Aluno: Rariman Lara Silva da Rocha

A Estimação da Frequência em Sistemas Elétricos de Potência usando uma Filtragem Adaptativa

Trabalho de Conclusão de Curso
apresentado à Escola de Engenharia de São
Carlos na Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase
em Sistemas de Energia e Automação

ORIENTADOR: Prof. Dr. Mario Oleskovicz

São Carlos
2010

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

R672e

Rocha, Rariman Lara Silva da
Estimação da frequência em sistemas elétricos de potência usando uma filtragem adaptativa / Rariman Lara Silva da Rocha ; orientador Mario Oleskovicz. -- São Carlos, 2010.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2010.

1. Sistemas elétricos de potência. 2. Estimação da frequência. 3. Filtragem adaptativa. 4. Métodos dos mínimos quadrados. 5.VHDL. 6. FPGA. I. Título.

Dedicatória

A minha mãe Heloisa e a toda minha família pelo incansável apoio ao longo do período da minha graduação.

Agradecimentos

- Em primeiro lugar a Deus pela minha vida e família.
- Ao professor Dr. Mario Oleskovicz, pela orientação, confiança, atenção, apoio e incentivo na elaboração deste trabalho.
- Aos meus companheiros de laboratório Daniel, Monaro, Ulisses e Janison , pela colaboração essencial na construção deste trabalho.
- A todos os professores e funcionários do Departamento de Engenharia Elétrica, que de algum modo colaboraram durante a minha graduação nesta universidade.
- Ao próprio Departamento de Engenharia Elétrica da Escola de Engenharia de São Carlos (EESC) - Universidade de São Paulo.
- Ao Laboratório de Sistemas de Energia Elétrica - LSEE, pela participação e fornecimento de todas as condições necessárias para o desenvolvimento deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AG Algoritmos Genéticos

ANEEL Agência Nacional de Energia Elétrica

ASIC Application-Specific Integrated Circuits

ATP *Alternative Transients Program*

CAD *Conversor Analógico Digital*

CLB *Configurable Logic Blocks*

CMOS *Complementary Metal Oxide Semiconductor*

CPFL *Companhia Paulista de Força e Luz*

DoD *Department of Defense*

DSP *Digital Signal Processing*

FPGA *Field Programmable Gate Array*

IEEE *Institute of Electrical and Electronic Engineers*

IOB *Input Output Blocks*

LE *Lógic Element*

LF *Lógica Fuzzy*

LUT *Lookup Table*

LMS *Least Mean Square*

LSI *Large Scale Integration*

MIT *Motor de Indução Trifásico*

MMQ *Método dos Minímos Quadrados*

ONS *Operador Nacional do Sistema*

PLD *Programmable Logic Devices*

PRODIST *Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional*

PROM *Programmable Read Only Memory*

pu *por unidade*

RAT *Regulador Automático de Tensão*

RNA *Redes Neurais Artificiais*

RP *Relé de Proteção*

SEP *Sistema Elétrico de Potência*

SRAM *Static Random Allocation Memory*

TDF *Transformada Discreta de Fourier*

VHDL *VHSIC Description Language*

VHSIC *Very High Speed Integrated Circuit*

VLSI *Very Large Scale Integration*

Resumo

Rocha, R. L. S. A Estimação da Frequência em Sistemas Elétricos de Potência usando uma Filtragem Adaptativa. São Carlos, 2010. Trabalho de Conclusão de Curso - Escola de Engenharia de São Carlos, Universidade de São Paulo.

Este trabalho de conclusão de curso apresenta um método de estimação da frequência em Sistemas Elétricos de Potência (SEP) usando uma filtragem adaptativa baseada no método dos mínimos quadrados - MMQ (*LMS - Least Mean Square*). Pela lógica estudada, as amostras digitalizadas das tensões trifásicas do sistema elétrico em análise foram submetidas a transformada $\alpha\beta$, resultando em um sinal na forma complexa. Em seguida, este sinal foi direcionado ao filtro adaptativo baseado no MMQ que permitiu o rastreamento do sinal de entrada e, consequentemente, a estimativa do valor da frequência. O desenvolvimento do trabalho consistiu no entendimento e aprimoramento do algoritmo responsável por esta tarefa. Tal algoritmo foi compilado em linguagem de programação *C* e, posteriormente, convertido para a linguagem VHDL (*VHSIC Hardware Description Language*). Justifica-se a implementação do algoritmo em VHDL no sentido de viabilizar a sua aplicação *on-line* em um SEP dispondo da tecnologia *Field Programmable Gate Array* (FPGA). Resultados satisfatórios foram encontrados pela aplicação desenvolvida, os quais serão reportados e comentados neste documento.

Palavras–Chave: Estimação da Frequência, Filtragem adaptativa, Sistema Elétrico de Potência, Método dos mínimos quadrados, VHDL e FPGA.

Abstract

Rocha, R. L. S. Frequency Estimation in a Power System using Adaptive Filtering. São Carlos, 2010.Course Conclusion Work - Engineering School of São Carlos, University of São Paulo.

This course conclusion report presents a method of frequency estimation in power systems using adaptive filtering based on Least Mean Square (LMS). By the logic studied, the power system's digitalized samples of the tree-phase voltages in analysis were submitted to $\alpha\beta$-transform, resulting in a complex form signal. Then, this signal was directed to the adaptive filter based on LMS which allowed the tracking of the input signal and, consequently, the estimative of the frequency value. This work's development consisted in understanding and improvement of the algorithm responsible for this task. Such algorithm was compiled in C programming language and, after that, converted to VHSIC Hardware Description Language (VHDL). The implementation of the algorithm in VHDL is justified in order to enable its on-line application in a Power System disposing of the Field Programmable Gate Array (FPGA) technology. Satisfactory results were found by the developed application, which will be reported and commented further in this report.

Keywords: Frequency Estimation, Adaptive Filter, Power System, Least Mean Square, VHDL and FPGA.

Capítulo 1

Introdução

Quando um sistema elétrico de potência (SEP) está operando normalmente, a quantidade de energia gerada é igual aquela que é consumida pela carga. Porém, os sistemas de potência estão constantemente sujeitos a distúrbios devidos ao comportamento dinâmico da carga, á faltas originadas por causas naturais e, algumas vezes, resultantes de falhas na operação dos equipamentos ligados na rede elétrica. Dentre os danos causados por essas situações indesejáveis estão os impactos sofridos pelas linhas de transmissão e de distribuição, banco de capacitores, máquinas elétricas e entre outros equipamentos e/ou dispositivos.

Em geral, como observado das situações práticas e das características do sistema, quando ocorre uma situação de falta (defeito e/ou curto-circuito), as magnitudes das correntes aumentam enquanto que as das tensões sofrem uma queda. Junto dessas mudanças, outras alterações podem ocorrer em um ou mais dos seguintes parâmetros: no ângulo de fase, na composição harmônica, nas potências ativa e reativa e na frequência. Dependendo da severidade destas perturbações , o SEP se mantém próximo a situação de regime permanente devido a proporção do sistema de potência em relação às cargas individuais ou geradores, e pela rápida ação do sistema de proteção (Horowitz e Phadke, 1996).

Sendo assim, tem-se que o objetivo de um sistema de proteção é agir rapidamente no isolamento da área afetada de modo que seja garantida a continuidade da operação do SEP. Dentre os dispositivos mais utilizados para realizar esta tarefa encontram-se os relés, que são equipamentos capazes de detectar condições anormais e inicializar ações de correção tão rápido quanto possível com o intuito de retornar o SEP ao seu estado normal de operação. Cabe destacar que esta ação tem que ser automática, rápida, eficaz e, ao mesmo tempo, causar o isolamento da menor seção

possível sobre o sistema protegido.

Os relés de proteção (RP) podem ser classificados de acordo com as suas funcionalidades, os princípios de operações, os parâmetros que monitoram e a arquitetura (Blackburn, 1987). Os primeiros RP eram eletromecânicos, baseados na interação eletromagnética entre as correntes elétricas e o fluxo magnético sobre um condutor móvel (geralmente um disco ou um cilindro). Tais equipamentos podem utilizar múltiplos sinais de entrada, porém possuem somente um determinado tipo de função de proteção, ou seja, são monofuncionais. Na Figura 1.1, são ilustrados os principais componentes que constituem o relé eletromecânico.

