fator de oito e assim, se realizarmos uma interpolação de oito em sua resposta ao impulso, poderemos encontrar seu equivalente, $h_{H_1}^{08}$, antes do *down-sampler* na taxa de amostragem oito vezes maior. O mesmo acontece quando $h_{H_2}[n]$ passa por um decimador de 4 e $h_{H_3}[n]$ por um decimador de 2 e podemos encontrar os equivalentes $h_{H_2}^{04}$ *e* $h_{H_3}^{02}$. A Fig. 5.6 apresenta um filtro matematicamente equivalente ao filtro da Fig. 5.5.

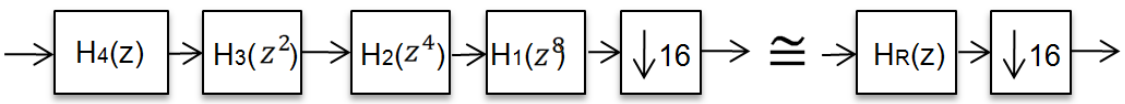


Figura 5.6 – Topologia equivalente para o filtro otimizado.

Podemos então expressar o filtro resultante H_R de forma que ele seja o produto na freqüência dos filtros H_1, H_2, H_3 e , H_4 tal que:

$$H_R(z) = H_4(z)\, H_3(z^2)\; H_2(z^4)\, H_1(z^8) \qquad (5.9)$$

Utilizando a Equação 5.1 do filtro FIR $H(z)$, temos que a função de transferência dada por:

$$H(z) = h_0 + \; h_1 z^{-1} + h_2 z^{-2} \; + h_3 z^{-3} + \cdots \qquad (5.10)$$

com sua resposta ao impulso $h_H(n)$ dada por:

$$h_H(n) = \; h_0\delta(n) + \; h_1\delta(n-1) + h_2\delta(n-2) + \cdots \qquad (5.11)$$

Assim, função de transferência de um filtro $H(z^2)$, interpolado por um fator de 2, é:

$$H(z^2) = h_0 + \; h_1 z^{-2} + h_2 z^{-4} \; + h_3 z^{-6} + \cdots \qquad (5.12)$$

com sua resposta ao impulso $h_H^{02}(n)$ expressa por:

$$h_H^{02}(n) = \; h_0\delta(n) + \; h_1\delta(n-2) + h_2\delta(n-4) + \cdots \qquad (5.13)$$

Da mesma forma, a resposta de um filtro $H(z^4)$, interpolado por um fator de 4, é dada por:

$$H(z^4) = h_0 + \; h_1 z^{-4} + h_2 z^{-8} \; + h_3 z^{-12} + \cdots \qquad (5.14)$$

com sua resposta ao impulso $h_H^{04}(n)$ expressa por:

$$h_H^{04}(n) = \; h_0\delta(n) + \; h_1\delta(n-4) + h_2\delta(n-8) + \cdots \qquad (5.15)$$

E assim sucessivamente para $h_H^{08}(n), h_H^{016}(n), h_H^{032}(n)$ ...

Podemos então calcular a resposta ao impulso do filtro final resultante, $h_{HR}(n)$, como a convolução das respostas ao impulso dos filtros equivalentes, tal que:

$$h_{HR}(n) = conv(\,\{h_{H_4}\}, \{h_{H_3}^{02}\}, \{h_{H_2}^{04}\}, \{h_{H_1}^{08}\}\,) \qquad (5.16)$$

em que $conv$ representa o operador de convolução.

O *script* completo, que calcula a resposta em freqüência do filtro resultante, pode ser consultado no Apêndice A6. A Fig. 5.6 mostra a comparação entre as

respostas em freqüência do filtro FIR realizado com o projeto convencional e a resposta em freqüência do filtro otimizado com decimadores de 2.

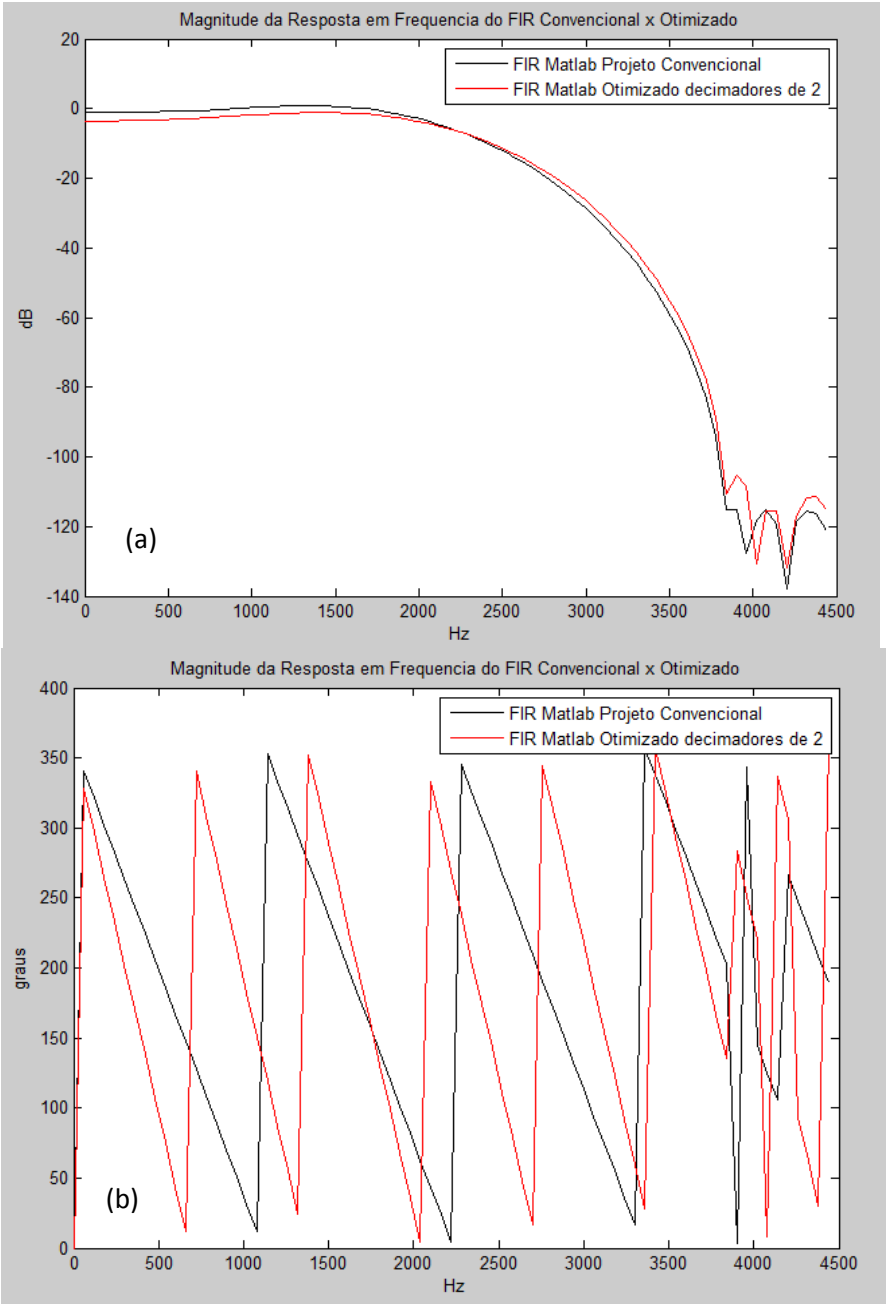


Figura 5.6 – Resposta em freqüência Filtro FIR x Otimizado, magnitude (a) fase (b).

Observando os gráficos é possível perceber as diferenças nas respostas em freqüência dos filtros. O filtro otimizado possui um *ripple* maior na banda de passagem e com isso sua atenuação nesta banda é maior do que a do filtro FIR do projeto convencional. Isto ocorre porque H_1, H_2, H_3 *e* H_4 possuem a mesma resposta em

freqüência e esta possui um suave *ripple* na banda de passagem. Quando utilizamos a configuração dos quatro filtros para otimizar o filtro final, este efeito é amplificado, pois as amostras passam pelo mesmo filtro quatro vezes. O aumento do *ripple* poderia estar associado à precisão finita dos coeficientes do filtro (quantização), entretanto os cálculos aqui realizados utilizam coeficientes em ponto-flutuante para ambos os casos para assim evitar quaisquer erros provocados pela quantização dos coeficientes dos filtros. A Fig. 5.7 mostra o gráfico de resposta em freqüência dos quatro filtros na qual observamos $H_1(z^8)$ no canto inferior direito, $H_2(z^4)$ no inferior esquerdo, $H_3(z^2)$ no superior direito. Podemos verificar que realmente existe um *ripple* na banda de passagem de cada módulo de filtragem.

Quando comparamos as respostas de fase dos filtros, verificamos que todas são lineares e que há uma diferença de fase entre elas que cresce ao longo da freqüência, como será explicado na Seção 7.5, onde são apresentados os resultados finais. Como a ordem resultante do filtro otimizado é de 96 (4 filtros de ordem 24 cada) e a ordem do filtro com projeto convencional é de 220, este gasta mais tempo calculando os resultados e com isso o intervalo de tempo entre a entrada a saída é maior. Como a freqüência aumenta ao longo do gráfico, apesar da diferença de tempo dos atrasos de resposta dos filtros se manter constante, a diferença de fase aumenta ao longo da freqüência de forma linear.

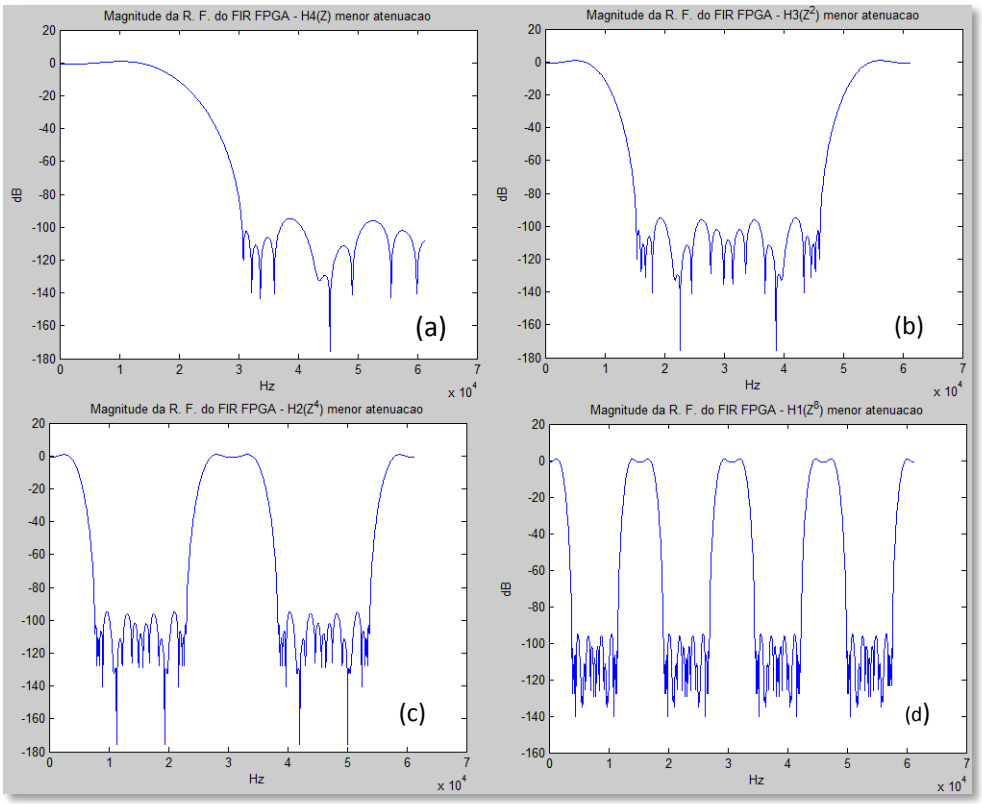


Figura 5.7 – Resposta em freqüência dos filtros: (a) FIR4 sem interpolação, (b) FIR3 interpolado de um fator de 2, (c) FIR2 interpolado de 4 e (d) FIR1 interpolado de 8.

Podemos aproveitar a Fig. 5.7 para observar de forma simulada o efeito espectral devido a uma interpolação. O sinal $h_{H_4}[n]$ tem sua transformada discreta de Fourier representada por $H_4(z)$, como mostra o gráfico Fig. 5.7(a). Quando interpolamos este sinal por um fator de 2, gerando o sinal $h_{H_4}^{02}$, o espectro deste sinal é replicado como mostrado no gráfico Fig. 5.7(b). O mesmo ocorre com os filtros $H_2(z)$ e $H_1(z)$ quando ocorre uma interpolação de 4 e 8, com o efeito de repetição espectral apresentados nos gráficos da Fig. 5.7(c) e Fig. 5.7(d), respectivamente.

5.3 Comparação das complexidades

Ao observamos o diagrama de blocos da Fig. 5.1 e considerando o *hardware* da Fig. 6.2, que será discutido no Capítulo 6, notamos que a complexidade computacional referente aos filtros FIR pode ser estimada calculando-se o número de operações de multiplicações das entradas pelos coeficientes do filtro. Sendo assim, podemos definir

que o número de operações de multiplicações é proporcional à quantidade de coeficientes do filtro e também à taxa de amostragem dos dados de entrada. Expressando isto matematicamente, pode escrever que:

$$C_{FIR} = K\, O_{FIR} Fs \qquad (5.17)$$

em que C_{FIR} representa a complexidade computacional do filtro FIR do projeto convencional, O_{FIR} sua ordem e K é uma constante de normalização. Substituindo os valores, temos que:

$$C_{FIR} = K\,.\ 220\ .\ \ 122880 = 27{,}03\ 10^6\ K \qquad (5.18)$$

Da mesma forma podemos calcular a complexidade do filtro FIR com decimação de 4, lembrando que a ordem dos sub-filtros é igual a 50, temos que:

$$C_{down4} = K\,.50\,(\ 122880 + 30720) = 7{,}68\ 10^6\ K \qquad (5.19)$$

Além disso, podemos calcular a complexidade do filtro FIR otimizado com decimadores de 2, que foi implementado por possuir melhor desempenho, tal que:

$$C_{otimz} = K\,.24\,(\ 122880 + 61440 + 30720 + 15360) \qquad (5.20)$$

$$C_{otimz} = 5{,}53\ 10^6\ K$$

Assim, como o *clock* do circuito se mantém o mesmo para os dois casos, o ganho de complexidade se refletirá em um ganho de velocidade no cálculo dos resultados, o qual pode ser definido por:

$$G_{comp1} = \frac{C_{FIR}}{C_{otimz}} \cong 4{,}9 \qquad (5.21)$$

$$G_{comp2} = \frac{C_{down4}}{C_{otimz}} \cong 1{,}4 \qquad (5.22)$$

Concluímos então que o filtro otimizado é, teoricamente, 4,9 vezes mais rápido do que o filtro FIR do projeto convencional e 1,4 vezes mais rápido do que o filtro com decimadores de 4.

Além disso, como será verificado no final do Capítulo 7, este resultado mostrou-se diferente do resultado encontrado empiricamente devido a diferenças de circuito de hardware. Como o filtro do projeto convencional é maior, exige mais acesso à memória de dados, e assim este tempo deve ser levado em consideração. O ganho empírico de velocidade obtido com o filtro otimizado, em relação ao projeto convencional do FIR, foi de aproximadamente nove vezes.

Capítulo 6 Implementação em FPGA

Como mencionado no capítulo anterior, a implementação dos filtros digitais pode ser realizada através de hardware ou software. Quando feita por software, uma CPU (processador) é utilizada para executar um conjunto de instruções (programa) de forma seqüencial em um DSP (Digital Signal Processor) ou um Microcontrolador. Os programas são escritos em liguagem C, Assembly ou outra equivalente, compilados e gravados nos controladores.

Quando a implementação é feita através de hardware, circuitos integrados (ASIC – *Application Specific Integrated Circuit*) ou circuitos lógicos programáveis (FPGA, CPLD, etc.) são projetados para desempenharem as funções desejadas. Para configurar estes dispositivos, um esquema de hardware deve ser realizado ou uma linguagem de descrição de hardware (HDL) deve ser utilizada.

Os FPGAs são formados por um conjunto de portas lógicas, chamados de blocos lógicos (CLBs), portas de entrada e saída (IOBs) e trilhas de interconexão. Os blocos lógicos são compostos por um conjunto de células lógicas formadas por LUTs (*lookup table*), um somador e *flip-flops* tipo D [25]. Os FPGAs também possuem memórias RAM e multiplicadores que são utilizados para desempenhar funções mais elaboradas, como a estrutura de um filtro digital, que necessita armazenar valores dos coeficientes para realizar as multiplicações e somas. Além disso, estes dispositivos possuem uma unidade de gerenciamento de *clock* (DCM), que é responsável por gerar a velocidade de *clock* necessária para cada área específica do circuito.