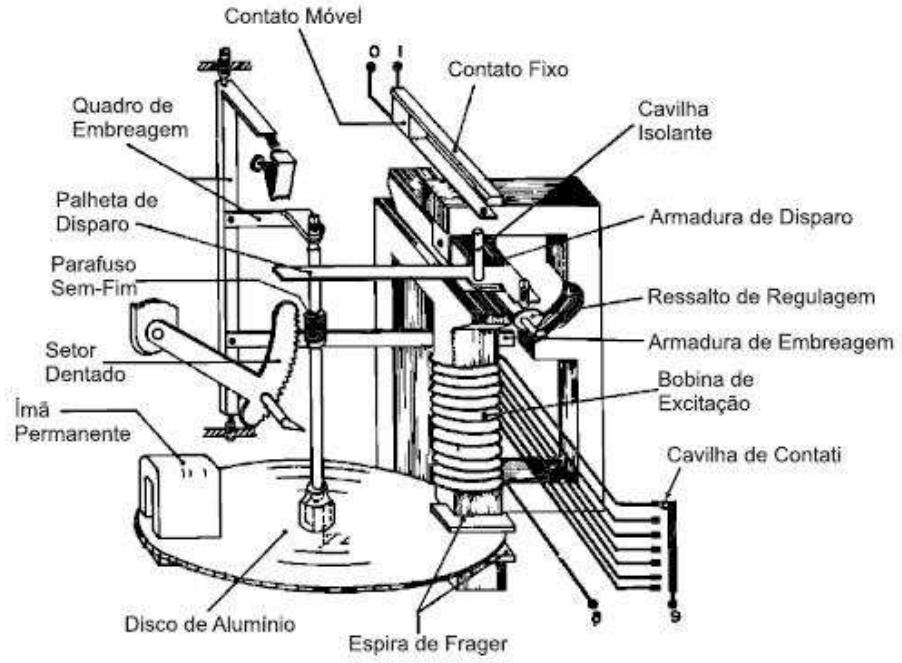

Figura 1.1: Principais componentes de um relé eletromecânico (Caminha, 1977)

Em seguida, devido ao aumento da complexidade dos SEP e á necessidade de equipamentos mais rápidos e confiáveis, surgiram os RP de estado sólido que utilizavam dispositivos semicondutores em seu projeto, no qual podiam empregar uma representação lógica em sua saída. A Figura 1.2 mostra uma possível configuração de um relé de estadom sólido para a função de sobrecorrente instantânea. Aléms permitirem maior confiabilidade e rapidez, estes tipos de relés necessitavam de menor espaço físico para das cabines de proteção nas subestações. Porém, estes dispositivos apresentam problemas com relação á condição do ambiente de instalação, como a umidade e temperatura.

Os RPs mais modernos são conhecidos como digitais ou microprocessados, onde é realizada uma amostragem dos sinais de entrada para uma aplicação em um algoritmo numérico arma-

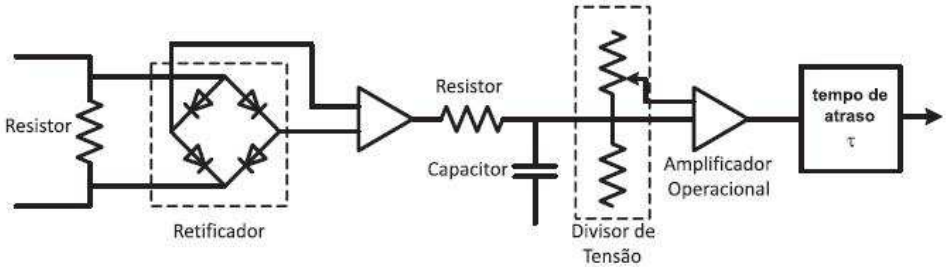

Figura 1.2: Esquema de relé de estado sólido (Horowitz e Phadke, 1996)

zenado na memória (Horowitz e Phadke, 1996). Esses RPs tomam a decisão de abertura do disjuntor (*trip*) baseada pelo resultado de operações matemáticas, o que possibilita a interação de várias funções em um único equipamento. Na Figura 1.3 é mostrado o diagrama funcional simplificado de um relé microprocessado.

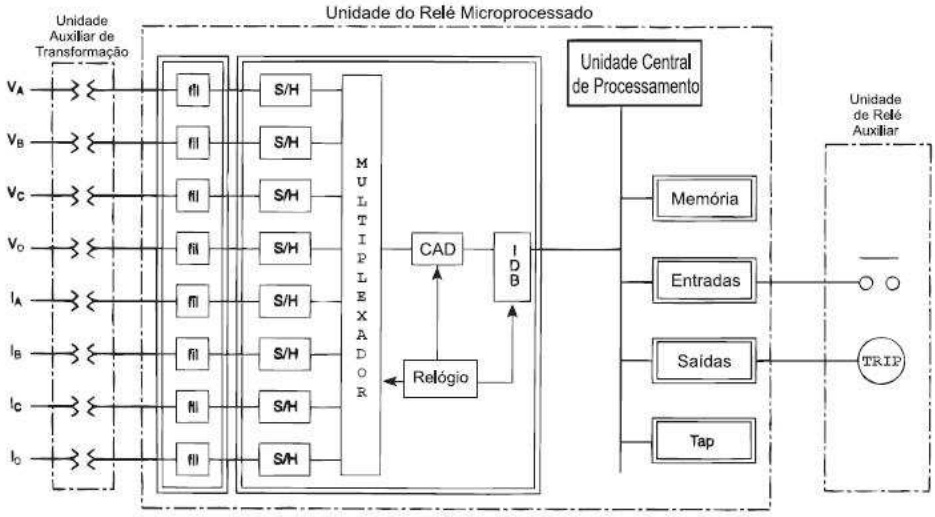

Figura 1.3: Esquema de relé microprocessado (Johns e Salman, 1995)

Quanto ás grandezas que monitoram, geralmente os RPs são empregados para extrair informações sobre os fasores das tensões e correntes, parâmetros de sequência positiva, as potências ativa e reativa, o fator de potência e, o alvo desta pesquisa, a frequência do sistema. Vale frisar que a frequência é uns dos parâmetros de maior importância em um SEP. Desse modo, foi estabelecido um valor nominal para a frequência do sistema em 60 Hz, podendo esta variar em uma faixa de $\pm$0,5 Hz (ONS, 2001). Entretanto, tem-se da prática que estes limites podem ser ultrapassados devido a própria dinâmica de operação do sistema, ou pela perda repentina de potência, ocasionando transtornos aos usuários da rede elétrica. Como por exemplo, pode-se citar a interrupção no processamento de informações, a alteração das características elétricas e o

comprometimento da vida útil de equipamentos, as mudanças da componente reativa das linhas de transmissão e variações na velocidade de rotação no eixo de geradores, visto que a frequência apresenta grande influência na operação dos mesmos.

Quando a frequência é menor do que o seu valor nominal implica que o sistema está sobrecarregado. Nessa situação para retomar o sincronismo do sistema, são utilizados os relés de sub-frequência que agem na desconexão de blocos de carga em um processo chamado de rejeição de carga. Já se a frequência está acima da tolerância permitida, a geração de energia é maior do que a carga. Nesse caso, são empregados os relés de sobre-frequência na proteção dos geradores (Sachdev e Giray, 1985).

No contexto apresentado, o objetivo deste trabalho é apresentar um método de estimação da frequência utilizando a filtragem adaptativa baseada na forma complexa do Método dos Minímos Quadrados (MMQ). O desenvolvimento a ser apresentado estará baseado na conversão de um algoritmo em linguagem *C* (Barbosa, 2007) para a plataforma VHDL (*VHSIC Hardware Description Language*). Do algoritmo implementado em VHDL espera-se a sua posterior aplicação *on-line* em um SEP empregando a tecnologia *Field Programmable Gate Array* (FPGA).

A linguagem VHDL será empregada para descrever e avaliar o comportamento da futura aplicação em FPGA. A motivação fundamental para usar a linguagem VHDL é que esta é adotada como padrão na indústria para a descrição de *hardware* em nível abstrato. Um comentário importante, considerando VHDL, é que contrário aos programas dos computadores tradicionais, os quais trabalham seqüencialmente, nesta linguagem suas declarações são inerentemente concorrentes (paralelas). Por essa razão, VHDL é usualmente referida como um código antes que um programa. Do algoritmo implementado em linguagem VHDL espera-se que este venha a apresentar resultados muito próximos aqueles a serem observados no FPGA.

Já o FPGA consiste em um arranjo matricial de blocos lógicos configuráveis contido em um único *chip*. Cada um desses blocos contém uma capacidade computacional para implementar funções lógicas, aonde são empregadas células de armazenamento e elementos sequênciais. Além disso, o FPGA também dispõe de recursos de interconexão prográmavel para realizar a comunicação entre tais blocos lógicos sem introduzir grande atraso de tempo.

Capítulo 2

Revisão Bibliográfica

Esta revisão bibliográfica visa levantar alguns dos diversos algoritmos presentes na literatura que são aplicados para a estimação da freqüência em um SEP. Uma vez que a freqüência representa um importante parâmetro de operação, nas utimas décadas, houve um grande avanço no número de publicações referentes a este tema. Cabe frisar que as técnicas aqui resumidas utilizam as amostras digitalizadas das formas de ondas das tensões do SEP para monitorar o comportamento dinâmico da freqüência. Dentre os métodos numéricos mais comuns, pode-se citar: a deteção dos pontos de referência; a transformada de *Fourier*; método de *Prony*; método dos mínimos quadrados; filtro de *Kalman* e sistemas inteligentes.

2.1 Detecção dos pontos de referência

Estes métodos consideram a tensão elétrica como sendo um sinal puramente senoidal. Desse modo, são detectados os cruzamentos consecutivos desta forma de onda no eixo das abcissas, sendo possível a obtencão do intervalo de tempo entre estes pontos e, conseqüentemente, a freqüência do sistema. Vale ressaltar que a precisão desta técnica apresenta grande sensibilidade quanto á presença de ruídos e componentes harmônicas.

Salcic e Mikhael (1999) propuseram a determinação dos pontos de referência usando as propriedades simétricas do sinal de entrada, detectando somente os locais onde a parte positiva possui simetria máxima. Os efeitos do ruído e das componentes harmônicas são removidos através da correlação cruzada entre o sinal original e o corrompido. Os pontos encontrados são empregados para medir o tempo entre dois ciclos consecutivos e, desse modo, a freqüência da forma de onda.

Já em Xue e Yang (2007), é utilizado o método da interpolação cúbica para encontrar os locais que ocorrem os cruzamentos no eixo das abcissas e melhorar a precisão da estimação. Segundo os autores, o algoritmo mostrou um bom desempenho para sinais não estacionários e poluídos.

2.2 Transformada de Fourier

A Transformada de *Fourier* é uma das ferramentas matématicas mais utilizadas em processamento digital de sinais onde uma determinada entrada pode ser representada por uma superposição de senóides complexas.

Pelos relatos encontrados, tem-se que Girgs e Ham (1982) foram os percursores na utilização do método de *Fourier* em relés de freqüência. A técnica implementada é capaz de calcular o valor e a taxa de variação da freqüência fundamental a partir da análise do coeficiente de *leakage* (espraiamento), onde, caso haja desvios na freqüência da rede, as componentes da TDF (Transformada Discreta de *Fourier*) serão diferentes de zero e, portanto, os desvios podem ser detectados. Esta técnica tem a capacidade de prever a porcentagem da geração ou da carga em desequilíbrio, permitindo que o sistema seja restaurado para as condições normais de operação através do processo de rejeição de carga.

Em Chen e Yu (2003), é proposto um algoritmo baseado na TDF para determinar as componentes harmônicas e o afundamento de tensão da forma de onda sobre um determinado SEP. Desde que a freqüência de uma harmônica é a freqüência fundamental multiplicada por um múltiplo inteiro, esta pode ser diretamente encontrada.

Similarmente ao trabalho anterior, Lu (2005) apresenta um filtro fundamentado na TDF para a medição das componentes harmônicas no domínio do tempo e da frequência. O filtro é baseado no processamento digital de sinais (DSP - *Digital Signal Processing*), permitindo que a taxa de amostragem possa ser variada sem que efeitos indesejáveis, como o erro de quantização, venha a afetar a estimação. De acordo com o autor, uns dos problemas encontrados foi o grande esforço computacional gerado pelo armazenamento de grande quantidade de dados no projeto do filtro.

Yulan e Cunyang (2007) propõem uma técnica que aplica a TDF para calcular as componentes real e imaginária da onda que representa o comportamento da tensão no sistema elétrico em questão. Em seguida, a freqüência do sinal resultante é obtida através da determinação dos locais de cruzamento no eixo das abcissas com o emprego de uma aproximação de curvas por

interpolação quadrática.

2.3 Método de *Prony*

O método de estimação de *Prony* consiste em realizar uma linearização apropriada de um problema não linear através de uma aproximação de modo que o erro calculado seja minimizado.

Sendo assim, Lobos e Rezmer (1997) sugerem um novo método para estimar a freqüência do sistema elétrico na presença de componentes harmônicas de alta ordem. Este procedimento consiste inicialmente em filtrar a forma de onda da tensão através de um algoritimo baseado na TDF. Porém, se ocorrer desequilíbrios no sistema, tal filtragem não será exata. Desse modo, utiliza-se o sinal de saída obtido anteriormente para calcular o valor verdadeiro da freqüência usando o método de *Prony* por meio de uma aproximação linear. Segundo os autores, o tempo de resposta para esse método é de três a quatro períodos em relação á componente fundamental.

Já o trabalho desenvolvido por Zivanovic (2005) emprega uma modificação no tradicional método de *Prony* inserindo iterações adcionais de modo que a atenuação causada por ruídos seja eliminada e a precisão na estimação melhorada.

2.4 Método dos Mínimos Quadrados

O algoritmo do método dos mínimos quadrados (MMQ) adaptativo foi originalmente desenvolvido por Widrow e Hoff (1960). A principal caracterÍstica desta técnica é a simplicidade, pois, não há a necessidade do cálculo de matrizes inversas e funções de correlação, tornando-a muito popular (Haykin, 2002). Além disso, o MMQ é considerado de fácil implementação, apresentando grande eficiência computacional e robustez.

Sachdev e Giray (1985) propõe um método através do qual o erro quadrático é minimizado. Para tal, utiliza-se da forma expandida da onda que representa a tensão do SEP através da série de *Taylor*. Similarmente, Sanaye-Pasand e Marandi (2004) combinam o método de pontos de referência e a série de *Taylor*. O valor obtido a partir da primeira técnica é utilizado como condição inicial para encontrar a freqüência verdadeira da rede elétrica através da série de *Taylor*.

Pradhan et al. (2005) empregam a transformada $\alpha\beta$ para obter um vetor complexo a partir das tensões trifásicas do sistema elétrico de potência. O vetor obtido é então utilizado como entrada para o ajuste dos coeficientes em um filtro adaptativo baseado no MMQ, onde que a

freqüência é estimada após a determinação dos mesmos. O algoritmo apresenta um parâmetro de convergência variável para aumentar a sua performance e robustez. Os autores enfatizam que os resultados mostraram-se rápidos e precisos na presença de distorções e mesmo em situações nas quais a freqüência foi variada.

2.5 Filtros de *Kalman*

O filtro de *Kalman* tem a sua formulação matemática representada em termos de espaços de estados. A principal característica deste tipo de filtragem é a capacidade de obter a variância do estado de um sistema dinâmico, onde o resultado encontrado na estimação, é estatisticamente ótimo (Haykin, 2002). Além disso, este algoritmo é amplamente utilizado em relés micropocessados para a estimação da freqüência e de componemtes harmônicas.

Girgs e Hwang (1984) propõem o uso do filtro de *Kalman* estendido para a medição *on-line* dos valores da amplitude da tensão elétrica, do ângulo de fase e do desvio da freqüência do sistema. De modo similar, Aghazadeh et al. (2005) descrevem um método que possui a capacidade de estimar simultaneamente a freqüência e a magnitude dos sinais de tensão e corrente presentes em um SEP.

O trabalho desenvolvido por Dash et al. (1999) utiliza os valores discretos das tensões trifásicas do sistema de potência para encontrar um vetor complexo a partir da transformada $\alpha\beta$. Através deste vetor complexo, é formulado um espaço de estado não-linear onde o filtro de *Kalman* estendido é utilizado para calcular o estado verdadeiro na presença de ruídos e componentes harmônicas. Como a freqüência é considerada uma variavel de estado, o estado verdadeiro proporciona o valor desconhecido desta. Os resultados destacam que a velocidade de convergência é reduzida para três ciclos quando os sinais de tensão são altamente poluídos, mas, segundo os autores, esta situação pode ser contornada se as harmônicas também forem empregadas como uma variável de estado.

2.6 Sistemas Inteligentes

Os sistemas inteligentes pertencem a uma ampla área da computação. Esses sistemas são inspirados no comportamento humano ou na natureza, onde se tem a tentativa de reproduzí-los. Dentre outras, aqui serão citadas as técnicas comumente aplicadas à SEPs como os algoritmos

genéticos, redes neurais artificiais e a lógica *Fuzzy*.

2.6.1 Algoritmos Genéticos

Os algoritmos genéticos (AGs) pertencem a família dos algoritmos inspirados na teoria da evolução das espécies e da sobrevivência dos mais aptos de Darwin (2004). Por esta teoria, tem-se a codificação por soluções potenciais, de um determinado problema baseado em uma estrutura de dados similar a de um cromossomo, onde se aplicam operadores genéticos e uma função de aptidão á tais estruturas preservando as informações críticas (Whitley, 1993).