Como visto no Capítulo 3, as unidades de aquisição são sincronizadas através de um sinal de *trigger* referenciado ao GPS. Este sistema de aquisição sincronizada é controlado através do FPGA. Como este dispositivo já havia sido escolhido para compensar os atrasos e disparar os conversores A/Ds de forma sincronizada, foi proposto utilizá-lo para implementar o filtro digital *anti-alising*, evitando a necessidade de outro controlador para a UAS.

O FPGA XC3S50AN utilizado neste projeto possui 50.000 portas lógicas, que formam 176 blocos lógicos, três *block RAM* de 18 Kbit, três multiplicadores, dois DCMs e 108 portas lógicas [26]. A arquitetura deste FPGA é mostrada na Figura 6.1.

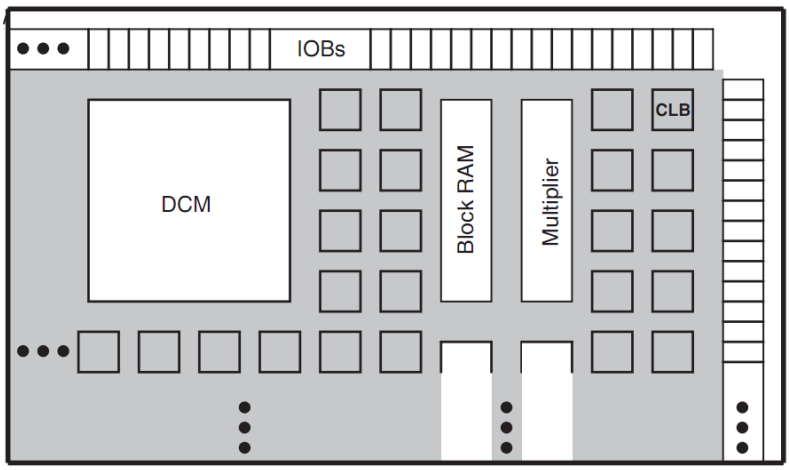


Figura 6.1 – Arquitetura interna do FPGA XC3S50AN. Fonte: Spartan-3AN datasheet.

6.1 Arquitetura do Filtro Digital

Como visto no Capítulo 5, o diagrama de blocos que realiza um filtro digital FIR na sua forma direta é mostrado na Fig. 5.1. Para formar esta estrutura, os componentes do FPGA utilizados são as memórias (*block RAM*) e os multiplicadores com acumuladores (MACs). A arquitetura do filtro digital dentro do FPGA é representada pelo diagrama de blocos da Fig. 6.2.

Nesta arquitetura, o sinal *x[n]* passa por uma *block RAM*, que armazena as amostradas passadas. Uma segunda *block RAM* armazena os valores de *h[n]*, coeficientes do filtro, de acordo com o filtro desejado. Caso mais de um filtro seja implementado, uma mesma *block RAM* pode conter os coeficientes dos diferentes filtros. O multiplicador realiza a multiplicação entre os valores de *h[n]* e *x[n].* O registrador de 34 bits é utilizado para guardar e acumular os resultados intermediários das multiplicações de acordo com a convolução representada pela Equação (5.3).

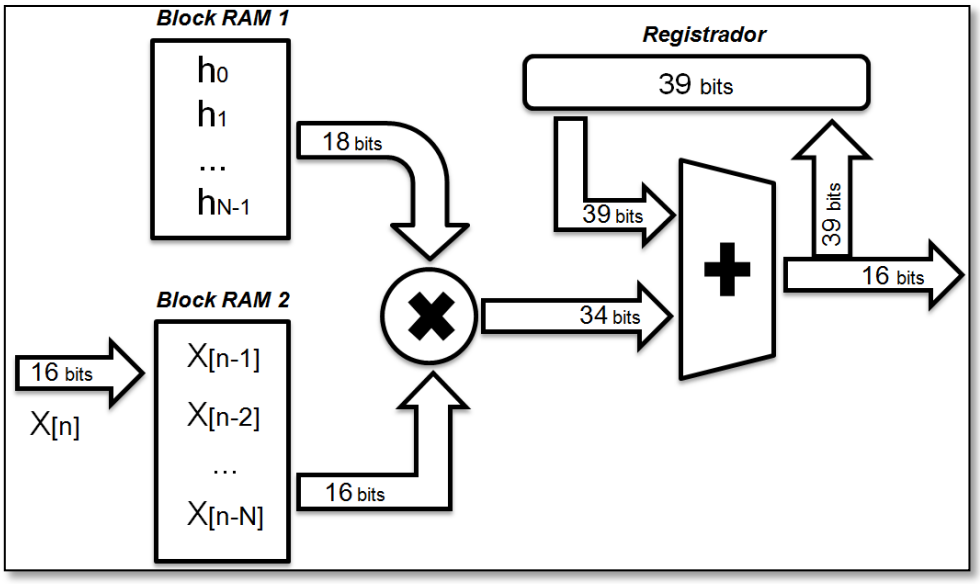


Figura 6.2 – Diagrama de blocos do hardware do FIR dentro do FPGA.

O FPGA utilizado é um hardware de ponto fixo. Dessa forma, é necessário especificar a quantidade de bits em cada parte do circuito de forma a manter a razão sinal ruído (SNR) mínima de 16 bits (95 dB). Como cada endereço das memórias *block RAM* possui um registrador de 18 bits, foi proposto utilizar este número para a resolução de bits dos coeficientes dos filtros no processo de conversão de ponto flutuante para ponto fixo, pois assim utiliza-se a máxima resolução possível.

Por outro lado, não há necessidade de armazenar as amostras passadas do sinal de entrada x[n] em 18 bits, pois este sinal vem do A/D, que já possui uma resolução máxima de 16 bits. Quando ambos os sinais são multiplicados, um valor de 34 bits é o mínimo necessário para manter a resolução. Como na operação de soma e acumulação (MAC) o valor de 34 bits do último resultado é reutilizado, é necessário estimar o número de bits que minimize o erro que é integrado ao longo das operações de multiplicações com acumulações. Assim, como a ordem do filtro define o número de operações de MAC, este valor é estimado durante o projeto dos núcleos como será visto na próxima seção.

6.2 Projeto dos núcleos dos filtros

A Xilinx, fabricante do FPGA, disponibiliza na web o *ISE Design Suite 12* [27], um pacote de desenvolvimento com licença gratuita (*webpack*). No ambiente do ISE é possível sintetizar uma eletrônica digital através da linguagem de descrição de *hardware* (VHDL). Além disso, o ISE permite projetar os núcleos dos filtros utilizando o *LogiCORE™ IP FIR Compiler* [28] e escrever um programa de teste de emulação do *hardware* (*testbench*) para verificação do filtro, como será visto na Seção 6.3.

O FIR Compiler é um módulo que proporciona uma interface com o usuário para projeto de filtros FIR de alto desempenho, ocupando uma área de *hardware* eficiente. Este módulo é capaz de projetar diferentes tipos de estruturas de filtros FIR, entretanto, ele foi utilizado para projetar filtros sem alteração de taxa, implementados na forma direta. Para a implementação otimizada (multi-taxas), quatro filtros em taxas fixas diferentes foram compilados. A decimação e a lógica de conexão entre os quatro filtros foram descritas através do VHDL. A Fig. 6.2 mostra a interface de projeto de um estágio do filtro FIR otimizado.

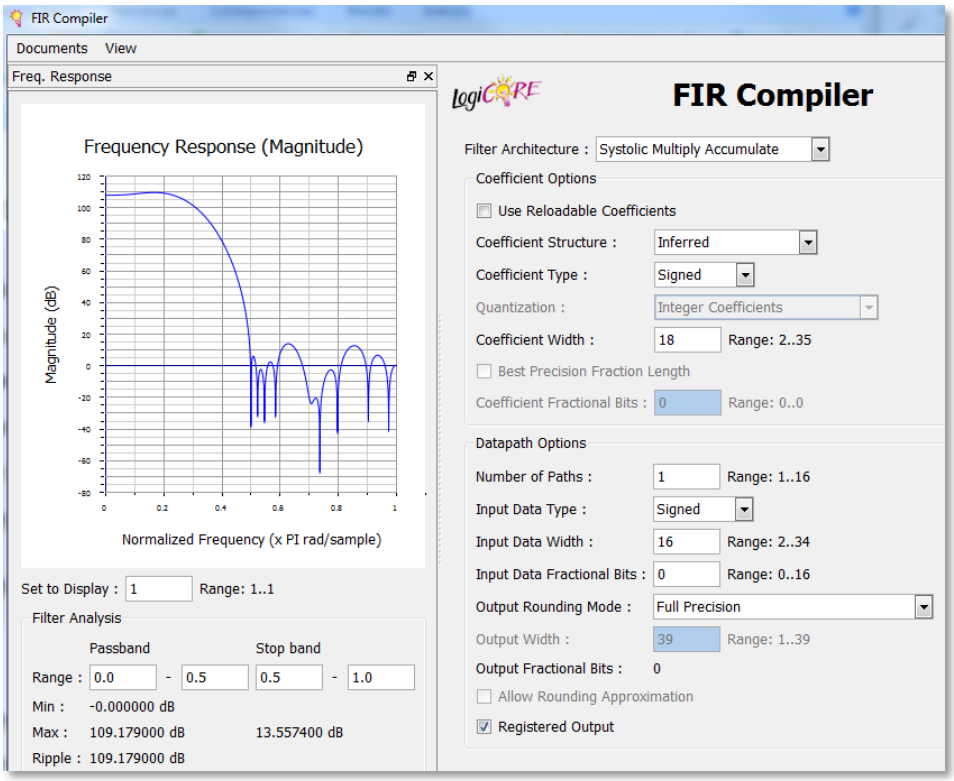


Figura 6.2 – Interface de projeto do FIR simples usando o FIR Compiler.

Ao projetarmos os filtros no Matlab, exportamos os coeficientes para um arquivo no formato “coe” e inserimos através da interface do FIR Compiler. Definimos ainda a resolução em bits dos dados de entrada e saída e os coeficientes do filtro. Após entrar com estes valores, a interface apresenta o gráfico de resposta em freqüência do filtro configurado e a resolução ótima em bits para a saída. Ao compararmos o gráfico da Fig. 6.2 com o da Fig. 5.7, notamos que as resposta em freqüência são iguais, como esperado.

Uma vez compilado, o módulo apresenta-se como um bloco disponível para ser conectado com o restante do circuito através dos seus pinos de entrada e saída. A Fig. 6.3 mostra este bloco.

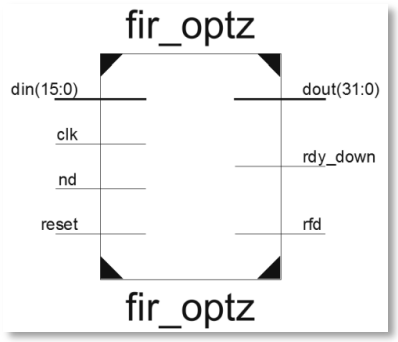


Figura 6.3 – Bloco lógico do Filtro FIR projetado com o FIR Compiler.

Assim, devemos conectar estes pinos no restante do circuito. Os esquemas completos de formação dos filtros são mostrados no Apêndice A7 ao final deste trabalho.

Uma máquina de estados foi projetada e incluída junto ao filtro para controlar o seu funcionamento. A entrada de dados do filtro, *din*, é conectada o circuito do A/D. A saída *dout* é conectada com o circuito de envio dos dados para o Concentrador. O sinal *rfd* (*ready for data*) é um sinal de controle, que indica que o filtro está pronto para receber um novo dado na entrada *din*. O sinal de controle *nd* (new data) indica que um novo valor foi escrito na saída. O sinal de controle *clock* fornece a temporização do circuito. No caso do filtro otimizado, mais um sinal de controle foi criado, o *rdy_down*, que indica que a amostra de entrada deve ser descartada para realizar o *down-sampling*.

6.3 *Testbench* VHDL

O VHDL é uma linguagem de descrição de circuitos lógicos que pode ser utilizada tanto para simular como para projetar um hardware digital [29]. Quando o código for desenvolvido para descrever uma lógica de circuito, este VHDL pode ser sintetizado por um programa, dando origem ao hardware. Por outro lado, o VHDL permite descrever rotinas que são abstratas e por isso não podem ser sintetizadas gerando o hardware. Os códigos não-sintetizáveis são criados para aplicar um conjunto de estímulos na unidade de teste e verificar o funcionamento subseqüente. Um conjunto de códigos de simulação é normalmente chamado de *testbench*.

Neste projeto, o *testbench* foi desenvolvido para aplicar as entradas nos filtros implementados no FPGA e armazenar os valores de saída em um arquivo de texto para ser interpretado posteriormente pelo Matlab. Além do sinal de entrada do filtro, o *testbench* gerencia os sinais de *clock*, a escrita em disco dos valores da saída e a lógica do gerador de freqüência da função *chirp*. Todos estes códigos são abstrações e são processados pelo programa de simulação e, assim, não são passíveis de síntese de hardware. Os códigos completos em VHDL que descrevem a simulação e a conexão com os filtros FIR projetados na seção anterior podem ser consultados no apêndice A8.

Durante o processo de simulação, a ferramenta de simulação chamada de ISIM [30] permite definir uma freqüência de clock para o circuito de teste observar os sinais de entrada, de saída, de clock e sinais intermediários. Com isso, foi possível definir um clock de 100 MHz e verificar o tempo de resposta para os diferentes filtros. A Figura 6.4 mostra uma janela do ambiente de simulação e os sinais observados. Os resultados das simulações são apresentados ao final do Capítulo 7.

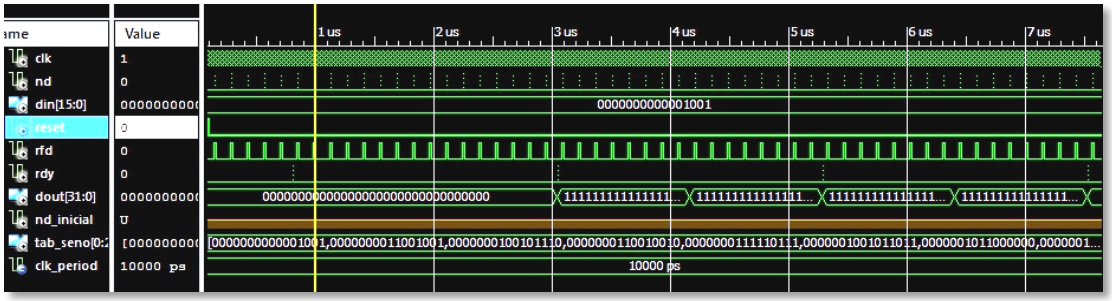


Figura 6.4 – Ambiente de simulação Xilinx ISIM.

Capítulo 7 Metodologia de validação da Resposta em Freqüência

Uma vez implementados os filtros FIR no FPGA, uma metodologia foi proposta para validar a resposta em freqüência dos filtros de forma experimental. Como dito na Seção 6.3, o código do *testbench* escrito em VHDL é capaz de emular as estruturas de *hardware* sintetizadas no FPGA.

Para verificar o funcionamento do filtro, propomos entrar com um sinal no tempo que represente uma senóide de freqüência, amplitude e fase conhecidas. Através da análise da saída, poderemos comparar este sinal com a entrada e, através de algoritmos, aferir as características de ganho e defasagem do filtro naquela freqüência.

Assim, com objetivo de medir a resposta do filtro para as freqüências até a faixa de rejeição, inserimos os 75 primeiros harmônicos de 60 Hz (incluindo a fundamental) na entrada, o que provoca uma resolução espectral com 75 pontos de 60 Hz a 4500 Hz.

Para entendermos melhor os passos realizados para a construção desta metodologia, um fluxograma foi elaborado e é mostrado na Figura 7.1.

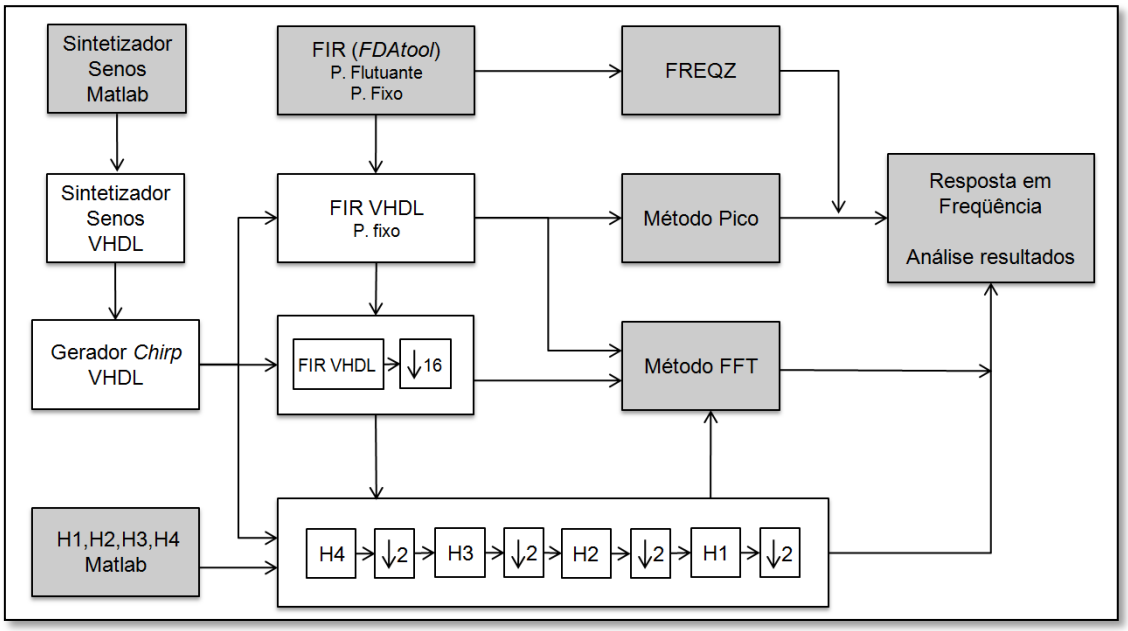


Figura 7.1 – Fluxograma da Metodologia de projeto e verificação.