O uso de AG se inicia com um conjunto de cromossomos que representa uma população de indivíduos. A partir desta população, novos indivíduos são gerados através de mecanismos como: seleção, reprodução e mutação. Desse modo, os indivíduos que possuem melhor aptidão representam soluções potenciais para um certo problema, sendo que, a cada iteração acontece a busca pelo ótimo global do mesmo. Na Figura 2.1 é mostrado o diagrama típico de um sistema baseado em AG.

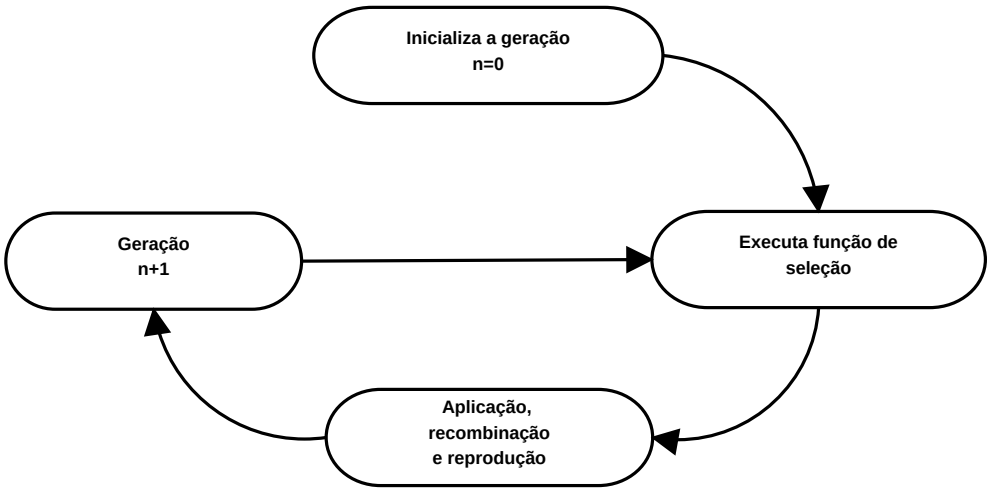

Figura 2.1: Ciclo evolutivo de um sistema baseado em AG.

El-Naggar e Youssef (2000) sugerem o uso dos algoritmos genéticos em um relé para o cálculo *on-line* do desvio da freqüência, a amplitude e o ângulo de fase do sinal de entrada. O algoritmo é submetido à diversas situações onde os parâmetros da forma de onda são variados de modo a minimizar o erro estimado.

O trabalho desenvolvido por Vargas et al. (2005) objetiva a estimação dos parâmetros associados a forma de onda como amplitude, freqüência e ângulo de fase direcionados a uma proposição do relé de freqüência também baseado em AG.

2.6.2 Redes Neurais Artificiais

As redes neurais artificiais (RNAs) são algoritmos que procuram imitar o funcionamento do cérebro humano. Esses sistemas são formados por elementos de processamentos interligados, chamados de neurônios, os quais são dispostos em camadas sendo responsáveis pela não linearidade da rede, através da execução interna de funções matemáticas. Uma característica importante das RNAs é a aprendizagem das mesmas através de exemplos que lhes são apresentados. A Figura 2.2 ilustra uma arquitetura de RNA com três entradas, na qual são evidenciados o combinador linear, a função de aptidão g(.) e θ é o limiar.

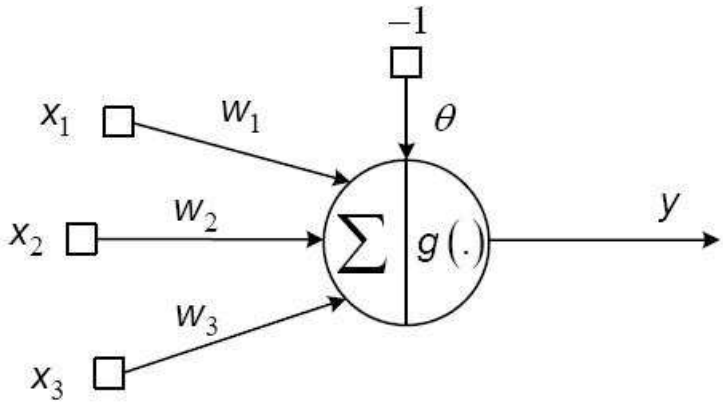

Figura 2.2: Arquitetura de uma RNA com três entradas.

Girgs e Hwang (1997) utilizam as RNAs para encontrar o valor da freqüência da rede. Essa aproximação usa uma estrutura de neurônio adaptativo linear chamado de "Adaline". Os parâmetros de aprendizagem são ajustados de modo a minimizar o erro na estimação tornando a técnica imune à ruídos e distorções harmônicas.

2.6.3 Lógica Fuzzy

A LF está presente em consolidadas áreas de pesquisa, como por exemplo, em sistemas de controle e supervisão, pois, não há a necessidade de realizar a modelagem matemática do processo no qual a técnica será aplicada(Barbosa et al., 2006). Nos sistemas elétricos de potência, a lógica *Fuzzy* pode ser observada na proteção diferencial dos transformadores de potência e no monitoramento do comportamento da freqüência (Barbosa et al., 2006).

O trabalho desenvolvido por Soliman et al. (2003) considera as amostras digitalizadas das tensões do SEP como números *Fuzzy* para otimizar a estimação da freqüência. Segundo os

autores, a operação do algoritmo independe do grau de *fuzzificação*, porém a taxa de amostragem dos sinais tem uma considerável influência no seu funcionamento.

2.6.4 Considerações

O principal intuito de fazer a revisão bibliográfica sobre o tema da estimação de freqüência foi apurar quais as principais técnicas estudas por pesquisadores e a evolução delas durante as últimas décadas, pricipalmente devido ao uso de ferramentas computacionais cada vez mais poderosas. Além disso, a revisão bibliográfica mostrou quais as principais dificuldades encontradas pelos autores na implementação e validação dos algoritmos, tal como a imprecisão gerada por ruídos e pelas componentes harmônicas. No capítulo 3 será descrito o algoritmo MMQ adaptativo e a lógica implementada em linguagem de programação C que será convertida para VHDL.

Capítulo 3

Algoritmo de estimação da frequência usando o MMQ

Este capítulo apresenta o algoritmo adaptativo adotado para estimar a frequência dos sinais de tensão do SEP. O método numérico utilizado é o método dos mínimos quadrados (MMQ) na sua forma complexa. Cabe colocar que os filtros adaptativos têm a capacidade de se auto-projetarem através de um algoritmo recursivo em um ambiente no qual não se tem o conhecimento das características dos sinais presentes. Os algoritmos iniciam o processo a partir de uma condição inicial predeterminada. Em um ambiente estacionário, os parâmetros do filtro são atualizados a cada iteração até que a convergência atinja a solução ótima ou um ponto próximo a ela. Entretanto, se o sinal não for estacionário, o algoritmo irá rastrear as variações estatistícas para proporcionar a melhor solução, desde que tais variações sejam suficientemente lentas (Haykin, 2002).

3.1 Método dos mínimos quadrados - MMQ

O método dos mínimos quadrados adaptativo foi originalmente desenvolvido por Widrow e Hoff (1960). O MMQ é um membro da família dos algoritmos de gradiente estocástico e é amplamente empregado nas mais diversas aplições devido a sua simplicidade e robustez. Uma das vantagens de sua utilização é a sua baixa complexidade computacional, pois não requer o cálculo de matrizes inversas e de funções de correlação (Haykin, 2002). A figura 3.1 apresenta o diagrama de blocos do algoritmo, onde $\hat{W}(\mathrm{n}) = [w_1\ w_2\ w_3\ \ldots\ w_{M-1}]^T$ são os coeficientes, $\mathbf{U}(\mathrm{n}) = [u(n)\ u(n-1)\ u(n-2)\ u(n-3)\ \ldots\ u(n-M+1)]$ representa as amostras do sinal de entrada

no tempo n e M é o tamanho do filtro.

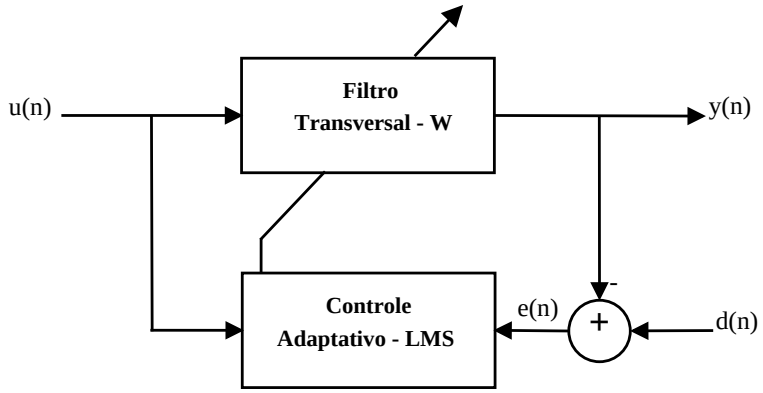

Figura 3.1: Diagrama de blocos do MMQ.

O MMQ atua no rastreamento do ponto de mínimo (ótimo) em uma superfície de desempenho descrita pelo quadrado do erro estimado $|e(n)|^2$, através do ajuste recursivo dos coeficientes do filtro. Tais coeficientes podem ser calculados de acordo com a equação 3.1 :

$$\mathbf{W}(n+1) = \mathbf{W}(n) - \mu \nabla e^2(n) \tag{3.1}$$

Nesta equação, $\mathbf{w}$(n+1) representa o vetor dos coeficientes atualizados, μ é o parâmetro que controla a convergência e a velocidade do algoritmo e $\nabla e^2(n)$ é o operador gradiente de desempenho do erro. Para estimar o valor de ∇e^2(n) utiliza-se a aproximação mostrada na equação 3.2 (Farhang-Boroujeny, 1999):

$$\nabla e^2(n) = -2e(n)\mathbf{u}(n) \tag{3.2}$$

Assim, a equação 3.1 pode ser reescrita como:

$$\mathbf{W}(n+1) = \mathbf{W}(n) + 2\mu e(n)\mathbf{U}(n) \tag{3.3}$$

O MMQ apresenta grande sensibilidade quanto a variação de μ, sendo que quando este possui um alto valor, o algoritmo leva menos tempo para alcançar a convergência, porém apresenta maior oscilação em torno do ponto ótimo. Para garantir a estabilidade do algoritmo, esse parâmetro deve satisfazer a seguinte condição (Haykin, 2002) :

$$0 < \mu < \frac{2}{MS_{max}} \tag{3.4}$$

onde M é o tamanho do filtro e S_{max} é máxima densidade de potência espectral da entrada $\mathbf{u}$(n).

3.2 Algoritmo de estimação da frequência

A freqüência é um importante parâmetro no monitoramento, no controle e na proteção dos sistemas de potência. Desse modo, diversos métodos numéricos foram propostos e desenvolvidos na literatura com o objetivo de analisar o comportamento dinâmico da freqüência, uma vez que a partir da mesma pode-se extrair várias informações a respeito do estado do SEP frente às mais diversas situações de operação.

Diante deste contexto, uma das alternativas apresentadas para a estimação da freqüência é a utilização de um filtro adaptativo baseada no algoritmo dos mínimos quadrados. O procedimento da filtragem digital é realizado para as tensões trifásicas obtidas a partir da rede elétrica que podem ser representadas como :

$$V_a(t) = V_{max}\cos(\omega t + \phi) \tag{3.5}$$

$$V_b(t) = V_{max}\cos(\omega t + \phi - \frac{2\pi}{3}) \tag{3.6}$$

$$V_c(t) = V_{max}\cos(\omega t + \phi + \frac{2\pi}{3}) \tag{3.7}$$

Nas equações V_{max} é a aplitude de pico da onda, ϕ é a fase do sinal e ω[1] é a freqüência angular. Mas como se trata de uma simulação numérica, as tensões retratadas nas equações podem ser consideradas discretas no tempo, porém com uma alta taxa de amostragem de modo que não se perca a fidelidade aos sinais analógicos presentes no sistema elétrico (Barbosa, 2007).

$$V_a(n) = V_{max}\cos(\omega n\Delta T + \phi) \tag{3.8}$$

$$V_b(n) = V_{max}\cos(\omega n\Delta T + \phi - \frac{2\pi}{3}) \tag{3.9}$$

[1] $\omega = 2\pi\ f_s$, onde f_s é a freqüência do sistema, dada em hertz (Hz)

$$V_c(n) = V_{max}\cos(\omega n\Delta T + \phi + \frac{2\pi}{3}) \tag{3.10}$$

onde n é o número da amostra e ΔT é a taxa de amostragem.

Os dados são obtidos a partir de um arquivo "*texto*", e a leitura realizada de forma janelada, com o tamanho de um ciclo de informação e com passo/deslocamento fixo de uma amostra. Os dados adquiridos são submetidos a um filtro passa-baixa para reduzir a participação de componentes de alta freqüência e, em seguida, a um conversor analógico-digital (CAD) no qual ocorre uma reamostragem dos sinais trifásicos.

Em seguida, ocorre a normalização das tensões digitalizadas, aonde as mesmas são padronizadas. Cada vez que esta etapa é executada, ocorre um avanço (deslocamento) da janela de dados, onde o próximo dado é lido e o primeiro da janela anterior é descartado, de modo que seja constistuída uma nova janela para as análises posteriores.

Os dados normalizados são convertidos para uma representação complexa utilizando a transformada $\alpha\beta$ (Akke, 1997), aonde os cálculos são realizados para cada janela de dados apresentada. A forma de onda resultante desta transformação é usada como entrada para o filtro baseado no MMQ com parâmetro de convergência variável.

A filtragem adaptativa é realizada para toda a janela, sendo que a freqüência do sistema é estimada através do ajuste dos coeficientes até que o sinal de saída seja bem próximo ao sinal de entrada complexo, considerando como critério de parada, o erro absoluto e o número de iterações. Além disso, também é empregado um filtro que analisa a variação da freqüência no tempo com o intuito de reduzir as oscilações da mesma.

3.3 Lógica computacional implementada

Nesta seção serão descritos em detalhe os procedimentos adotados para o desenvolvimeto do algoritmo utilizado na estimação da freqüência do sistema elétrico em linguagem *C* (Barbosa, 2007) e que será tomado como base para a conversão na linguagem de programação VHDL.

3.3.1 Fluxograma

Na figura 3.2 está representado o fluxograma da estrutura do programa, no qual se evidencia os processos de aquisição dos dados, normalização, a transformação $\alpha\beta$, o filtro adaptativo, a filtragem do sinal de saída, a estimação da freqüência e o critério de parada.

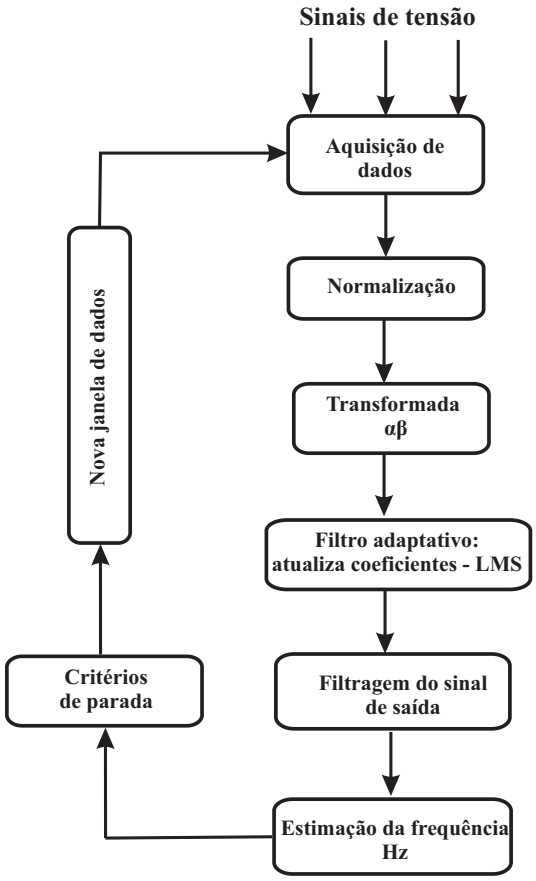

Figura 3.2: Fluxograma do algoritmo de estimação da freqüência.

3.3.2 Aquisição dos dados e digitalização

A primeira parte consiste na aquisição e no pré-processamento dos dados a serem usados como os valores de referência para a continuidade dos estudos. A figura 3.3 mostra o diagrama que representa esta etapa inicial.

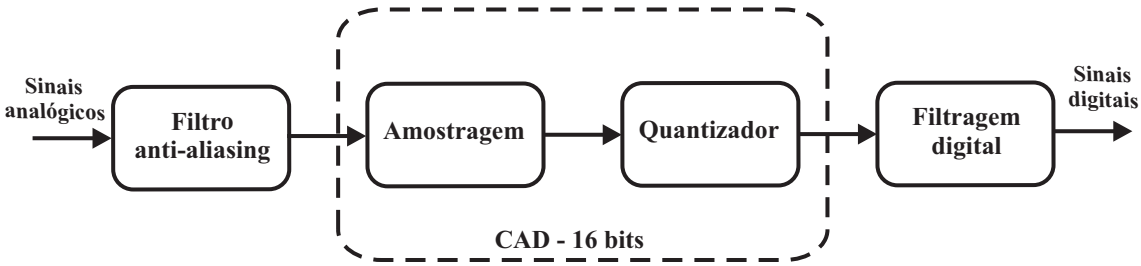

Figura 3.3: Diagrama de aquisição dos dados.