Neste fluxograma, os blocos escuros representam os procedimentos que foram realizados no Matlab e os blocos claros mostram os procedimentos realizados no VHDL.

No primeiro passo, utilizamos a função *freqz* do MATLAB para construir o gráfico de resposta em freqüência teórico a partir dos coeficientes do filtro FIR realizado com projeto convencional.

No segundo passo, desenvolvemos um algoritmo para sintetizar freqüências diferentes a partir de uma tabela de seno e verificamos seu funcionamento no Matlab. De posse do sintetizador de freqüências, construímos um sinal do tipo *chirp* que contém todos os harmônicos de 60 Hz (60 Hz a 4500 Hz) e implementamos este algoritmo no VHDL.

No terceiro passo, escrevemos um programa no Matlab que reconstrói a resposta em freqüência dos filtros utilizando dois algoritmos. O primeiro é o Método do Pico, que faz uma análise no tempo e identifica o valor máximo do sinal de entrada. E o segundo é o Método FFT (*Fast Fourier Transform*), que faz uma análise na freqüência através da transformada discreta de Fourier (DFT).

No quarto passo, para validar os métodos desenvolvidos, inserimos o sinal *chirp* na entrada do filtro FIR do projeto convencional e armazenamos os valores de entrada e saída em um arquivo texto. Entramos então com este arquivo no Matlab para reconstruir a resposta em freqüência com os métodos do terceiro passo. Como será verificado na Seção 7.5.1, referente aos resultados obtidos, o método da FFT apresentou uma resposta mais precisa e desta forma foi escolhido para os próximos passos.

No quinto passo, incluímos um *down-sampler* de 16 ao filtro do VHDL e armazenamos a resposta em outro arquivo texto. Entramos com o sinal *chirp* e utilizamos o método FFT para validar a resposta em freqüência utilizando uma metodologia semelhante ao quarto passo. O mesmo é feito para o filtro FIR otimizado.

Por último, comparamos os resultados obtidos para as diferentes topologias.

Neste capítulo, detalhamos o funcionamento dos procedimentos utilizados na metodologia de validação da resposta em freqüência dos filtros FIR do FPGA e, ao final, realizamos uma comparação e análise dos resultados obtidos.

7.1 Sintetizador de Senos

Como mencionado no final do Capítulo 5, o *testbench* é capaz de simular o hardware sintetizado pelo VHDL. O algoritmo para gerar diferentes freqüências que irão construir o sinal de entrada para testar o filtro foi testado no Matlab e implementado na parte não-sintetizável do VHDL. O código escrito em Matlab que testa o gerador de senos também escreve um arquivo de saída com um código em VHDL para formar uma tabela de seno no *testbench* e pode ser consultado no apêndice A9.

Lembramos que a taxa de amostragem na entrada do filtro é de $\mathrm{F_s} = 122880\ \mathrm{A/s}$ e, portanto, a freqüência de estímulo para o filtro $(\mathrm{F_{in}})$ deve ser sintetizada com esta taxa.

Como desejamos gerar os 75 primeiro harmônicos de 60 Hz, para sintetizar a freqüência de entrada mínima ($\mathrm{F_{in}} = 60\ \mathrm{Hz}$), precisamos de 2048 pontos para gerar um período, como pode ser verificado na expressão:

$$\mathrm{F_{in}} = \frac{\mathrm{F_s}}{\mathrm{N}} = \frac{122880}{2048} = 60\ \mathrm{Hz} \qquad (7.1)$$

e assim o tamanho da tabela é de 2048 pontos.

Para gerar um sinal de 120 Hz passamos duas vezes pela tabela de forma circular pegando um ponto a cada dois, ou seja, $\mathrm{Salto} = 2$, e para 180 Hz com $\mathrm{Salto} = 3$, assim sucessivamente. Então, podemos escrever que a freqüência de entrada é dada por:

$$\mathrm{F_{in}} = \mathrm{F_S}\frac{\mathrm{Salto}}{\mathrm{N}} \quad (\mathrm{Hz}) \qquad (7.2)$$

Segundo o teorema de Nyquist, com uma freqüência de amostragem de $\mathrm{F_S} = 122880$ A/s é possível sintetizar uma freqüência de até $\mathrm{F_{in}} = 61440$ Hz (excluindo este valor). Entretanto, na prática este resultado não é utilizado, pois exige que algoritmos de interpolação sejam utilizados para reconstruir o sinal amostrado por aquela taxa. Um exemplo da interpolação pode ser visualizado na Figura 7.5. Entretanto, como o filtro atenua freqüências acima de 3840 Hz em 95dB, para verificar a curva de resposta

em freqüência iremos injetar as freqüências apenas até 4500 Hz. A Figura 7.2 mostra um sinal de 6 kHz gerado a partir da tabela de seno.

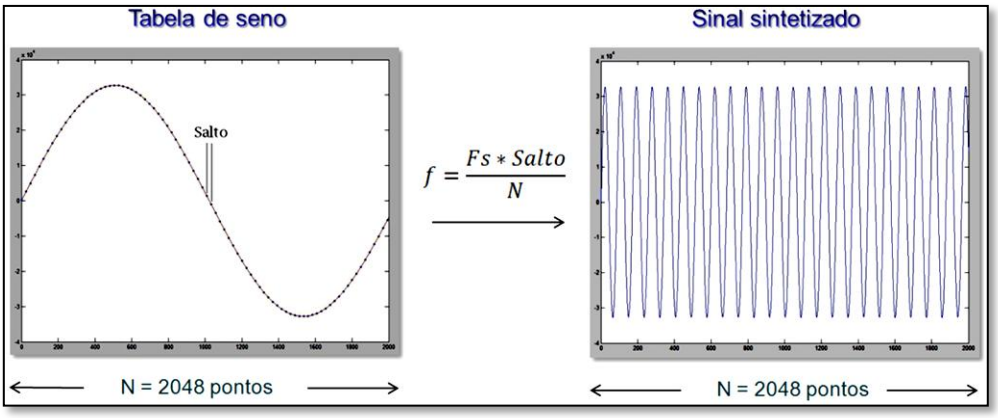


Figura 7.2 – Exemplo de síntese de um sinal de 6 KHz a partir da tabela de seno.

Expressando em termos matemáticos o que ocorre com o gerador de senos, definimos o Salto na tabela tal que:

$$\text{Salto} = \frac{\text{F}_{\text{in}}}{60\ \text{Hz}} \qquad (7.3)$$

Para $\text{F}_{\text{in}} = 60, 120, 180, \ldots 4500\ \text{Hz}$. Assim, a tabela tem um período de seno que varia de -32767 a 32767, em ponto fixo com 16 bits de resolução. O sinal da tabela $\text{t}[\text{n}]$, para $1 < n < 2048$ é definido como:

$$\text{t}[\text{n}] = 32767\ \text{sin}(2\pi\text{n}) \qquad (7.4)$$

Para sintetizar um sinal que possui uma freqüência de entrada $\text{x}_{\text{F}_{\text{in}}}[\text{n}]$, temos que:

$$\text{x}_{\text{F}_{\text{in}}}[\text{n}] = 32767\ \text{sin}(\ 2\pi\text{n}\ \ \text{Salto}) \qquad (7.5)$$

Definindo o operador mod (resto da divisão) tal que:

$$\text{x}_{\text{F}_{\text{in}}}[\text{n}] = \text{t}\left[\ \text{mod}\left(\frac{\text{F}_{\text{in}}}{\text{F}_{60\text{Hz}}}\ \text{n}\ , \text{N}\right)\right] \qquad (7.6)$$

em que $\text{N} = 2048$ pontos, que é o tamanho da tabela.

7.2 O Sinal *Chirp*

A definição de chirp é de um sinal em que a freqüência aumenta ou diminui ao longo do tempo. Neste trabalho, a função chirp é um sinal que utiliza o sintetizador de senos para aumentar a freqüência de 60 Hz em 60 Hz até 4500 Hz, cada freqüência possuindo o mesmo número de amostras coladas uma ao lado da outra. A Fig. 7.3 mostra o sinal que forma a função Chirp.

Como foi visto na seção anterior, o sintetizador de freqüências utiliza 2048 pontos para formar a freqüência mais baixa de 60 Hz. Ao colar os períodos dos senos, geramos descontinuidades na transição de freqüência e esta mudança abrupta componentes de freqüências altas. Assim, para desconsiderar este efeito devemos aguardar K/2 amostras na saída do filtro após a transição de freqüência, K sendo a ordem do filtro. Foi escolhido então, por efeitos de praticidade, o valor de K = 300 amostras. Com isso, o número de amostras de cada componente senoidal é de 2348 pontos.

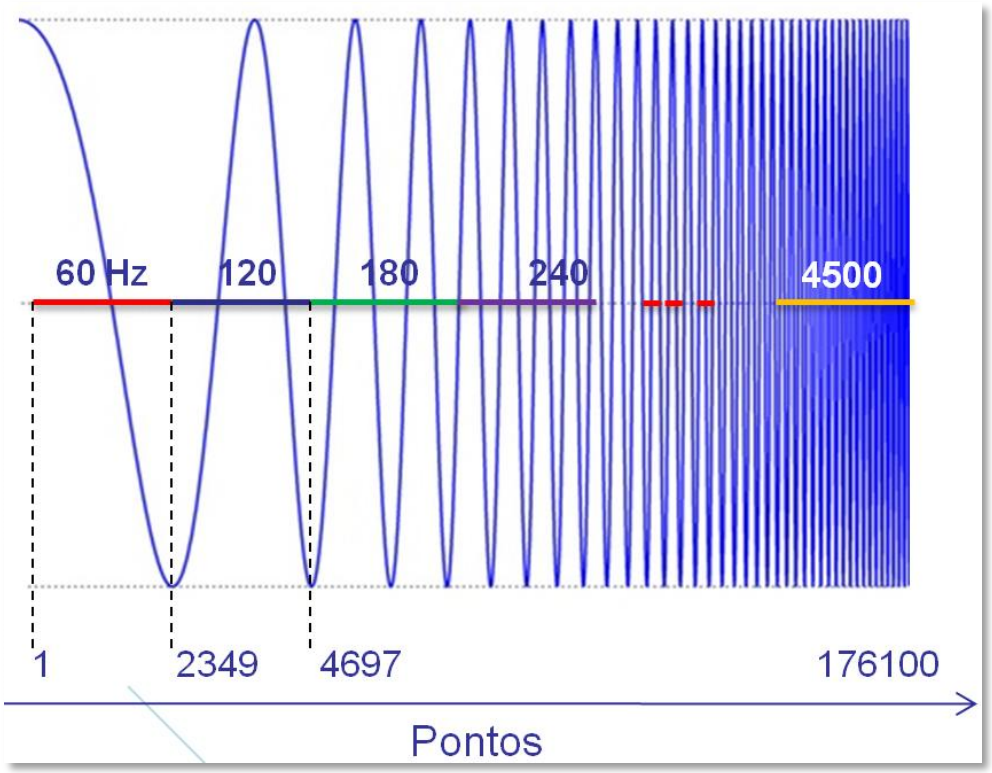


Figura 7.3 – Função Chirp gerada com o sintetizador de senos.

7.3 Análise no tempo: Método Pico

Para verificar a resposta em freqüência do filtro digital analisando as amostras de entrada e saída no tempo, o Método do Pico foi desenvolvido. A base deste algoritmo é achar os valores de amplitude máximos tanto na entrada como na saída e o numero de amostras que existe entre estes picos, que corresponde ao atraso de um sinal em relação ao outro. De posse destes valores, podemos facilmente calcular as características de ganho e defasagem do filtro para a freqüência em questão, que são dados por:

$$G_{dB} = 20\ log_{10}\left(\frac{A_{out}}{A_{in}}\right) \qquad (7.7)$$

em que G_{dB} é definido como ganho e A_{in} e A_{out} são as amplitudes de entrada e saída, respectivamente. Podemos ainda definir o cálculo da *fase* do filtro como:

$$fase_{rad} = -2\pi\ M\ d \qquad (7.8)$$

em que $d = (n_{out} - n_{in})$ é o atraso do sinal de saída com relação a entrada e $M = \frac{F_S}{F_{in}}$ é o período em número de amostras da freqüência de entrada. A Figura 7.4 mostra graficamente os sinais de entrada e saída e seus parâmetros.

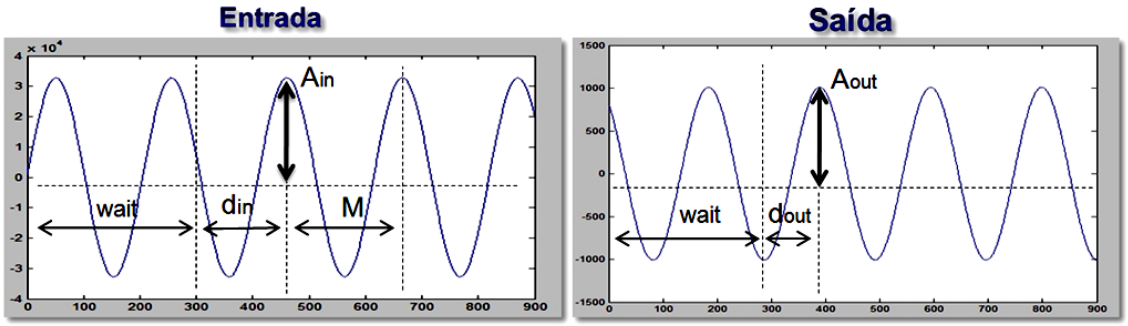
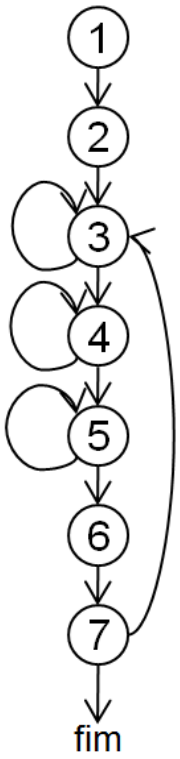


Figura 7.4 – Exemplo gráfico do Método do Pico

Dessa forma um algoritmo em Matlab foi desenvolvido para receber um arquivo texto, contendo os valores de entrada e saída, e construir os gráficos de ganho e de fase da resposta em freqüência do filtro. Podemos então definir o algoritmo do Método Pico como:



1) Insere-se o arquivo texto no Matlab e separam-se as freqüências, tanto das entradas como das saídas.
2) Realiza-se a interpolação para aumentar a resolução dos sinais.
3) Descartam-se os K primeiros pontos do transiente da entrada.
4) Procura-se nos próximos M pontos a posição referente a amostra com o valor de pico do sinal de entrada (*din*). Guarda-se o valor de amplitude A_{in} neste ponto.
5) Da mesma forma, procura-se o pico do sinal de saída (*dout*). Guarda-se o valor de amplitude A_{out}, neste ponto.
6) Calcula-se ganho e fase para a freqüência usando as Equações (7.7) e (7.8).
7) Repete para a próxima freqüência, se não houver mais, fim.

Uma versão de teste escrita em Matlab pode ser consultada no apêndice A10 e a versão completa do script em Matlab que mostra os gráficos da resposta em freqüência, utilizando a metodologia do Capítulo 7, é mostrada no apêndice A9.

Um ponto importante a ser comentado sobre a análise da resposta em freqüência no tempo está relacionado à resolução das freqüências de entrada. A Fig. 7.5 mostra um exemplo de um sinal de alta freqüência gerado pelo sintetizador de senos. O algoritmo utiliza o interpolador para melhorar a precisão do método de análise no tempo.

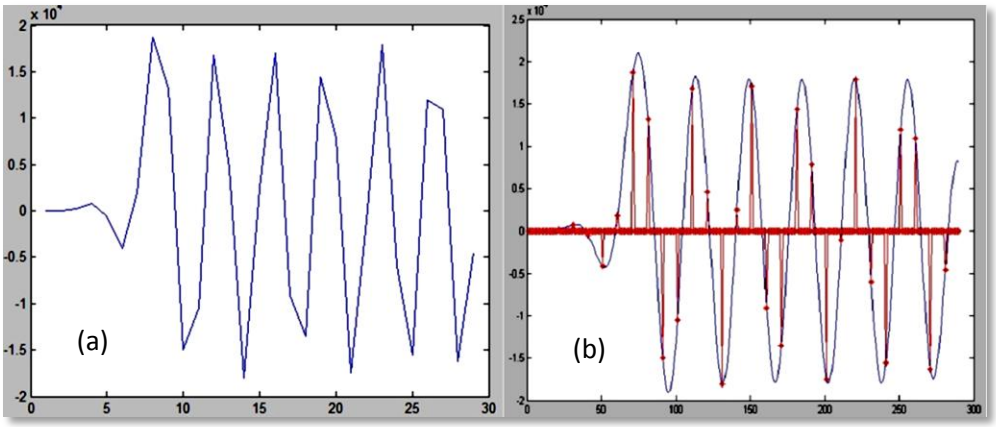


Figura 7.5 – Exemplo de um sinal. (a) Sem interpolação. (b) Com interpolação por um fator de 10.

Podemos perceber que o sinal da Fig. 7.5(a) possui seu valor de pico que varia ao longo dos períodos. Para resolver este problema utilizamos uma função do Matlab chamada de *interp* [31] com a razão de 1 para 10, foi utilizada. O efeito desta função pode ser visualizado no gráfico da Fig.7.5(b). Podemos verificar que os pontos originais que caracterizam a freqüência de entrada se mantêm. O que esta função faz é aumentar a taxa de amostragem do sinal inserindo zeros na seqüência original e aplicando um filtro passa-baixas. Podemos observar que o gráfico interpolado possui 10 vezes mais pontos.

7.4 Análise na freqüência: Método FFT

O algoritmo de análise na freqüência utiliza como base a transformada discreta de Fourier (DFT) e seu algoritmo rápido (FFT). A transformada de um sinal x[n] já carrega a informação espectral de magnitude e fase. Desta forma, o que esta lógica deve fazer é aplicar a DFT para cada componente senoidal da entrada e procurar a amostra relativa ao maior valor na curva de magnitude. Definimo-la como: p_{max}. Em seguida, calculamos a DFT para a saída e verificamos a magnitude em p_{max} e assim calculamos as características de ganho e defasagem do filtro.

Expressando matematicamente, podemos considerar a transformada de Fourier discreta definida como:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn} = DFT(x[n], N) \qquad (7.9)$$

e considerando que o sinal de entrada e saída na freqüência f é dado, respectivamente, por $x_f[n]$ e $y_f[n]$. Ao encontrarmos p_{max} da entrada do sinal, temos que o ganho do filtro é dado por:

$$G_{dB} = 20\, log_{10}\left(\frac{|Y_f[p_{max}]|}{|X_f[p_{max}]|}\right) \qquad (7.10)$$

e podemos calcular a defasagem do filtro como:

$$fase_{rad} = \angle\, (Y_f[p_{max}]) - \angle\, (X_f[p_{max}]) \qquad (7.11)$$

A Figura 7.6 mostra graficamente as regiões da DFT dos sinais de entrada e saída e seus valores máximos.

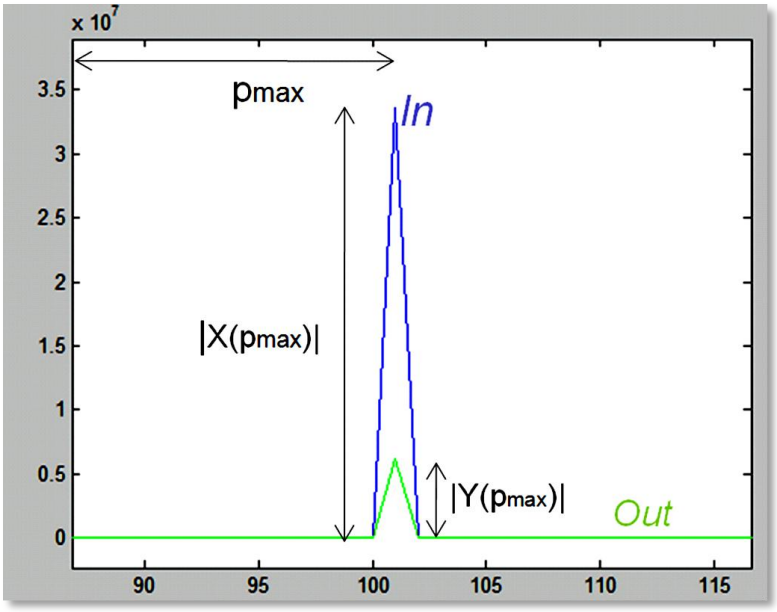
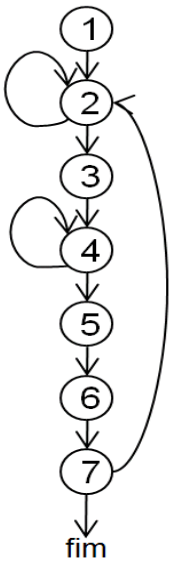


Figura 7.6 – Exemplo de gráfico do método FFT.

Assim o algoritmo que implementa o Método da FFT foi desenvolvido e tem seus passos que podem ser definidos como:



1) Insere-se o arquivo texto referente à entrada e saída do filtro. Realiza-se a separação das freqüências.
2) Descartam-se as primeiras K amostras.
3) Calcula-se a FFT de 2048 pontos do sinal de entrada.
4) Busca-se nesta FFT a posição da amostra que possui valor máximo na entrada. Chamamos de p_{max}.
5) Calcula-se a FFT de 2048 pontos do sinal de saída.
6) Calculam-se os valores de ganho e defasagem do filtro utilizando as Equações (7.10) e (7.11).
7) Repete para todas as componentes senoidais, se não houver mais, fim.

É importante lembrar que este método não utiliza a interpolação. Como a análise é feita na freqüência, a DFT utiliza todas as amostras do sinal para encontrar o valor de magnitude e fase. No caso da análise no tempo, nos restringimos ao primeiro valor de pico após o transiente. Dessa forma, como será verificado na próxima seção, o método FFT se mostrou mais preciso do que o método do Pico.

7.5 Resultados da validação

Nesta seção iremos apresentar os resultados da resposta em freqüência quando aplicamos um sinal tipo *chirp*, descrito na Seção 7.2, na entrada dos filtros implementados no FPGA e interpretamos os sinais de saída através dos métodos descritos nas Seções 7.3 e 7.4.

7.5.1 Validação do Projeto FIR convencional

Para verificar o correto funcionamento da metodologia de validação e das implementações no FPGA, iremos realizar inicialmente uma comparação das respostas em freqüência do filtro sem a otimização (projeto convencional do FIR) sem decimação, com o filtro do projeto convencional com decimador de fator 16 e com a resposta teórica simulada no Matlab. A Fig. 7.7 mostra os gráficos de magnitude e fase dos filtros e o *script* desta validação encontra-se no Apêndice A12.

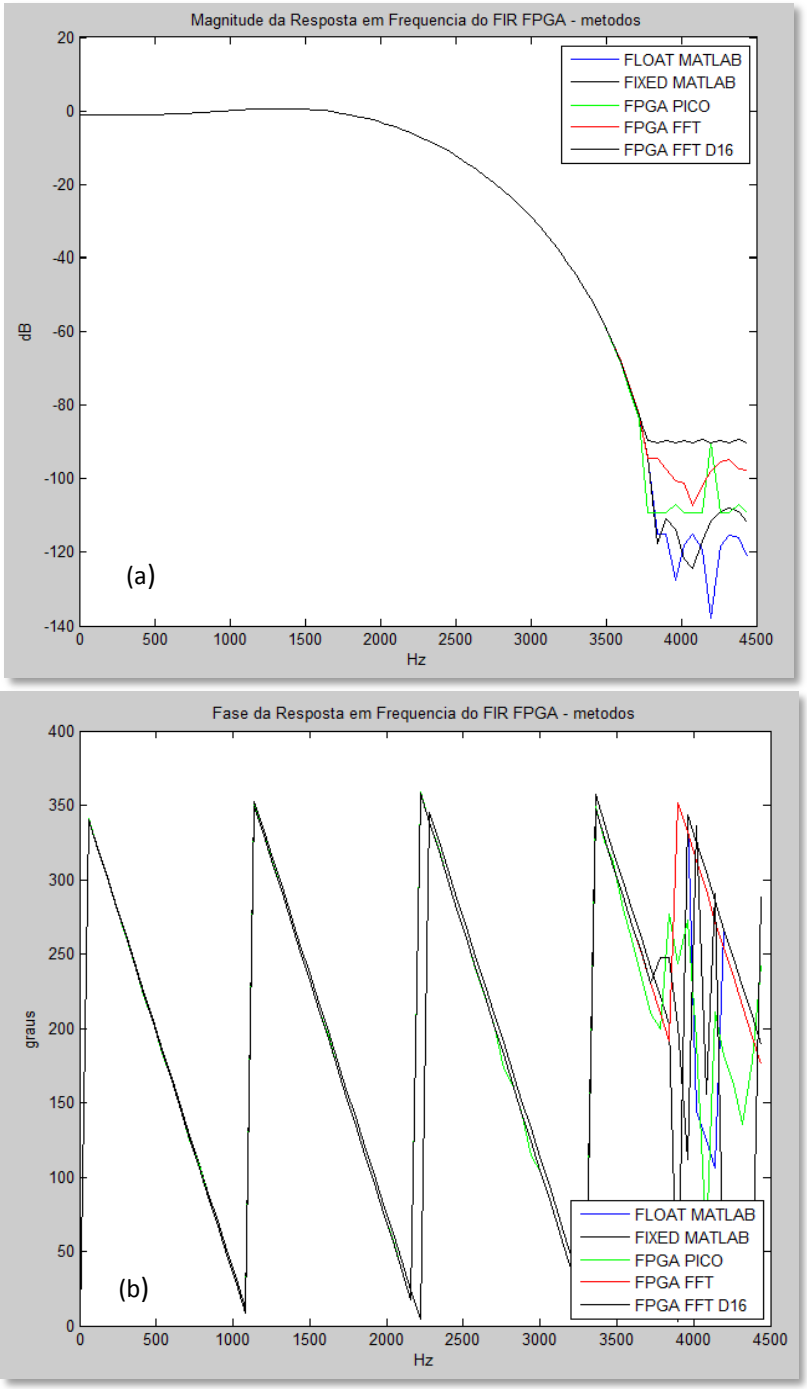


Figura 7.7 – Resposta de (a) magnitude e (b) de fase para os filtros: teórico e prático.

Ao observarmos as respostas em freqüência dos filtros, notamos que a diferença entre o filtro FIR teórico e o filtro FIR do projeto convencional é muito pequena. Além disso, estes gráficos também apresentam a resposta do filtro FIR do projeto convencional implementado no FPGA com um *down-sampler* de fator 16 na saída. Esta análise é importante, pois permite verificar o funcionamento do decimador e concluir que ele somente difere do filtro sem o decimador nas freqüências da faixa de rejeição, mas que mantêm uma atenuação de no mínimo 95 dB.

Concluímos então que o filtro FIR do projeto convencional implementado no FPGA funcionou conforme o esperado. As Fig. 7.8 (a) e Fig. 7.8 (b) apresentam os erros de magnitude e fase, respectivamente, quando comparamos os resultados práticos sem decimação com o teórico calculado pelo Matlab.

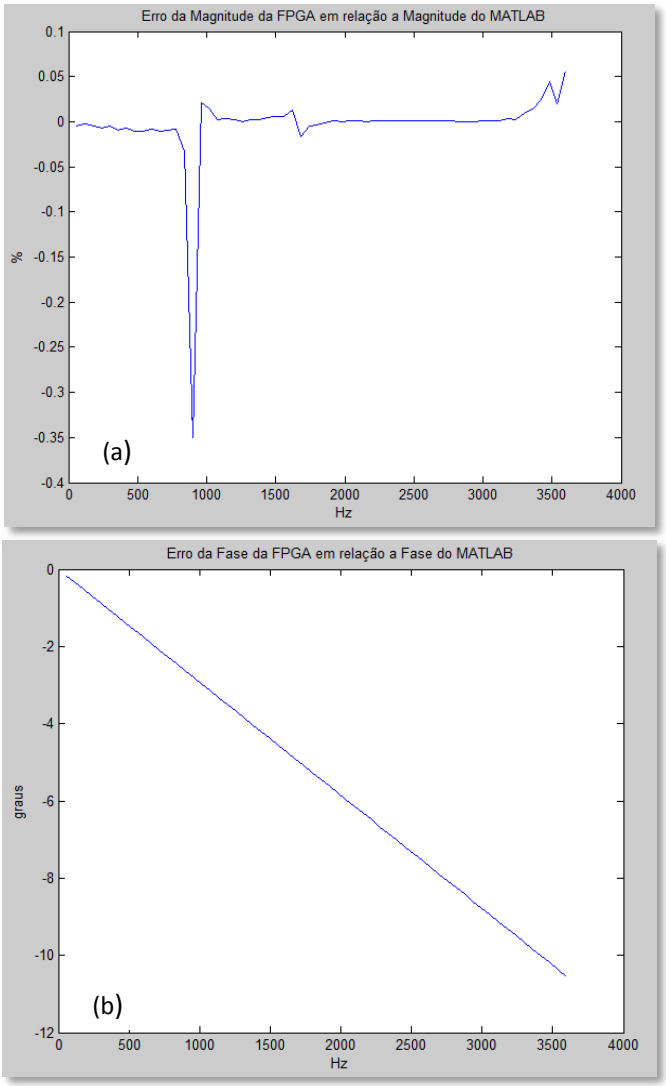


Figura 7.8 – Erros de magnitude (a) e fase (b) para os filtros: teórico e prático.

Observamos então, que o erro referente à magnitude da resposta em freqüência entre o valor teórico e prático é muito pequeno, sendo que na banda de interesse, em que se encontram os harmônicos (de 0 a 300 Hz), este erro ficou próximo a $\pm\ 0{,}01\%$.

Para o erro na resposta de fase, verificamos que a diferença em graus entre os filtros aumenta ao longo da freqüência de forma linear. Isto ocorre porque, como os filtros são digitais e são implementados em hardware diferentes, o tempo de processamento entre eles é diferente e assim o atraso da resposta em relação à saída também será diferente. Assim, a diferença entre os atrasos se mantém constante e ao longo da freqüência mostra-se como um aumento do erro de forma linear.

7.5.2 Validação do Projeto do FIR Otimizado

Para validar o funcionamento do filtro otimizado implementado com os decimadores de 2, primeiro utilizamos a Equação (5.16) para simular matematicamente no Matlab o filtro FIR teórico otimizado e traçar o gráfico de resposta em freqüência, como mostram as Fig. 7.9. Em seguida, utilizamos a metodologia desenvolvida neste capítulo para extrair a resposta em freqüência do mesmo filtro implementado no FPGA, que é mostrada também na Fig. 7.9. O *script* desta validação pode ser consultado no Apêndice A12.

Ao compararmos os gráficos de magnitude da resposta em freqüência teórica e prática da Fig. 7.9, vemos que a diferença entre as respostas só é significativa para freqüências acima de aproximadamente 3500 Hz. Como estas freqüências estão muito acima da banda de interesse (0 a 300 Hz) e ainda sofrem uma atenuação maior que 100 dB, podemos assumir que estes filtros são equivalentes. A Fig. 7.10(a) apresenta um gráfico correspondente ao erro entre as respostas de magnitude do FIR otimizado prático e teórico.