As tensões elétricas aplicadas na entrada, como retratado no fluxograma da seção anterior, são obtidas via simulação computacional com alta taxa de amostragem de modo que seja mantida a fidelidade com relação aos sinais analógicos presentes na rede elétrica, como representado nas

equações 3.8, 3.9 e 3.10. Desta maneira, tais tensões com alta freqüência de amostragem podem ser consideradas uma fiel aproximação aos valores contínuos na implementação do algoritmo.

Os sinais adquiridos são direcionados a um filtro *anti-aliasing* modelado de acordo com a aproximação de *Butterworth* de segunda ordem e freqüência de corte igual á 200 Hz, de modo que as componentes harmônicas presentes em tais sinais sejam removidas.

Após essa filtragem ocorre o processo de digitalização, aonde é empregado um conversor analógico digital (CAD) de 16 bits e uma freqüência de amostragem de 1920 Hz. Nesta etapa, também é utilizado um quantizador para verificar se as saídas ultrapassam a capacidade do CAD. Na realidade este processo pode ser visto como uma reamostragem das tensões trifásicas aplicadas na entrada do algoritmo. Os sinais resultantes podem ser representados pelas equações 3.11, 3.12 e 3.13.

$$V_a(n) = V_{max}\cos(\omega n\Delta T_a + \phi) + \xi_{n_a} \tag{3.11}$$

$$V_b(n) = V_{max}\cos(\omega n\Delta T_a + \phi - \frac{2\pi}{3}) + \xi_{n_b} \tag{3.12}$$

$$V_c(n) = V_{max}\cos(\omega n\Delta T_a + \phi + \frac{2\pi}{3}) + \xi_{n_c} \tag{3.13}$$

onde $\mathbf{V}_{a,b,c}$ se referem ás tensões da amostra n, ΔT_a o tempo de amostragem do conversor analógico-digital e ξ o ruído inserido após a digitalização.

Em seguida, ocorre uma filtragem de estabilização com o intuito de reduzir a participação dos ruídos introduzidos pela digitalização, no qual é empregado novamente um filtro *Butterworth* de segunda ordem com uma freqüência de corte igual á 200 Hz.

3.3.3 Normalização

Esta etapa consiste na padronização dos dados para qualquer nível de tensão analisado. Desta maneira, os efeitos gerados por possíveis distúrbios, como por exemplo, um afundamento ou pela elevação em uma das fases do SEP serão reduzidos de forma que a performance do algoritmo seja mantida na seqüência do processamento. Cabe frisar que a normalização não altera a freqüência dos dados coletados, uma vez que apenas a amplitude da forma de onda está sendo modificada.

3.3.4 Transformada $\alpha\beta$

A transformada $\alpha\beta$ é utilizada para converter os sinais das tensões trifásicas em um fasor complexo, onde a parte real e a imaginária possuem uma defasagem de 90°(Akke, 1997). O

objetivo da aplicação desta transformada é o desacoplamento das três fases, tendo como resultado três circuitos monofásicos e independentes.

O sinal complexo, obtido após essa transformação, é usado como entrada da etapa de filtragem digital do algoritmo, que servirá como base para a estimação da freqüência do sistema. Tal sinal possui dois modos, chamados de componentes de *Clarke*, que podem ser calculadas como mostrada na Eq. 3.14 :

$$\mathbf{M_T} = \sqrt{\frac{2}{3}} \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \tag{3.14}$$

$$\begin{pmatrix} \mathbf{V}_\alpha \\ \mathbf{V}_\beta \end{pmatrix} = \mathbf{M_T} \times \begin{pmatrix} \mathbf{V}_a \\ \mathbf{V}_b \\ \mathbf{V}_c \end{pmatrix} \tag{3.15}$$

Nesta $\mathbf{V}_{\alpha,\beta}(n)$ representam os vetores modais e $\mathbf{V}_{a,b,c}$ são os vetores das fases da tensão. Após a transformação, a tensão complexa pode ser representada de acordo com a equação 3.16:

$$\mathbf{U}(n) = \mathbf{V}_\alpha(n) + j\mathbf{V}_\beta(n) \tag{3.16}$$

onde $\mathbf{U}(\mathrm{n}) = [u(n)\ u(n-1)\ u(n-2)\ u(n-3)\ \ldots\ u(n-N+1)]$ e N é o tamanho da janela de dados.

3.3.5 Filtro adaptativo

A filtragem adaptativa corresponde a etapa mais importante da implementação do algoritmo de estimação da freqüência, pois, é nela que o sinal é efetivamente processado, através do ajuste recursivo dos coeficientes, proporcionando a redução do erro médio quadrático. Para isso é utilizado o MMQ na sua forma complexa inicialmente desenvolvido por Widrow et al. (1975). A estrutura do filtro proposto para o desenvolvimento desta técnica é mostrada na figura 3.4.

O sinal de erro $e(n)$ é definido como a diferença entre o sinal desejado, representado pela tensão complexa de entrada, e a atual saída do filtro, como mostrado na equação 3.17:

$$e(n) = u(n) - y(n) \tag{3.17}$$

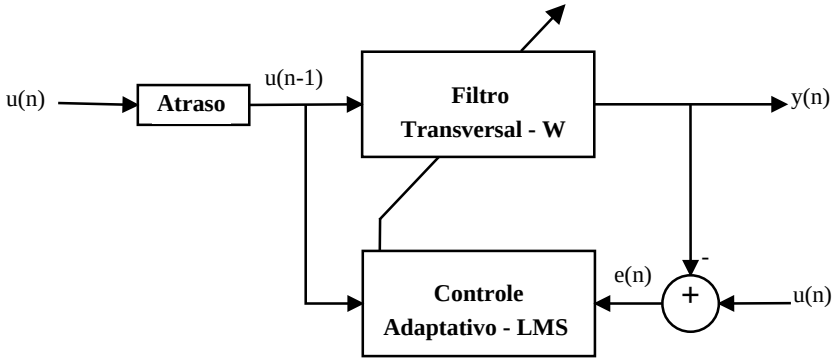

Figura 3.4: Diagrama do filtro proposto

A saída no tempo da amostra n pode ser calculado como :

$$y(n) = \mathbf{W}^H \mathbf{U}(n-1) \tag{3.18}$$

onde H é a transformada de *Hilbert* (Haykin, 2002).

A cada iteração ocorre a atualização dos coeficientes do filtro de modo que ocorra a minimização do erro médio quadrático, o que resulta na equivalência entre sinal de saída do filtro e a entrada complexa de referência. Tais coeficientes são encontrados de acordo com a expressão 3.19 (Haykin, 2002):

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \mu(n) e^*(n) \mathbf{U}(n-1) \tag{3.19}$$

Nesta e^* representa o complexo conjugado do erro e μ é o fator que controla a estabilidade e a convergência do algoritmo.

Para proporcionar uma melhora na velocidade da busca pelo conjunto de coeficientes calculados por 3.19, adotou-se um parâmetro de convergência de tamanho variável. Ou seja, a cada iteração, o fator μ é alterado dentro de limites pré-estabelecidos, como sugerido por Aboulnasr e Mayyas (1997) :

$$\mu(n+1) = \lambda \mu(n) + \gamma p(n) p^*(n) \tag{3.20}$$

na qual p é a autocorrelação dos erros conseguintes e(n) e e(n-1), e calculada por :

$$p(n+1) = \rho p(n) + (1-\rho)e(n)e(n-1) \tag{3.21}$$

Nas equações 3.20 e 3.21, as constantes ρ, γ e λ são determinadas por estudos estatísticos, podendo variar entre 0 e 1.

3.3.6 Estimação da freqüência

A freqüência é estimada através da diferença de fase entre duas amostras consecutivas da saída, como mostrado na equação 3.22 :

$$\Gamma = y(n)y^*(n-1) \tag{3.22}$$

Desse modo, a freqüência do sistema é calculada como na expressão 3.23

$$f_{est} = \frac{f_s}{2\pi} \arctan(\frac{\Im(\Gamma)}{\Re(\Gamma)}) \tag{3.23}$$

Na qual f_s é a freqüência de amostragem e $\Im()$ e $\Re()$ são as partes imaginária e real, respectivamente.

3.3.7 Filtragem do sinal de saída

Após a estimação do comportamento da freqüência, foi utilizado um filtro *Butterworth* do tipo passa-baixa de segunda ordem, com frequência de corte de 5 Hz, para minimizar os efeitos dos ruídos inseridos durante a execução do algoritmo.