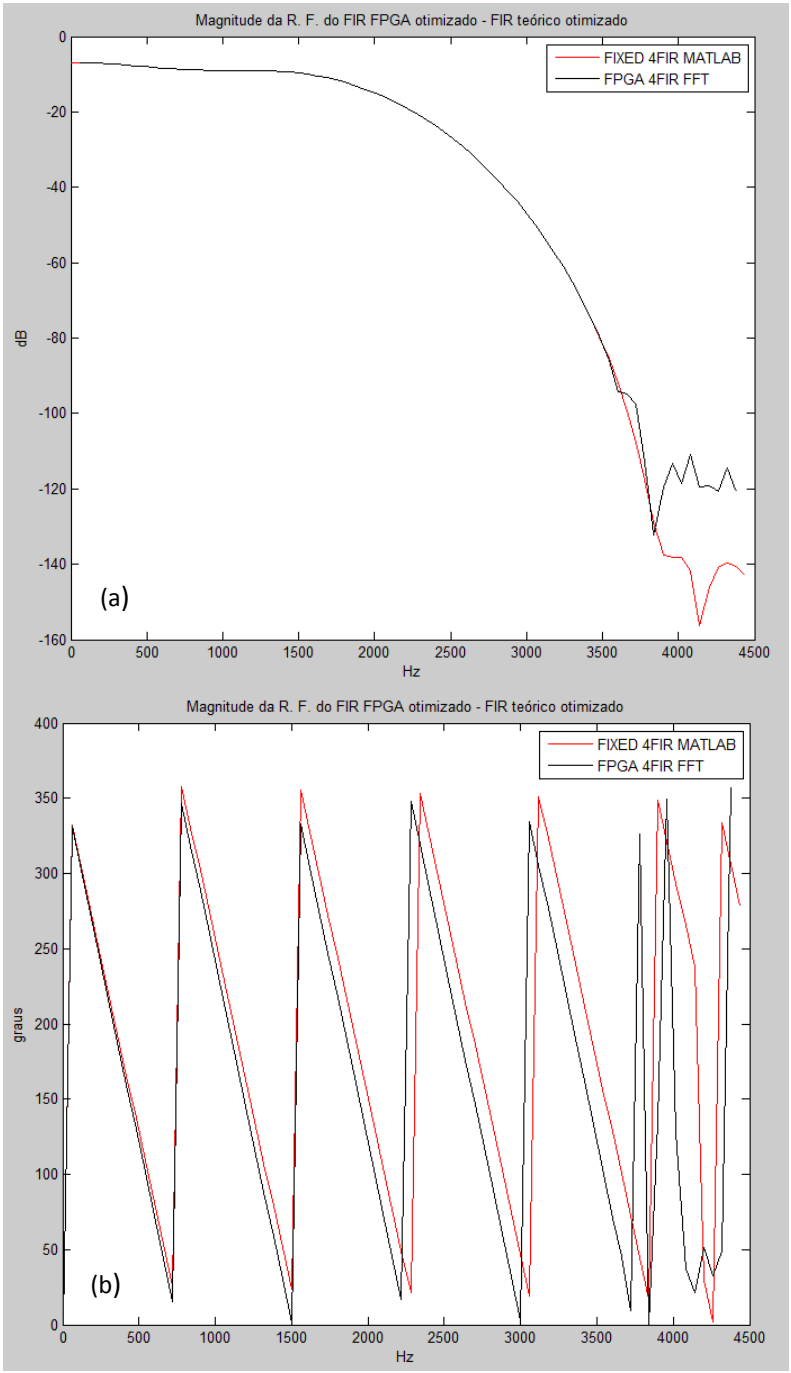


Figura 7.9 – Resposta de magnitude (a) e fase (b) para os filtros otimizados: teórico e prático.

Ao compararmos as fases da resposta em freqüência dos filtros, notamos que existe uma diferença entre elas e que esta aumenta ao longo da freqüência. Novamente, isto ocorre porque os filtros são processados em *hardwares* diferentes e com isso uma diferença no tempo de resposta é observada entre os dois filtros. A Fig. 7.10(b) mostra o gráfico do erro de fase entre as respostas dos dois filtros.

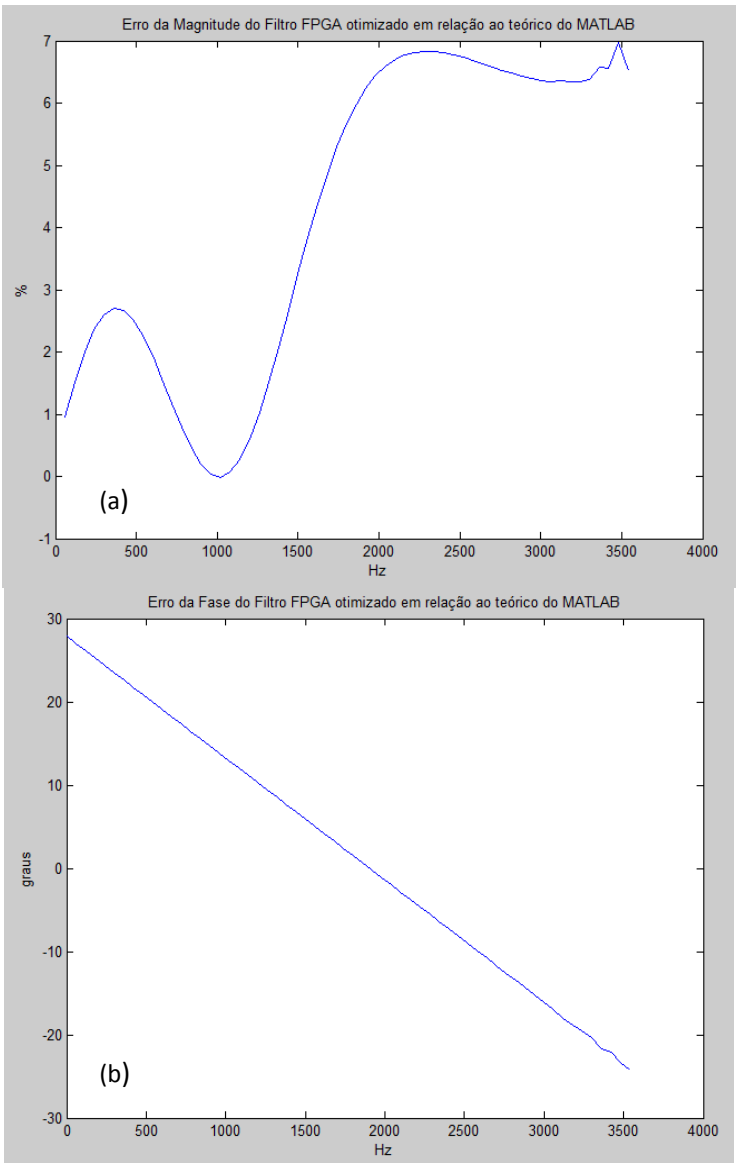


Figura 7.10 – Erros de magnitude (a) e fase (b) para os filtros otimizados: teórico e prático.

Ao analisarmos o gráfico do erro relativo entre as respostas de magnitude dos filtros (Fig. 7.10(a)), percebemos que existe uma diferença de até aproximadamente 3% entre os filtros na banda de interesse. Como estudado no Capítulo 5, os filtros digitais possuem uma variação desprezível no processo de repetição e, assim, podemos assumir que esta resposta será idêntica para todas as unidades de aquisição onde ele é implementado. Como o objetivo final é realizar um somatório das correntes digitalizadas, no caso de todas elas sofrerem a mesma distorção de magnitude, isto não será um problema, pois não irá alterar o valor final do somatório.

Para o gráfico de erro de fase da Fig. 7.10(b), novamente verificamos que o erro em graus aumenta ao longo da freqüência de forma linear devido à diferença no tempo de processamento entre os dois filtros. Como mencionado anteriormente, a variação entre filtros digitais iguais implementados no mesmo hardware é desprezível. Portanto assumimos que a distorção de fase provocada por este filtro, em todas as unidades de aquisição de sinal, é exatamente a mesma e, assim, este projeto mostra-se suficiente para o Sistema SCADA Harmônico.

7.5.3 Ganho de velocidade

A vantagem de se implementar o projeto do FIR otimizado, em relação ao projeto convencional, está no ganho de velocidade que é alcançado quando aquele é utilizado. Um ganho de desempenho é importante, pois permite que um *hardware* menos potente possa realizar a mesma tarefa no mesmo período de tempo e assim o custo do projeto diminui. Além disso, o filtro FIR otimizado torna-se mais eficiente, consumindo menos recursos energéticos.

Para aferir o tempo gasto no processamento dos filtros, utilizamos o ambiente de emulação da Xilinx durante a execução do código *testbench*, descrito na Seção 6.3. Este ambiente permite ser configurado para medir o tempo gasto pelo hardware do FPGA ao processar um conjunto de estímulos de entrada, formado pelo sinal *chirp* da Seção 7.2, e produzir todas as saídas. O custo de processamento é proporcional ao número de pulsos de *clock* gastos para processar todas as amostras. A Fig. 6.4 mostra a interface do ambiente Isim em que é possível observar os pulsos de *clock* e medir o tempo de execução do hardware. A Tabela 7.1 mostra os tempos medidos.

Filtro FIR (clock = 100 MHz)	**Sinal *Chirp***	**Uma amostra de entrada**
Projeto Convencional	393,84 ms	2,22 us
Projeto Otimizado	44,35 ms	0,25 us
Ganho de Velocidade	8,9	8,9
Ganho de Complexidade	4,9	4,9

Tabela 7.1 – Comparação de desempenho para os projetos de filtro convencional e otimizado.

A Tabela 7.1 mostra os tempos gastos pelos filtros implementados no FPGA no processamento de uma amostra e no processamento de um conjunto de amostras, formado pelo sinal *chirp*. Verificamos que o ganho de velocidade, como era esperado, se mostrou o mesmo para ambos os casos.

Ao compararmos este resultado com o ganho de complexidade, calculado no final do Capítulo 5, verificamos que o ganho de desempenho obtido empiricamente se mostrou maior do que o valor teórico. Como mencionado no final do Capítulo 5, o cálculo teórico não leva em consideração as diferenças de hardware dos filtros. Como visto no Capítulo 6, os filtros utilizam memórias do tipo *block RAM* para armazenar tanto as amostras passadas como os coeficientes do filtro. Assim, no funcionamento real dos filtros, o tempo gasto no processamento das amostras deve levar em consideração o tempo de acesso à memória. Como no projeto convencional o filtro possui uma ordem maior e uma taxa de amostragem maior, mais acessos à memória são realizados e com isso o ganho de velocidade se torna maior que o ganho de complexidade.

7.5.4 Considerações finais

Neste capítulo realizamos uma metodologia para validar o funcionamento dos filtros implementados no FPGA e comparamos os resultados obtidos com os seus respectivos valores teóricos encontrados durante a simulação do Matlab.

Através da análise dos resultados obtidos nos experimentos, podemos concluir que as diferenças entre os filtros não são significativas e todos podem servir como filtro *anti-aliasing* digital para as unidades de aquisição do Sistema SCADA Harmônico, atendendo as especificações, tanto para a resposta de magnitude quanto para resposta de fase. Além disso, concluímos que o filtro FIR otimizado com multi-estágios e decimadores se mostrou ser um circuito mais eficiente, podendo ser implementado em um hardware mais lento, e conseqüentemente, reduzir os custos de produção. Portanto este filtro será utilizado na versão final das UASs do Sistema SCADA.

Capítulo 8 Conclusões

Neste trabalho, vimos que os harmônicos não-característicos estão causando problemas de sobrecarga nos filtros da subestação de Ibiúna, que é responsável por converter a energia, que vem de Itaipu em corrente-contínua, para corrente-alternada.

Para resolver este problema, um sistema de monitoramento em tempo real das componentes harmônicas das correntes que trafegam em diferentes pontos da subestação, Sistema SCADA Harmônico, está sendo desenvolvido no laboratório LEMT da COPPE/UFRJ. Mostramos que para satisfazer as necessidades deste sistema, o filtro *anti-aliasing* das unidades digitalizadoras deve ser projetado cuidadosamente para cumprir especificações exigentes e que a solução requer uma combinação de filtros analógico e digital.

Como o sistema de monitoramento funciona de forma sincronizada, a distorção de fase das correntes digitalizadas não pode ser maior do que $0{,}1°$. Caso contrário, o ângulo dos fasores das correntes seria alterado, inviabilizando a identificação das fontes geradoras da poluição harmônica.

Para projetar filtro *anti-aliasing* analógico cumprindo esse requisito de resposta de fase, utilizamos o conceito de que uma distorção de fase igual para todos os filtros analógicos construídos não altera o resultado final do somatório das correntes. Podemos, então, entender que a exigência de $0{,}1°$ é na verdade uma especificação de fase relativa entre as unidades de medição.

Concluímos que o filtro *anti-aliasing* analógico com aproximação *Bessel* de quarta ordem possui uma resposta em freqüência suficiente para satisfazer as especificações de projeto quando utilizado em conjunto com o filtro digital. Além disso, uma análise estatística da variação dos componentes que formam o circuito eletrônico mostrou que uma precisão mínima de 0,1% e 0,5% para os resistores e os capacitores, respectivamente, deve ser utilizada para garantir que o erro de fase relativo esteja dentro das especificações.

Vimos também que os filtros digitais não possuem o problema de variação na resposta de fase quando repetidos e que foram projetados para utilizarem o *hardware* do FPGA. A decimação mostrou-se importante por permitir a redução da taxa de amostragem impedindo que um excesso de dados fosse inserido na rede formada pelas unidades de aquisição. Quando realizado um processo de otimização, baseado em multi-estágios e decimações, o filtro FIR mostrou, empiricamente, uma resposta nove vezes mais rápida do que o filtro FIR realizado com o projeto convencional. Isto permite a utilização de um *hardware* digital menos potente e, assim, uma realização do filtro mais eficiente.

Além disso, uma metodologia foi desenvolvida para validar o funcionamento dos filtros digitais e verificamos que a resposta em freqüência destes filtros corresponde ao esperado, sendo suficiente para o funcionamento correto do Sistema SCADA Harmônico.

Em suma, concluímos que este trabalho é importante, pois viabiliza a construção dos filtros *anti-aliasing* das unidades de aquisição quando os requisitos especificações devido ao sincronismo das aquisições e ao processamento dos dados são bastante exigentes. Também, esta solução contribui para a implantação do sistema que irá monitorar as componentes harmônicas de corrente que causam a sobrecarga nos filtros da SE de Ibiúna. Com o sucesso no funcionamento deste sistema, poderemos melhorar a qualidade de energia e evitar que novos filtros sejam construídos nesta subestação.

8.1 Trabalhos Futuros

Como foi verificado no projeto do filtro digital, os quatro sub-filtros que implementam o filtro FIR otimizado com multi-estágios são iguais. Com isso, uma arquitetura que permita reutilizar apenas um estágio do filtro poderia ser implementada para otimizar a área de circuito dentro do FPGA. Assim, os dados passariam no mesmo filtro com decimação de fator 2 quatro vezes, ocupando uma área menor e tornando o filtro FIR ainda mais eficiente.

Além disso, uma estrutura do tipo *polyphase* [32] combinada com a decimação poderia ser utilizada para reduzir a complexidade dos quatro estágios do filtro FIR otimizado [33]. O módulo *FIR Compiler* [28] do ambiente da Xilinx poderia ser estudado com mais profundidade para formar esta topologia no *hardware* do FPGA. O filtro resultante realizaria, então, a filtragem em um tempo menor tornando-se ainda mais eficiente.

Referências Bibliográficas

1. WIKIPEDIA, *Itaipu binacional* [online]. Disponível em: <http://pt.wikipedia.org/wiki/Usina_Hidrel%C3%A9trica_de_Itaipu>. [Capturado em: 10 de fevereiro de 2011].

2. AREDES, M., IASBECK S. L. G., NDIAYE, M. S., FERNANDES, D. M., EMMERIK, E. V. "SCADA Harmônico para Análise da Propagação Harmônica em Sistemas de Potência". In: *XX Seminário Nacional de Produção e Transmissão de Energia Elétrica*,Recife-PE,2009

3. BOSE, B. K., "Voltage-Fed Converters". *Modern Power Eletronics,* chapter 5, NJ, USA, Prentice-Hall, 2002.

4. DESOER, C. A., KUH E. S., "Lumped Circuits and Kirchhoff Laws". *Basic Circuit Theory*, International Student Edition, Chapter 1, pp. 5, Tokyo, Japan, McGraw-Hill, 1969.

5. MITRA, S. K., "Digital Processing of Continuous-Time Signals". *Digital Signal Processing*, 3rd ed, chapter 4, pp.172, New York, USA, McGraw-Hill, 2006.

6. ANALOG DEVICES, *AD7687 Datasheet* [online], Disponível em: <http://www.analog.com/static/imported-files/data_sheets/AD7687.pdf>.[capturado em: 1 de março de 2011].

7. PETRAGLIA, M. R.; PETRAGLIA, A.,"Amostragem", *Notas de Aula de Filtros Digitais*, chaper 6, Rio de Janeiro, Brazil, UFRJ, 2009.

8. MITRA, S. K., "Digital Processing of Continuous-Time Signals". *Digital Signal Processing*, 3rd ed, chapter 4, pp.176-177, New York, USA, McGraw-Hill, 2006.

9. FILHO, S. N., "Aproximações". *Filtros Seletores de Sinais*, 2ª edição, chapter 2, pp. 40, SC, Brasil, Editora UFSC, 2003.

10. FILHO, S. N., "Aproximações". *Filtros Seletores de Sinais*, 2ª edição, chapter 2, pp. 45, SC, Brasil, Editora UFSC, 2003.

11. FILHO, S. N., "Aproximações". *Filtros Seletores de Sinais*, 2ª edição, chapter 2, pp. 57, SC, Brasil, Editora UFSC, 2003.

12. FILHO, S. N., "Aproximações". *Filtros Seletores de Sinais*, 2ª edição, chapter 2, pp. 65, SC, Brasil, Editora UFSC, 2003.

13. MATHWORKS. *Besself* [online], Disponível em: <http://www.mathworks.com/help/toolbox/signal/besself.html>. [capturado em: 22 de fevereiro de 2011].

14. TEXAS INSTRUMENTS, *Texas Instruments* [online], Disponível em: <http: //www.ti.com/> . [capturado em: 10 de fevereiro de 2011].

15. WIKIPEDIA. *Spice* [online], Disponível em: <http://en.wikipedia.org/wiki/SPICE>. [capturado em: 20 de fevereiro 2011].

16. DESOER, C. A., KUH E. S., “Thévenon and Norton Equivalent Circuits”. *Basic Circuit Theory*, International Student Edition, Chapter 1, pp. 27, Tokyo, Japan, McGraw-Hill, 1969.

17. WIKIPEDIA. *Monte Carlo Method* [online], Disponível em: <http://en.wikipedia.org/wiki/Monte_Carlo_method>. [capturado em: 25 de fevereiro 2011].

18. WIKIPEDIA. *Digital Filter* [online], Disponível em: <http://en.wikipedia.org/wiki/Digital_filter>. [capturado em: 25 de fevereiro 2011].

19. MITRA, S. K., "Digital Filter Structures". *Digital Signal Processing*, 3rd ed, chapter 8, pp.432, New York, USA, McGraw-Hill, 2006.

20. MITRA, S. K., "FIR Digital Filter Design". *Digital Signal Processing*, 3rd ed, chapter 10, pp.523, New York, USA, McGraw-Hill, 2006.

21. MATHWORKS. *Fdatool* [online], Disponível em: <http://www.mathworks.com/help/toolbox/signal/fdatool.html>. [capturado em: 22 de fevereiro de 2011].

22. MITRA, S. K., "Multirate System". *Digital Signal Processing*, 3rd ed, chapter 13, pp.739, New York, USA, McGraw-Hill, 2006.

23. MITRA, S. K., "Multirate System". *Digital Signal Processing*, 3rd ed, chapter 13, pp.743, New York, USA, McGraw-Hill, 2006.

24. MITRA, S. K., "Multirate System". *Digital Signal Processing*, 3rd ed, chapter 13, pp.746, New York, USA, McGraw-Hill, 2006.

25. WIKIPEDIA. *Field-programmable gate array* [online], Disponível em: <http://en.wikipedia.org/wiki/Field-programmable_gate_array#cite_note-FPGA-1>. [capturado em: 25 de fevereiro 2011].

26. XILINX, *Spartan-3AN datasheet* [online], Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf>.[capturado em: 25 de fevereiro 2011].

27. XILINX, *ISE Design Suite: Logic Edition* [online], Disponível em: <http://www.xilinx.com/tools/logic.htm>. [capturado em: 25 de fevereiro 2011].

28. XILINX. *LogiCore IP FIR Compiler Ver 5.0* [online], Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/fir_compiler_ds534.pdf>.

[capturado em: 25 de fevereiro 2011].

29. D'AMORE, R., "Introdução". *VHDL - Descrição e Síntese de Circuitos Digitais,* 1ª edição, chapter 1, RJ, Brasil, LTC Editora, 2005.

30. XILINX, *Isim User Guide* [online], Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/plugin_ism.pdf>. [capturado em: 1 de março 2011].

31. MATLAB, *Interp* [online], Disponível em: <http://www.mathworks.com/help/toolbox/signal/interp.html>. [capturado em: 1 de março 2011].

32. MITRA, S. K., "The Polyphase Decomposition". *Digital Signal Processing*, 3rd ed, chapter 13.4, pp.761, New York, USA, McGraw-Hill, 2006.

33. MITRA, S. K., "Computationally Eficient Interpolator and Decimator Structures". *Digital Signal Processing*, 3rd ed, chapter 13.4, pp.764, New York, USA, McGraw-Hill, 2006.

Apêndice

A1 – Formas de onda do conversor 12 pulsos.

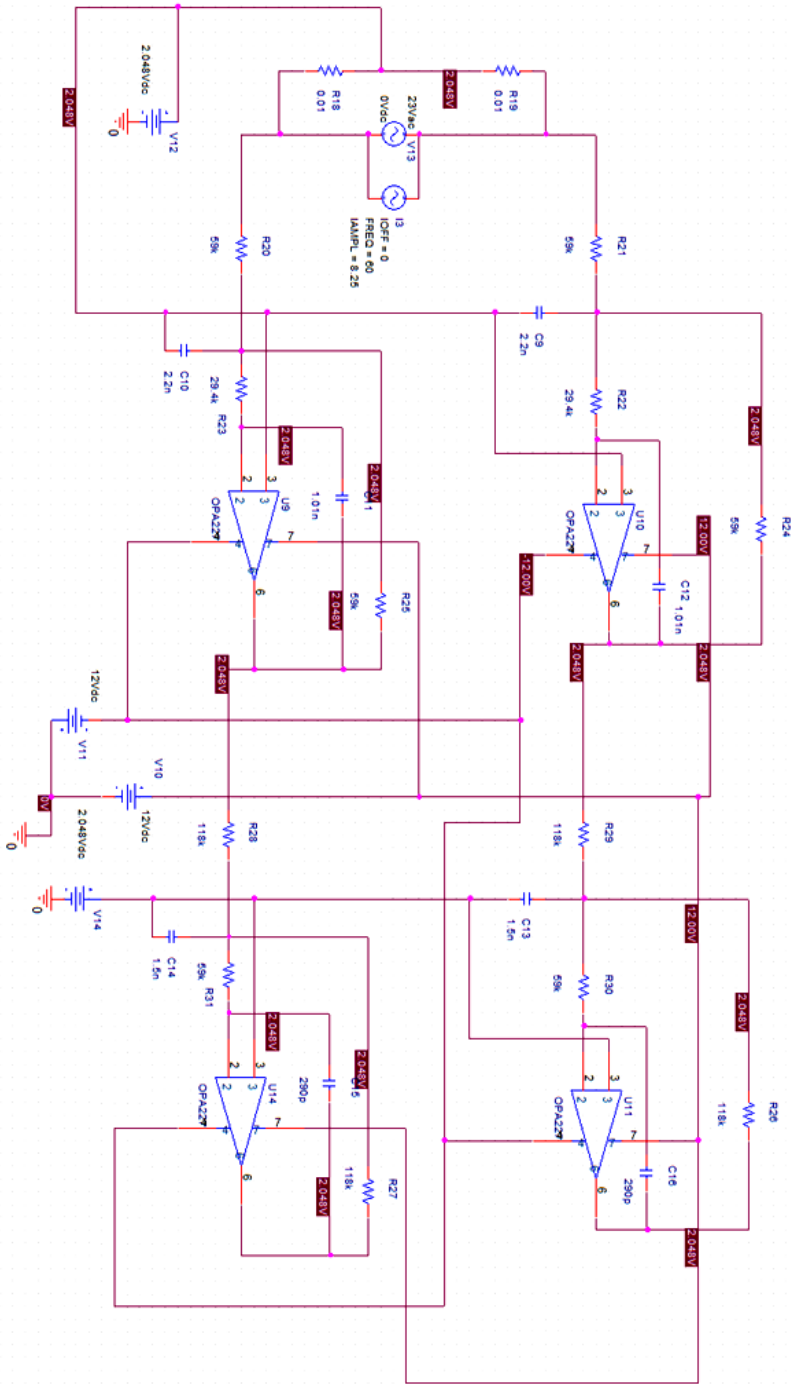
```
%FORMAS DE ONDA NA SAÍDA DO CONVERSOR DE CORREENTE DE 12 PULSOS----------
close all;
%Conversor de 12 pulsos para até 31o harmônico
H=101;
n = [1:1000]; %vetor de pontos, escala de tempo.
Fs = 20000;   %frequencia de amostragem
t = n./Fs; %ietor de tempo
Fc = 60;      %frequencia da rede
i = .75;      %tensão de pico da rede
wn = 2*pi*n*Fc/Fs;

iab1=0;ide1=0;ief1=0;
ibc1=0;ica1=0;ifd1=0;
for k=1:2:H
    iab1 = iab1 + i*2*sqrt(3)/(k*pi)*sin(k*wn);
    ibc1 = ibc1 + i*2*sqrt(3)/(k*pi)*sin(k*(wn-2/3*pi));
    ica1 = ica1 + i*2*sqrt(3)/(k*pi)*sin(k*(wn+2/3*pi));

    ide1 = ide1 + i*2/(k*pi)*sin(k*(wn-pi/6));
    ief1 = ief1 + i*2/(k*pi)*sin(k*(wn-pi*5/6));
    ifd1 = ifd1 + i*2/(k*pi)*sin(k*(wn+pi/2));
end

%iz =
(cos(wn)+1/11*cos(11*wn)+1/13*cos(13*wn)+1/23*cos(23*wn)+1/25*cos(25*wn));
ia0 = iab1 + ide1 - ief1;
ib0 = ibc1 + ief1 - ifd1;
ic0 = ica1 + ifd1 - ide1;
in0 = 1/3*(ia0+ib0+ic0);
ian = ia0 - in0;
ibn = ib0 - in0;
icn = ic0 - in0;
iab = ia0-ib0;
ibc = ib0-ic0;
ica = ic0-ia0;


figure;
subplot(3,1,1);
plot(t,ia0);
title(['Formas de onda das correntes do conversor 12 pulsos']);
subplot(3,1,2);
xlabel('Segundos');
ylabel('Ampere');
plot(t,ib0,'red');
subplot(3,1,3);
xlabel('Segundos');
ylabel('Ampere');
plot(t,ic0,'green');
xlabel('Segundos');
ylabel('Ampere');

figure;
subplot(3,1,1);
plot(t,ian,'black');
xlabel('Segundos');
```

```
ylabel('Ampere');
title(['Formas de onda das correntes do conversor 12 pulsos']);
subplot(3,1,2);
plot(t,ibn,'black');
xlabel('Segundos');
ylabel('Ampere');
subplot(3,1,3);
plot(t,icn,'black');
xlabel('Segundos');
ylabel('Ampere');

figure;
subplot(3,1,1);
plot(t,iab);
xlabel('Segundos');
ylabel('Ampere');
title(['Formas de onda das correntes do conversor 12 pulsos']);
subplot(3,1,2);
plot(t,ica,'red');
xlabel('Segundos');
ylabel('Ampere');
subplot(3,1,3);
plot(t,ibc,'green');
xlabel('Segundos');
ylabel('Ampere');
```

A2 – *Script* da resposta em freqüência dos filtros analógicos.

```
%Filtro Bessel, ORDEM = 25, Fc = 1800Hz, ou ORDEM = 4
%[b,a] = besself(4,2700*2*pi);
[b,a] = besself(25,1800*2*pi);
w = (100:100:100000)*2*pi;

h = freqs(b,a,w);
mag = abs(h);
phase = angle(h);

f = w/(2*pi);
mag = 20*log10(mag);
phase = phase*180/pi;

subplot(2,1,1), semilogx(f,mag);
ylabel('Magnitude (dB)');
grid on;xlabel('frequency (Hz)');
subplot(2,1,2), semilogx(f,phase);
ylabel('Phase (degrees)');
grid on;xlabel('frequency (Hz)');
```

A3 – Esquemático do Filtro *anti-aliasing* analógico.



A4 – *Script* do Erro na Resposta de fase do filtro analógico Spice x Matlab.

```
close all;
clear all;
resolution = 10000;

%1o estagio
R1 = 59E3;
R2 = 29.5E3;
R3 = 59E3;
C1 = 1.01E-9;
C2 = 2.2E-9;

b0 = -R3;
a2 = R1*R2*R3*C1*C2;
a1 = C1*( R2*R3 + R2*R1 + R1*R3 );
a0 = R1;

%2o estagio
R4 = 118E3;
R5 = 59E3;
R6 = 118E3;
C3 = 290E-12;
C4 = 1.5E-9;

b3 = -R6;
a5 = R4*R5*R6*C3*C4;
a4 = C3*( R5*R6 + R5*R4 + R4*R6 );
a3 = R4;

B0 = b0*b3;
A0 = a0*a3;
A1 = a0*a4 + a1*a3;
A2 = a0*a5 + a2*a3 + a1*a4;
A3 = a2*a4 + a1*a5;
A4 = a2*a5;

b = [B0];
a = [A4 A3 A2 A1 A0];
w =  logspace(3,6,resolution);

h = freqs(b,a,w);
mag = abs(h);
phase = angle(h);

f = w/(2*pi);
mag = 20*log10(mag);
phase = phase*180/pi;

title('Resposta em frequencia filtro analogico');
subplot(2,1,1), semilogx(f,mag)
xlabel('Hz');
ylabel('dB');
subplot(2,1,2), semilogx(f,phase)
xlabel('Hz');
ylabel('graus');

x = load('spice_fase.mat');

fx = x.p(:,1); %cria vetor com as frequencias do spice
wx = fx*2*pi;
hx = freqs(b,a,wx);
phase_x = angle(hx);
phase_x = phase_x*180/pi;

figure;
semilogx(fx,phase_x);
figure;
```

```
for i=1:1:size(x.p,1)
    p_err(i) = x.p(i,2)-phase_x(i);
    if(p_err(i) <= -350)
        p_err(i) = p_err(i) + 360;
    end
end

plot(fx(1:60),p_err(1:60));
title('Erro da resposta de fase Matlab x Spice');
xlabel('Hz');
ylabel('graus');
```

A5 – *Script* do histograma da variação de fase do filtro analógico.

```
close all;
clear all;
resolution = 10000;

%1o estagio
R1 = 59E3;
R2 = 29.5E3;
R3 = 59E3;
C1 = 1.01E-9;
C2 = 2.2E-9;

b0 = -R3;
a2 = R1*R2*R3*C1*C2;
a1 = C1*( R2*R3 + R2*R1 + R1*R3 );
a0 = R1;

%2o estagio
R4 = 118E3;
R5 = 59E3;
R6 = 118E3;
C3 = 290E-12;
C4 = 1.5E-9;

b3 = -R6;
a5 = R4*R5*R6*C3*C4;
a4 = C3*( R5*R6 + R5*R4 + R4*R6 );
a3 = R4;

B0 = b0*b3;
A0 = a0*a3;
A1 = a0*a4 + a1*a3;
A2 = a0*a5 + a2*a3 + a1*a4;
A3 = a2*a4 + a1*a5;
A4 = a2*a5;

b = [B0];
a = [A4 A3 A2 A1 A0];
w =  logspace(3,6,resolution);

h = freqs(b,a,w);
mag = abs(h);
phase = angle(h);

f = w/(2*pi);
mag = 20*log10(mag);
phase = phase*180/pi;

title('Resposta em frequencia filtro analogico');
subplot(2,1,1), semilogx(f,mag)
xlabel('Hz');
ylabel('dB');
subplot(2,1,2), semilogx(f,phase)
xlabel('Hz');
ylabel('graus');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%monte carlo uniforme
ER = 0.001; %erro dos resistores 1%
EC = 0.005; %erro dos capacitores 2%

iter = 10000;

for i=1:1:iter

%1o estagio
R1 = 59E3*(1+(rand-0.5)*2*ER);
```

```
R2 = 29.5E3*(1+(rand-0.5)*2*ER);
R3 = 59E3*(1+(rand-0.5)*2*ER);
C1 = 1.01E-9*(1+(rand-0.5)*2*EC);
C2 = 2.2E-9*(1+(rand-0.5)*2*EC);

%%% H1(S)
b0 = -R3;
a2 = R1*R2*R3*C1*C2;
a1 = C1*( R2*R3 + R2*R1 + R1*R3 );
a0 = R1;

%2o estagio
R4 = 118E3*(1+(rand-0.5)*2*ER);
R5 = 59E3*(1+(rand-0.5)*2*ER);
R6 = 118E3*(1+(rand-0.5)*2*ER);
C3 = 290E-12*(1+(rand-0.5)*2*EC);
C4 = 1.5E-9*(1+(rand-0.5)*2*EC);

%%% H2(S)
b3 = -R6;
a5 = R4*R5*R6*C3*C4;
a4 = C3*( R5*R6 + R5*R4 + R4*R6 );
a3 = R4;

%%% função de transf final
B0 = b0*b3;
A0 = a0*a3;
A1 = a0*a4 + a1*a3;
A2 = a0*a5 + a2*a3 + a1*a4;
A3 = a2*a4 + a1*a5;
A4 = a2*a5;

%%universo de freqs
b = [B0];
a = [A4 A3 A2 A1 A0];
w =  logspace(2,4,resolution);

h = freqs(b,a,w);
mag = abs(h);
phase = angle(h);

f = w/(2*pi);
mag = 20*log10(mag);
phase = phase*180/pi;

%figure;
%subplot(2,1,1), semilogx(f,mag)
%subplot(2,1,2), semilogx(f,phase)

%para resolução de 10mil pontos
%5o. harmônico tem f = 300.0182437122328 Hz nA amostra n = 6377
%3o. harmônico tem f = 180.02195849887102 Hz nA amostra n = 5268
%1o. harmônico tem f = 60.0174829007999 Hz nA amostra n = 2883
nf5 = 6377;
nf3 = 5268;
nf1 = 2883;
h5(i) = phase(nf5);
h3(i) = phase(nf3);
h1(i) = phase(nf1);

end

r = 0.04;
x = -10*r:r:10*r;

m5 = mean(h5);
h5 = h5- m5;

figure;
[j,k] = hist(h5,x);
title('Histograma 5o. harmonico 10mil amostras');
xlabel('graus');
ylabel('numero de amostras')

m3 = mean(h3);
h3 = h3- m3;
figure;
```

```
hist(h3,x);
title('Histograma 3o. harmonico 10mil amostras');
xlabel('graus');
ylabel('numero de amostras')

m1 = mean(h1);
h1 = h1- m1;
figure;
hist(h1,x);
title('Histograma 1o. harmonico 10mil amostras');
xlabel('graus');
ylabel('numero de amostras')
```