3.3.8 Critério de parada

Para os critérios de parada desta técnica foram levados em consideração o tempo em que o algoritmo pode processar continuamente a informação e o erro satisfatório referente a condição de convergência. Assim, adotou-se um número máximo de 1.000 iterações e/ou um erro absoluto inferior á 0,1% calculado da seguinte maneira :

$$e_{rel} = abs(\frac{y(n) - u(n)}{u(n)})100\% \tag{3.24}$$

onde e_{rel} é o erro percentual e abs o valor absoluto.

Capítulo 4

Linguagem de Programação VHDL

4.1 Conceitos sobre VHDL

Os sistemas eletrônicos evoluiram rapidamente durante as últimas décadas. Os primeiros circuitos integrados apresentavam pequena escala de integração, sendo constituídos com pouco mais de cem transistores. No surgimento da tecnologia LSI (*Large Scale Integration*), os projetistas já eram capazes de empregar milhares de portas em um único *chip*, gerando um certo nível de complexidade na organização dos sistemas.

Com o advento da tecnologia VLSI (*Very Large Scale Integration*), os circuitos eletrônicos puderam ser desenvolvidos aplicando mais de cem mil transistores, tornando-os altamente complexos e, como conseqüência, a indústria se viu na necessidade de abandonar o uso da representação por esquemáticos na confecção de seus projetos. Desse modo, os projetistas sentiram a necessidade de uma linguagem de programação que fosse capaz de descrever o funcionamento de tais circuitos sem precisar verificá-los manualmente (Palnitkar, 2003).

Assim a criação de uma linguagem de descrição de *hardware* era uma decorrência natural. Nas décadas de 70 e 80, o Departamento de Defesa (DoD - *Department of Defense*) Americano pôs em prática um programa para o desenvolvimento de circuitos integrados de altissíma velocidade, denominado VHSIC (*Very High Speed Integrated Circuit*), para fins militares. Ao mesmo tempo, foi criada a linguagem VHDL (*VHSIC Hardware Description Language*) para descrever o comportamento, a estrutura e a implementação desses circuitos (Naylor e Jones, 1997). Simultaneamente, outras linguagens de descrição de *hardware* surgiram no ramo industrial, como a Verilog HDL.

Após o seu aparecimento, a linguagem VHDL tornou-se muito popular, sendo amplamente utilizada na indústria e entre os pesquisadores. Segundo relatos, em 1987 a linguagem VHDL foi padronizada pelo IEEE (*Institute of Electrical and Electronic Engineers*) segundo a norma *IEEE Standard 1076*, sendo conhecida como VHDL-87. Como todos os padrões, a linguagem VHDL é submetida á uma revisão a cada cinco anos, no qual sugestões dos usuários são analisadas por uma comissão responsável. Assim, em 1992 a versão 87 foi revisada originando o VHDL-93. Em 1998, uma segunda revisão foi realizada dando origem ao VHDL-2001 (Ashenden, 2002). Recentemente foi lançada a atual variante da linguagem corresponde a VHDL-2008 (IEEE, 2009).

Basicamente, o VHDL se diferencia de outras linguagens seqüênciais, como C/C++ e Pascal, através de duas características: o paralelismo entre as rotinas e a capacidade de empregar explicitamente declarações de atraso de tempo. A primeira propriedade faz com que não haja um padrão para o fluxo de controle das instruções, ou seja, elas podem ser executadas todas concorrentemente, ao menos que se especifique a ocorrência das mesmas. Além disso, é preciso salientar que em VHDL somente as declarações inseridas dentro de processos, funções ou procedimentos são executadas seqüencialmente. Já o uso de rotinas de atraso permite o agendamento de uma determinada tarefa em relação ao tempo de simulação, podendo se aproximar de uma situação real.

Em VHDL existem três tipos de objetos: constantes, variáveis e sinais (Perry, 2002). Objetos do tipo constante não podem ser modificados e são usados para determinar explicitamente um nome e um tipo para um determinado valor. Uma variável pode ser alterada quantas vezes for necessário e tal alteração é realizada imediatamente. Por outro lado, os sinais só são atualizados quando os processos nos quais estão inseridos é suspenso, ou quando tal mudança é mencionada por rotinas de agendamento. Além disso, a linguagem VHDL lida com uma ampla faixa de tipos de dados que são usados para especificar as características dos objetos descritos. Os tipos de dados disponíveis são classificados basicamente em: escalares (ponto flutuante, inteiros, complexos, binários, booleanos, níveis lógicos, físicos e outros), compostos (vetores e estrururas), ponteiros e arquivos (Ashenden, 2002).

A estrutura de um código em VHDL é composto por quatro partes, como mostrado na Figura 4.1. A primeira parte, *package* (pacote), corresponde a declaração das bibliotecas a serem utilizadas no código, no qual podem ser inseridas as constantes, os tipos de dados e os subprogramas. Na *entity* (entidade) é especificada a interface do projeto, ou seja, os pinos de entrada e saída e os seus respectivos tipos. Na *architecture* (arquitetura) são implementadas

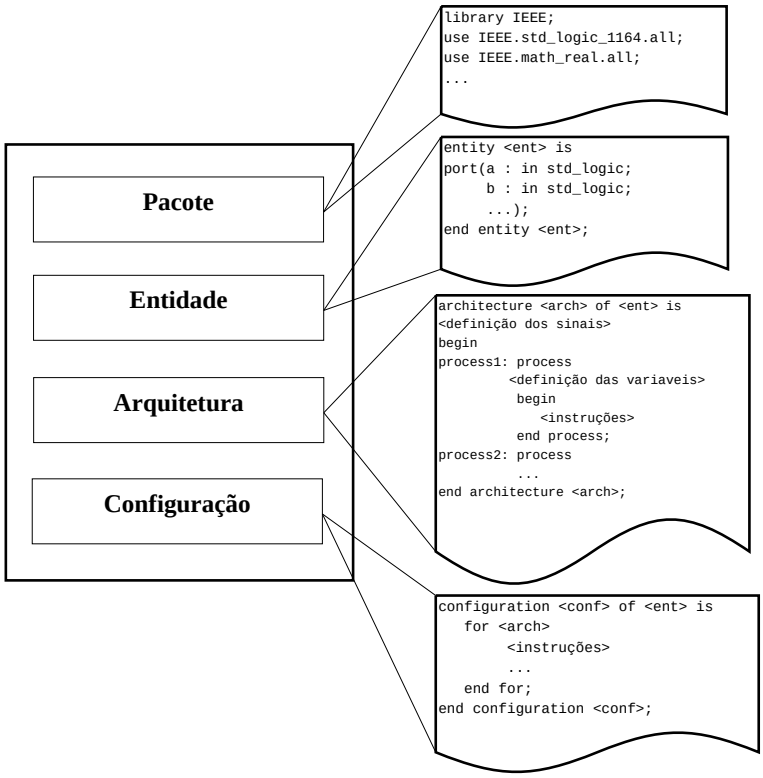

Figura 4.1: Estrutura de um código em VHDL.

as funções a serem desempenhadas pelo projeto. Uma entidade pode possuir uma ou mais arquiteturas, no qual pode haver vários processos. Dentro de cada processo é executado um programa seqüêncial que pode conter atribuição de variáveis, *loops* e chamada de funções e procedimentos. Por fim, na quarta parte, a *configuration* (configuração) determina qual das arquiteturas será utilizada pela entidade. Após a execução do código o resultado é chamado de RTL (*Register-Transfer Language*), no qual se definem os registros e a lógica combinacional entre entradas e saídas.

Um código em VHDL pode ser escrito usando basicamente duas formas de modelagem (descrição): a estrutural e a comportamental. Na descrição comportamental é exposta a funcionalidade do sistema. Nessa abordagem o programa possui o mesmo formato de um algoritmo implementado em uma linguagem de programação como C/C++, e não necessita do conhecimento da estrutura em *hardware*. A descrição estrutural especifica a organização física do sistema, no qual são caracterizados os componentes que compõem o projeto em questão, a interconexão entre eles e os sinais utilizados. Em VHDL são disponíveis bibliotecas que contêm entidades referentes as memórias, decodificadores, contadores e outros.