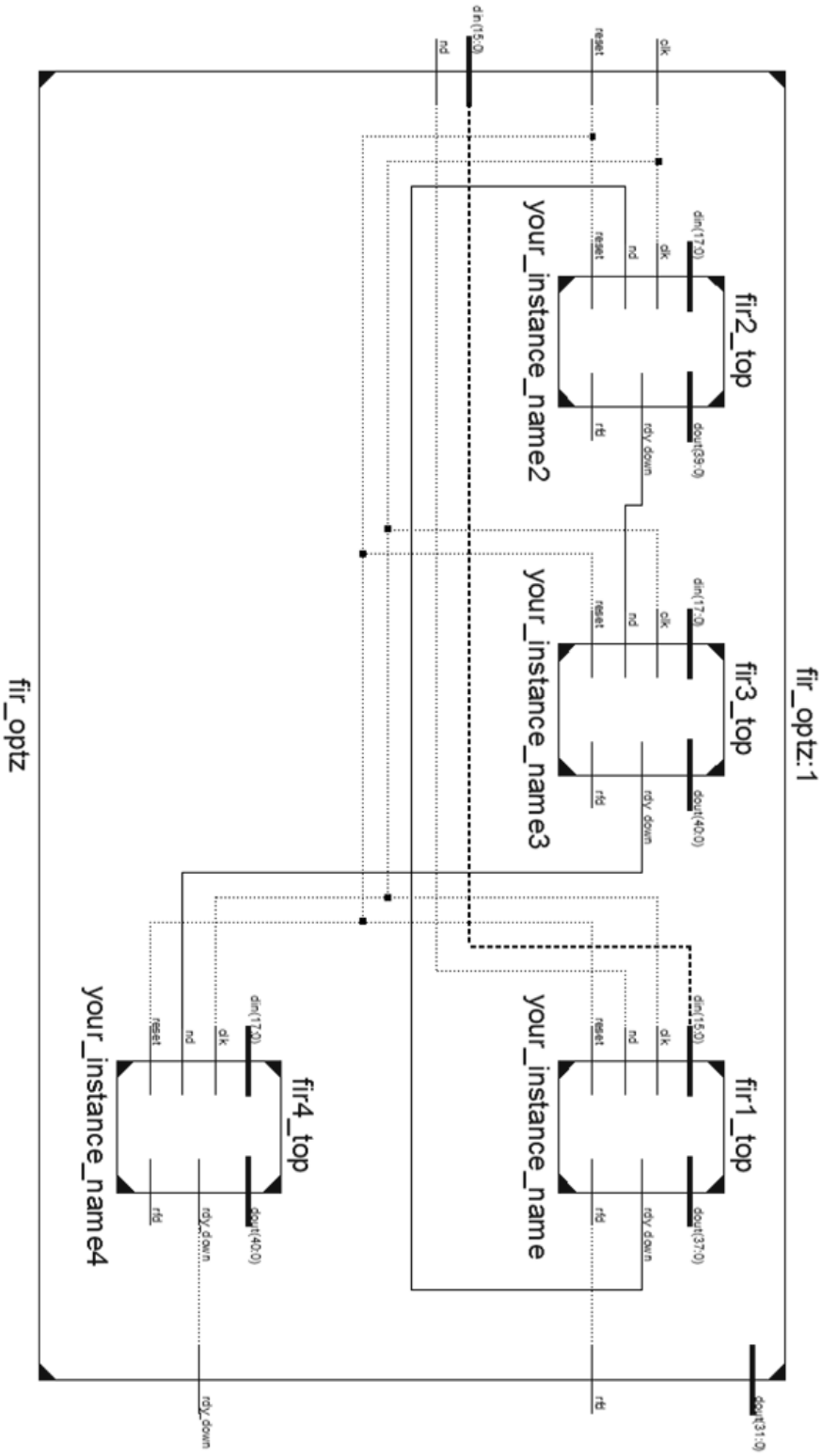
A6 – *Script* Resposta em Freqüência do Filtro para Projeto Convencional x Otimizado

```
%----Calculando a resposta em frequencia Filtro  Simples   ----------

h_ideal = load('fir_coe_float.mat');
[H_ideal,w_ideal] = freqz(h_ideal.float,1,1024);

for i=1:1:size(H_ideal,1)
    fH_ideal(i) = func_rad2deg(angle(H_ideal(i)));
end

%----Calculando a resposta em frequencia resultante dos 4 filtros----------
f3 = w_ideal*Fs/(2*pi);

h0 = load('h1.mat'); %coeficientes do filtro
%h1 = load('h2.mat');

for i=1:size(h0.h,2)
   hr2((i-1)*2+1) = h0.h(i);
   for j=1:1
        hr2((i-1)*2+i+j) = 0;
   end
end

for i=1:size(h0.h,2)
   hr3((i-1)*4+1) = h0.h(i);
   for j=1:3
        hr3((i-1)*4+i+j) = 0;
   end
end

for i=1:size(h0.h,2)
   hr4((i-1)*8+1) = h0.h(i);
   for j=1:7
        hr4((i-1)*8+i+j) = 0;
   end
end


hr_2 = conv(conv(conv(h0.h,hr2),hr3),hr4);
[Hr2,w_r2] = freqz(hr_2,1,1024);

f4 = w_r2*Fs/(2*pi);

for i=1:1:size(Hr2,1)
    fase_Hr(i) = func_rad2deg(angle(Hr2(i)));
end



%----Plotando os valores de resposta em frequencia-------------------------

%magnitude
mx = 75;
plot(f3(1:mx),20*log10(abs(H_ideal(1:mx))),'black');
hold;
plot(f4(1:mx),20*log10(abs(Hr2(1:mx))),'red');
legend('Filtro Simples','Filtro decimações 2',1);
title('Magnitude da Resposta em Frequencia do Simples x Otimizado')
xlabel('Hz')
ylabel('dB')

%fase
figure;
plot(f3(1:mx),fH_ideal(1:mx),'black');
hold;
plot(f4(1:mx),fase_Hr(1:mx),'red');
legend('Filtro Simples','Filtro decimações 2',1);
```

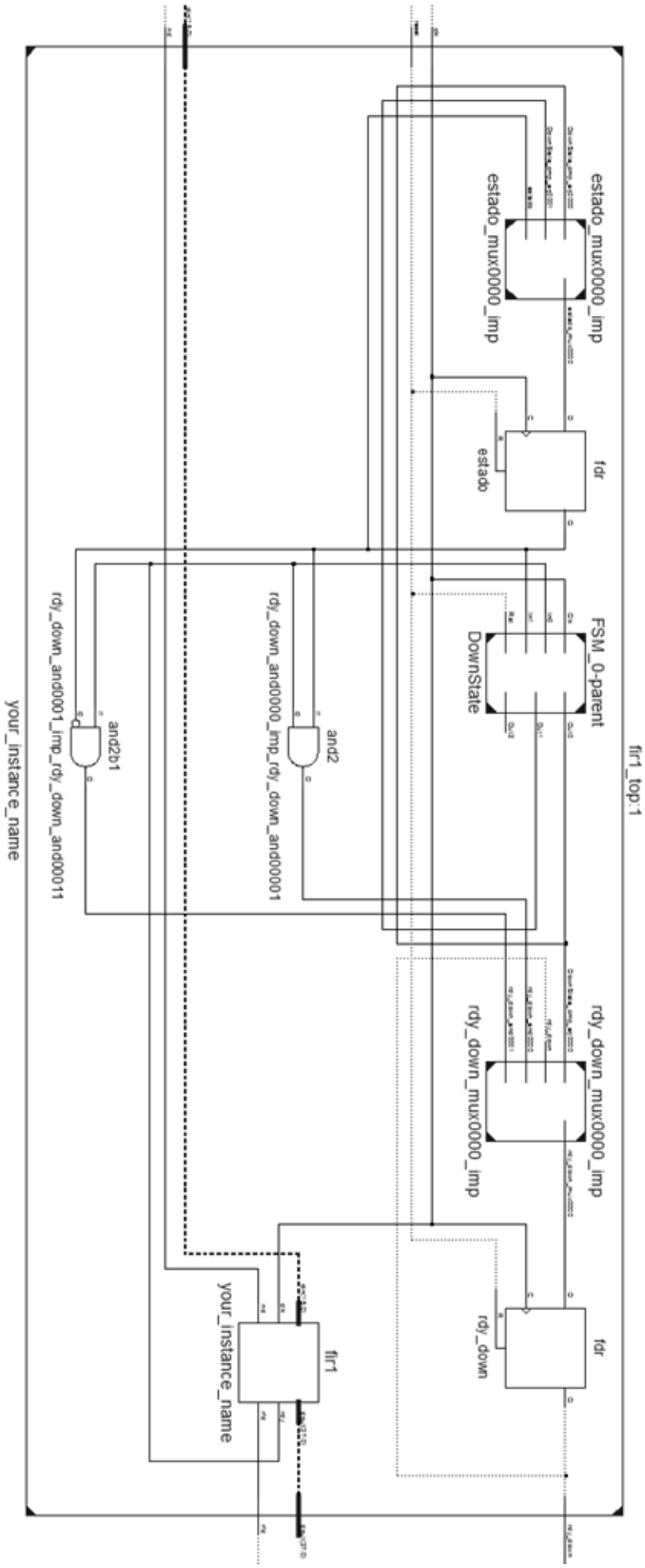
```
title('Magnitude da Resposta em Frequencia do Simples x Otimizado')
xlabel('Hz')
ylabel('graus')
```

A7 –Esquemático do Filtro Digital Otimizado

1) Esquemático do Filtro FIR Otimizado com 4 Filtros



2) Esquemático de um módulo



A8 – *Testbench* escrito em VHDL

```
--------------------------------testebench FIR-------------------------------------------
-----------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
--use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
--use ieee.numeric_bit.all;
--USE ieee.numeric_std.ALL;



ENTITY test_bench_fir IS
END test_bench_fir;

ARCHITECTURE behavior OF test_bench_fir IS


    COMPONENT fir_optz
    Port ( clk : in  STD_LOGIC;
                      reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (15 downto 0);
           dout : out  STD_LOGIC_VECTOR (31 downto 0));
    END COMPONENT;



   --Inputs
   signal clk : std_logic := '0';
   signal nd : std_logic := '0';
   signal din : std_logic_vector(15 downto 0) := (others => '0');
   signal reset : std_logic := '0';

       --Outputs
   signal rfd : std_logic;
   signal rdy : std_logic;
   signal dout : std_logic_vector(31 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

       --aux signals
       signal nd_inicial : std_logic;
       file out_file: TEXT open WRITE_MODE is "out.txt";
       file out_file_D: TEXT open WRITE_MODE is "outD.txt";
       shared variable salto : integer := 0;--1;
       shared variable pos : integer := 1;--511;
       shared variable counter : integer := 0;
       shared variable downsamp : integer := 0;
       shared variable freq : integer := 0;
       shared variable dout_int : integer := 0;
       shared variable dout_signed : signed(31 downto 0);
       shared variable zero_flag : integer := 0;
       shared variable my_line : LINE;
       shared variable my_line2 : LINE;
       type TABELA is array(0 to 2047) of signed (15 downto 0);
       signal tab_seno : TABELA;

       --Functions
       function to_string(sv: std_logic_vector; sv2: std_logic_vector) return line is
        variable in_signed : signed(15 downto 0) := signed(sv);
        variable out_signed : signed(20 downto 0) := signed(sv2);
        variable in_int: integer := to_integer(in_signed);
        variable out_int: integer := to_integer(out_signed);
     variable lp: line;
   begin
```

```
        write(lp, integer'image(in_int) & ',' & integer'image(out_int) & ',');
    return lp;
  end;


BEGIN

       --definindo os valores da tabela do seno calculados pelo matlab - 2048 pts
       tab_seno(0) <= X"0009";
       tab_seno(1) <= X"00c9";
       tab_seno(2) <= X"012e";
       tab_seno(3) <= X"0192";
       tab_seno(4) <= X"01f7";
       .
       .
       .
       tab_seno(2044) <= X"fed2";
       tab_seno(2045) <= X"ff37";
       tab_seno(2046) <= X"ff9b";
       tab_seno(2047) <= X"0000";
       --final da atribuição da tabela


   uut: fir_optz PORT MAP (
          clk => clk,
          nd => nd,
                       reset => reset,
          rfd => rfd,
          rdy_down => rdy,
          din => din,
          dout => dout
        );

   -- Clock process definitions
   clk_process :process
   begin
               clk <= '0';
               wait for clk_period/2;
               clk <= '1';
               wait for clk_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.

               reset <= '1';

      wait for 100 ns;

               reset <= '0';
      wait;
   end process;



        stim_proc2: process(clk)
   begin
        --CALCULO DA FREQUENCIA DE ENTRADA DA SENOIDE


   if(clk' event) and (clk = '1') then

               if(rfd = '1') and (nd = '0') then -- ready for data

                       counter := counter + 1;

                       if(counter = 2304) then
                              salto := salto + 1;
                              counter := 0;
                       end if;

                       if(salto = 77) then
                              salto := 0;
                       end if;
```

```
                        if(salto = 0) then
                                pos :=0;
                        else
                                pos := (pos + salto) rem (2048);
                        end if;

                        din <= std_logic_vector(tab_seno(pos));
                        my_line := to_string(din,dout(31 downto 11));
                        writeline(out_file, my_line);
                        nd <= '1';
                else
                        nd <= '0';
            end if;


                if(rdy = '1') then

                        --data_str_in(1) := LF;
                        --my_line := to_string(din,dout(31 downto 11));
                        --writeline(out_file, my_line);
                        my_line2 := to_string(din,dout(31 downto 11));
                        writeline(out_file_D, my_line2);


                        downsamp := downsamp + 1;

                end if;

        end if;

   end process;

END;
Hol



--------------------------------fir_optz-----------------------------------------------
-----------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fir_optz is
    Port ( clk : in  STD_LOGIC;
                        reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (15 downto 0);
           dout : out  STD_LOGIC_VECTOR (31 downto 0));
end fir_optz;

architecture Behavioral of fir_optz is

component fir1_top
    Port ( clk : in  STD_LOGIC;
                        reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (15 downto 0);
           dout : out  STD_LOGIC_VECTOR (38 downto 0));
end component;

component fir2_top
    Port ( clk : in  STD_LOGIC;
```

```
                        reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (17 downto 0);
           dout : out  STD_LOGIC_VECTOR (40 downto 0));
end component;

component fir3_top
    Port ( clk : in  STD_LOGIC;
                        reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (17 downto 0);
           dout : out  STD_LOGIC_VECTOR (40 downto 0));
end component;

component fir4_top
    Port ( clk : in  STD_LOGIC;
                        reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (17 downto 0);
           dout : out  STD_LOGIC_VECTOR (40 downto 0));
end component;


signal dlink1: std_logic_vector(38 downto 0);
signal rdy_down_link1: std_logic;
signal rfd_link1: std_logic;

signal dlink2: std_logic_vector(40 downto 0);
signal rdy_down_link2: std_logic;
signal rfd_link2: std_logic;

signal dlink3: std_logic_vector(40 downto 0);
signal rdy_down_link3: std_logic;
signal rfd_link3: std_logic;

signal dout2: std_logic_vector(40 downto 0);

begin

your_instance_name : fir1_top
              port map (
                       clk => clk,
                       reset => reset,
                       nd => nd,
                       rfd => rfd,
                       rdy_down  => rdy_down_link1,
                       din => din,
                       dout => dlink1);

your_instance_name2 : fir2_top
              port map (
                       clk => clk,
                       reset => reset,
                       nd => rdy_down_link1,
                       rfd => rfd_link1,
                       rdy_down  => rdy_down_link2,
                       din => dlink1(36 downto 19),
                       dout => dlink2);

your_instance_name3 : fir3_top
              port map (
                       clk => clk,
                       reset => reset,
                       nd => rdy_down_link2,
                       rfd => rfd_link2,
                       rdy_down  => rdy_down_link3,
                       din => dlink2(36 downto 19), --dando ganho de 4 bits
                       dout => dlink3);

your_instance_name4 : fir4_top
              port map (
```

```
                        clk => clk,
                        reset => reset,
                        nd => rdy_down_link3,
                        rfd => rfd_link3,
                        rdy_down  => rdy_down,
                        din => dlink3(36 downto 19),
                        dout => dout2);

dout <=        dout2(35 downto 4);--dlink1(37 downto 6);----       dlink2(39 downto 8);
       dlink3(40 downto 9);


end Behavioral;

----------------------------------fir1_top-----------------------------------------
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fir1_top is
    Port ( clk : in  STD_LOGIC;
                        reset : in  STD_LOGIC;
           nd : in  STD_LOGIC;
           rfd : out  STD_LOGIC;
           rdy_down : out  STD_LOGIC;
           din : in  STD_LOGIC_VECTOR (15 downto 0);
           dout : out  STD_LOGIC_VECTOR (38 downto 0));
end fir1_top;

architecture Behavioral of fir1_top is


TYPE DownState_type IS (IDLE, S1, S2);
SIGNAL DownState : DownState_Type;
signal estado: std_logic;
signal rdy: std_logic;

component fir1
       port (
       clk: IN std_logic;
       nd: IN std_logic;
       rfd: OUT std_logic;
       rdy: OUT std_logic;
       din: IN std_logic_VECTOR(15 downto 0);
       dout: OUT std_logic_VECTOR(38 downto 0));
end component;


begin

PROCESS (clk)
BEGIN
       IF clk'event AND clk='1' THEN
              IF (reset = '1') THEN
                     DownState <= IDLE;
                     estado <= '0';
                     rdy_down <= '0';
              ELSE
                     CASE DownState IS
                            WHEN IDLE =>
                                   IF rdy = '1' AND estado = '0' THEN
                                          DownState <= S1;
                                          rdy_down <= '1';
                                   END IF;
                                   IF rdy = '1' AND estado = '1' THEN
                                          DownState <= S2;
                                          rdy_down <= '0';
```

```
                                        END IF;
                                WHEN S1 =>
                                                DownState <= IDLE;
                                                estado <= '1';
                                                rdy_down <= '0';
                                WHEN S2 =>
                                                DownState <= IDLE;
                                                estado <= '0';
                                                rdy_down <= '0';
                        END CASE;
                END IF;
        END IF;
END PROCESS;

your_instance_name : fir1
                port map (
                        clk => clk,
                        nd => nd,
                        rfd => rfd,
                        rdy => rdy,
                        din => din,
                        dout => dout);


end Behavioral;
```

A9 – Sintetizador de senos em Matlab

```
Fs = 122880;                %frequencia de amostragem
Freq = 1920;                %frequencia desejada
N = 1:2048;                 %numero de pontos na tabela
TN = size(N,2);             %idem
salto = round(Freq*TN/Fs); %valor do salto na tabela, num inteiro mais prox
A = 32767;                  %valor da amplitude
V = 32767*sin(2*pi*N/TN);   %um ciclo na tabela

V = round(V)';              %quantização
V(1) = -1;


n = 2048;      %num de pontos para plotar
fid = fopen('seno_2048pts_16bits_hex.txt','wt');
for i=1:1:n
    fix16 = fi(V(i),1,16,0);  %convertendo para ponto fixo 16 bits, com sinal, fração 0
    h16 = hex(fix16);
    fprintf(fid,'tab_seno(%d) <= X"%s";\n',i-1,h16);  %salvando a tabela em um arquivo
txt
end
fclose(fid);



%algoritimo que calcula os n primeios pontos da senoide e plota na tela
for i=1:1:n
  a = (salto*i)/n;  %se o salto passar do tamanho da tabela
  b = a - fix(a);   %i da a volta no valor proporcional
  c = 1 + b*n;
  v(i) = V(c);
end
plot(v(1:n))
```

A10 – Teste do Método Pico no Matlab

```
clear all;
close all;

Fs = 122880;                %frequencia de amostragem
Freq = 300;                 %frequencia desejada
TN = 4096;
salto = round(Freq*TN/Fs); %valor do salto na tabela, num inteiro mais prox
fase_in = 3.00;
fir_delay = 256;

N = 1:TN;                  %numero de pontos na tabela
Tam = size(N,2);             %idem
A = 32767;                  %valor da amplitude
V = 32767*sin(2*pi*N/TN);   %um ciclo na tabela
V2 = 32767*sin(2*pi*N/TN+ fase_in);
%V = round(V)';               %quantização
%V2 = round(V2)';
n = TN;      %num de pontos para plotar

%algoritimo que calcula os n primeios pontos da senoide e plota na tela
a = 1;
for i=1:1:n
  %a = (salto*i)/n;  %se o salto passar do tamanho da tabela
  %b = a - fix(a);   %i da a volta no valor proporcional
  %c = 1 + b*n;
  a = rem((a+salto),TN);
  v(i,1) = V(a+1);
  v(i,2) = V2(a+1);

end

INTERP = 1;
max_in = 1;
max_out = 1;
if(Freq==60)
    a = 1;
    b = TN;
    per_pt = INTERP*TN;
else
    per_pt = INTERP*Fs/Freq;  %periodo em pts para a freq
    a = INTERP*fir_delay;
    b = a+per_pt;
end

per_inc = round(per_pt);

u(:,1) = interp(v(:,1),INTERP,8,0.5);
u(:,2) = interp(v(:,2),INTERP,8,0.5);
for i=a:1:b

    if( u(i+1,1)> u(max_in,1))
        max_in = i+1;
    end

    if( u(i+1,2)> u(max_out,2))
        max_out = i+1;
    end

    diff_pts = (max_in - max_out);

end

plot(u);
hold;
plot(v);

gain = u(max_out,2)/u(max_in,1)
fase = ((max_in - max_out)/per_pt)*2*pi
erro_fase = abs((fase_in-fase)/(fase_in))*100
```

A11 – Teste do Método FFT no Matlab

```
clear all;
close all;

Fs = 122880;                %frequencia de amostragem
Freq = 300;                 %frequencia desejada
TN = 4096;
salto = round(Freq*TN/Fs); %valor do salto na tabela, num inteiro mais prox
fase_in = 5.2;
G = 0.8;
fir_delay = 256;

N = 1:TN;                  %numero de pontos na tabela
Tam = size(N,2);             %idem
A = 32767;                  %valor da amplitude
V = A*sin(2*pi*N/TN);    %um ciclo na tabela
V2 = G*A*sin(2*pi*N/TN+fase_in);
%V = round(V)';               %quantização
%V2 = round(V2)';
n = TN;       %num de pontos para plotar

%algoritimo que calcula os n primeios pontos da senoide e plota na tela
a = 1;
for i=1:1:n
  %a = (salto*i)/n;  %se o salto passar do tamanho da tabela
  %b = a - fix(a);   %i da a volta no valor proporcional
  %c = 1 + b*n;
  a = rem((a+salto),TN);
  v(i,1) = V(a+1);
  v(i,2) = V2(a+1);

end

INTERP = 1;
max_in = 1;
max_out = 1;
if(Freq==60)
    a = 1;
    b = TN;
    per_pt = INTERP*TN;
else
    per_pt = INTERP*Fs/Freq;  %periodo em pts para a freq
    a = INTERP*fir_delay;
    b = a+per_pt;
end

per_inc = round(per_pt);

u(:,1) = interp(v(:,1),INTERP,8,0.5);
u(:,2) = interp(v(:,2),INTERP,8,0.5);
for i=a:1:b

    if( u(i+1,1)> u(max_in,1))
        max_in = i+1;
    end

    if( u(i+1,2)> u(max_out,2))
        max_out = i+1;
    end

    diff_pts = (max_in - max_out);

end

plot(u);
hold;
plot(v);


gain = u(max_out,2)/u(max_in,1)
fase = ((max_in - max_out)/per_pt)*2*pi
erro_fase = abs((fase-fase_in)/(fase_in))*100
```

```
%calculando magnitude e fase pela fft

%plot(u)
i = fft(u(:,1),TN);
o = fft(u(:,2),TN);

m=0;
for j=1:TN

    if(m < abs(o(j)))
      m = abs(o(j));
      n = j;
    end
end
%figure;
%plot(abs(o(1:30+n)));
%figure;
%plot(abs(i(1:30+n)));

angle2 = angle(o(n)) - angle(i(n))
mag2_i = abs(i(n))/(2048);
mag2_o = abs(o(n))/(2048);
gain = mag2_o / mag2_i
```

A12 – *Script* Completo que calcula a resposta em freqüência para Filtro do FPGA

```
close all;
clear all;
in = load('out2.mat');

%sinal de entrada é composto por frequencias (senos adjacentes) começando em 60Hz de
%até 4500Hz, crescendo de 60 em 60Hz. Cada frequencia dura 2048 pontos.

%Fin = 60, 120, 180, 240, ... 4500

Fs = 122880;                %frequencia de amostragem
D = 256;                    %numero de pontos para compensar o transiente do filtro.
M = 2048;                   %n ultimos pontos para calcular fft
N = M+D;                    %numero de pontos para cada harmonico de 60Hz
NFreq = 74;                 %numero de frequencias da PA de 60Hz
FIRd = D;                   %atraso do filtro
Tam = size(in.out,1);      %tamanho do vetor de entrada
INTRP = 10;                 %interpolation factor
DR = 16;                    %downsampler rate
INTRPD = INTRP*DR;
MD = M/DR;

%----------transformando o arquivo de saída do VHDL em matrizes in e out---

k = 0;
j = 1;
inD = load('outD2.mat');
Tam16 = size(inD.outD,1) - N/DR;
for j=0:round(N/DR):Tam16
    k = k+1;
    for i=1:1:round(N/DR)
        in1D(i,k)  = inD.outD(i+j,1);
        out1D(i,k) = inD.outD(i+j,2);
    end
end

A = interp(inD.outD(:,2),16);
k = 0;
j = 1;
for j=0:N:Tam-N
    k = k+1;
    for i=1:1:N
        in1(i,k)  = in.out(i+j,1);
        out1(i,k) = in.out(i+j,2);%A(i+j);
    end
end


%------calculando a Resposta em Frequencia pela FFT-----------------------
for j=1:1:NFreq

    max_in=1;
    max_out=1;

    F_in1 = fft(in1(D:N,j),M);
    F_out1 = fft(out1(D:N,j),M);

    m=0;n=0;
    for l=1:M
        if(m < abs(F_in1(l)))
          m = abs(F_in1(l));
          n = l;
        end
    end

    mag2_in = abs(F_in1(n))/M;
    mag2_out = abs(F_out1(n))/M;
    gainFTT = mag2_out / mag2_in;
```

```
    angleFFT = angle(F_out1(n)) - angle(F_in1(n));
    angleFFT = func_rad2deg(angleFFT);

    Resp_Freq_FFT(j,1) = 20*log10(gainFTT);
    Resp_Freq_FFT(j,2) = angleFFT;


end
%------calculando a Resposta em Frequencia pela FFT c/  DOWNSAMPLER-----------
for j=1:1:NFreq

    max_in=1;
    max_out=1;

%incluindo zeros para aumentar a resolução do espectro
%     MD = M;
%     s = out1D((N-M)/DR:N/DR,j);
%     s = [s ; zeros(1920,1)];
%     F_outD = fft(s,M);
%
%     s = in1D((N-M)/DR:N/DR,j);
%     s = [s ; zeros(1920,1)];
%     F_inD = fft(s,M);

    F_outD = fft(out1D(D/DR:N/DR,j),MD);
    F_inD = fft(in1D(D/DR:N/DR,j),MD);


    m=0;n=0;
    for l=1:MD
        if(m < abs(F_inD(l)))
          m = abs(F_inD(l));
          n = l;
        end
    end

    magD = abs(F_outD(n))/MD;
    magDin = abs(F_inD(n))/MD;
    gainD = magD/magDin;
    angleD = angle(F_outD(n)) - angle(F_inD(n));
    angleD = func_rad2deg(angleD);

    Resp_Freq_FFT_D(j,1) = 20*log10(gainD);
    Resp_Freq_FFT_D(j,2) = angleD;

end

f = (0:NFreq-1)*60;


%----Calculando a resposta em frequencia resultante dos 4 filtros----------

h_ideal = load('fir_coe_float.mat');
[H_ideal,w_ideal] = freqz(h_ideal.float,1,1024);
for i=1:1:size(H_ideal,2)
    fH_ideal(i) = func_rad2deg(angle(H_ideal(i)));
end
f3 = w_ideal*Fs/(2*pi);

h0 = load('h1.mat');
h1 = load('h2.mat');

for i=1:size(h0.h,2)
   hr2((i-1)*2+1) = h0.h(i);
   for j=1:1
        hr2((i-1)*2+i+j) = 0;
   end
end

for i=1:size(h0.h,2)
   hr3((i-1)*4+1) = h0.h(i);
   for j=1:3
        hr3((i-1)*4+i+j) = 0;
   end
end

for i=1:size(h0.h,2)
```

```
   hr4((i-1)*8+1) = h0.h(i);
   for j=1:7
        hr4((i-1)*8+i+j) = 0;
   end
end

[H_r1,w_r1] = freqz(h0.h,1,1024);
[H_r2,w_r2] = freqz(hr2,1,1024);
[H_r3,w_r3] = freqz(hr3,1,1024);
[H_r4,w_r4] = freqz(hr4,1,1024);

fr1 = w_r1*Fs/(2*pi);
fr2 = w_r2*Fs/(2*pi);
fr3 = w_r3*Fs/(2*pi);
fr4 = w_r4*Fs/(2*pi);

%plotando a resposta teórica dos 4 filtros
plot(fr1,20*log10(abs(H_r1)));
title('Magnitude da R. F. do FIR FPGA - H4(Z) menor atenuacao')
xlabel('Hz')
ylabel('dB')
figure;
plot(fr2,20*log10(abs(H_r2)));
title('Magnitude da R. F. do FIR FPGA - H3(Z^2) menor atenuacao')
xlabel('Hz')
ylabel('dB')
figure;
plot(fr3,20*log10(abs(H_r3)));
title('Magnitude da R. F. do FIR FPGA - H2(Z^4) menor atenuacao')
xlabel('Hz')
ylabel('dB')
figure;
plot(fr4,20*log10(abs(H_r4)));
title('Magnitude da R. F. do FIR FPGA - H1(Z^8) menor atenuacao')
xlabel('Hz')
ylabel('dB')


%calculando o filtro resultante
hr_2 = conv(conv(conv(h0.h,hr2),hr3),hr4);
[Hr2,w_r2] = freqz(hr_2,1,1024);

for i=1:1:size(Hr2,1)
    fase_Hr(i) = func_rad2deg(angle(Hr2(i)));
end

fr = w_r2*Fs/(2*pi);

%----Calculando a resposta em frequencia do filtro ideal em float----------



coef1 = load('fir_coe_float.mat');
[h,w] = freqz(coef1.float,1,1024);

coef2 = load('fir_coe_fix.mat');
[h_fix,w_fix] = freqz(coef2.fix,1,1024);

for i=1:1:size(h,1)
    phase_h(i) = func_rad2deg(angle(h(i)));
    phase_h_fix(i) = func_rad2deg(angle(h_fix(i)));
end

f1 = w*Fs/(2*pi);
f2 = w_fix*Fs/(2*pi);


%----Plotando os valores de resposta em frequencia-------------------------

mx = 75;
%magnitude
figure;


plot(fr(1:mx),20*log10(abs(Hr2(1:mx))),'red');
hold;
plot(fr(1:mx),20*log10(abs(H_ideal(1:mx))));
```

```
plot(f,Resp_Freq_FFT_D(:,1),'black');
legend('FIXED 4F MATLAB','FPGA 4F FFT D16',1);
title('Magnitude da R. F. do FIR FPGA - 4 Filtros menor atenuacao')
xlabel('Hz')
ylabel('dB')

%fase
figure;
plot(fr(1:mx),fase_Hr(1:mx),'red');
hold;
plot(f,Resp_Freq_FFT_D(:,2),'black');
legend('FIXED 4F MATLAB','FPGA 4F FFT D16',1);
title('Fase da R. F. do FIR FPGA - 4 Filtros menor atenuacao')
xlabel('Hz')
ylabel('graus')


% %----Plotando o erro da resposta em frequencia ------------------------------
%
% fqs = 60;
% for i=1:1:fqs
%
%     erro_mag(i) = 100*(1-(Resp_Freq_FFT(i,1)/(20*log10(abs(h(i+1))))));
%     ang3 = Resp_Freq_FFT(i,2) - phase_h(i+1);
%     if(ang3 > 180)
%         ang3 = ang3-360;
%     end
%
%     erro_ang(i) = ang3;
% end
%
% figure;
% plot(f(1:fqs),erro_mag);
% title('Erro da Magnitude da FPGA em relação a Magnitude do MATLAB')
% xlabel('Hz')
% ylabel('%')
%
% figure;
% plot(f(1:fqs),erro_ang);
% title('Erro da Fase da FPGA em relação a Fase do MATLAB')
% xlabel('Hz')
% ylabel('graus')
%
% for i=2:1:fqs
%     diff(i) = erro_ang(i) - erro_ang(i-1);
% end
% diff(1) = diff(2);
%
% figure;
% plot(f(1:fqs),diff);
% title('Derivada do erro da Fase da FPGA em relação a Fase do MATLAB')
% xlabel('Hz')
ylabel('graus')
```