O VHDL oferece as seguintes vantagens na confecção de projetos (Chang, 1997):

- Padronização: reduz confusões e facilita interfaces entre ferramentas, fornecedores e produtos diferentes;
- Suporte da indústria: aumento do compromisso da indústria devido ao lançamento de ferramentas mais potentes em VHDL;
- Portabilidade: o mesmo código em VHDL pode ser simulado e usado com ferramentas de diversos fornecedores e em diferentes estágios de desenvolvimento;
- Reusabilidade: a linguagem VHDL permite a criação de bibliotecas com códigos parametrizados desenvolvidos para o futuro;
- Independência da tecnologia: a descrição comportamental e estrutural do projeto pode ser criada em VHDL independentemente da tecnologia a ser usada;
- Documentação: VHDL é uma linguagem de alto nível cuja codificação pode ser considerada como uma documentação do projeto;
- Novas metodologias de desenvolvimento: com o uso do VHDL é possível criar uma nova metodologia de modo a aumentar a produção e diminuir o ciclo de desenvolvimento.

Cabe frisar que os códigos em VHDL são amplamente empregados em componentes de lógica programável. Embora inicialmente a linguagem VHDL tenha sido desenvolvida para a descrição de sistemas eletrônicos, ela também pode ser utilizada para descrever redes de computadores, filtros digitais, sitemas inteligentes, dentre outras aplicações.

4.2 ModelSim

Para a implementação do algoritmo de estimação de freqüência descrito no capítulo 3 em VHDL, foi utilizado o *software* ModelSim™(MentorGraphics, 2009). O ModelSim permite a simulação e a verificação de códigos em VHDL, Verilog, SystemVerilog e SystemC. O fluxograma apresentado na Figura 4.2 ilustra as etapas básicas para o desenvolvimento de um projeto no ModelSim.

- Criação de novo projeto: Esta primeira etapa consiste em nomear o projeto e a biblioteca de trabalho, que normalmente é definida pelo nome padrão *"work"*. Todas as entidades e arquiteturas utilizadas nos códigos são definidas nesta biblioteca.

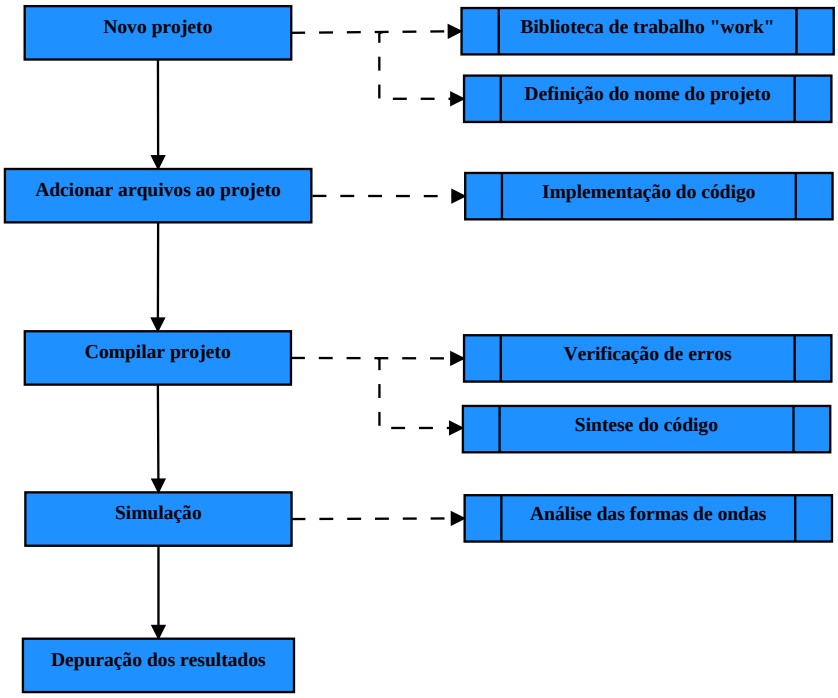

Figura 4.2: Fluxograma básico de projeto no ModelSim.

- Adicionar arquivos ao projeto: Elaboração dos códigos que irão compor a estrutura computacional do projeto. Tais códigos irão compor uma estrutura de hierarquia no projeto.

- Compilar projeto: O compilador analisa todos os arquivos definidos na biblioteca de trabalho, verificando a existência de possíveis erros quanto à sintaxe e semântica da linguagem. Caso nenhum erro for encontrado, é realizada a síntese do código em termos das entradas e saídas.

- Simulação: Após a compilação, a simulação do projeto é feita especificando, primeiramente, a entidade que representa o mais alto nível hieráquico do projeto. O ModelSim disponibiliza uma análise da evolução dos sinais no tempo, permitindo a verificação dos resultados da simulação através das formas de ondas.

- Depuração dos resultados: o ModelSim possui a ferramenta de depuração, no qual o uso de *breakpoints* possibilita a análise da execução de cada instrução.

4.3 Estrutura computacional do algoritmo proposto em VHDL

A implementação do algoritmo apresentado no capitulo 3 foi executada utilizando o padrão IEEE Standard 1076-1993 (IEEE, 1993). Uma listagem do programa desenvolvido encontra-se

no Anexo A. A conversão de tal algoritmo da linguagem de programção C para VHDL visa a otimização da técnica por meio do uso de estruturas concorrentes de modo que o tempo de processamento em *hardware* seja minimizado. A Figura 4.3 ilustra os estágios para a conversão da técnica proposta em VHDL.

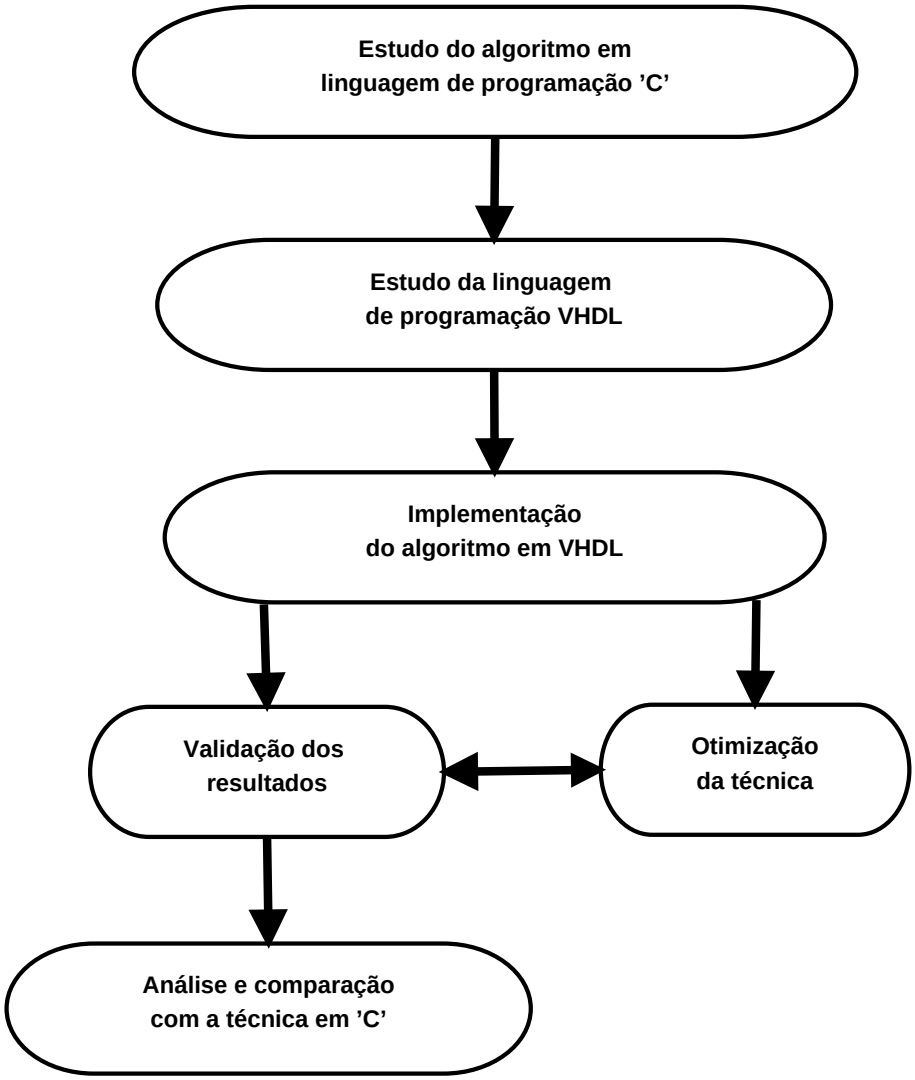

Figura 4.3: Metodologia usada para implementar o algoritmo em VHDL.

A Figura 4.4 mostra a maneira em que o algoritmo estimador da freqüência foi implementado, tendo em vista a sua decomposição em sub-sistemas menores, aonde são enfatizados o *test bench*, o condicionamento dos sinais trifásicos, a transformada $\alpha\beta$ e o filtro adaptativo.

A entidade que representa o algoritmo possui como entradas um sinal de habilitação, as tensões trifásicas e a quantidade de amostras geradas no *test bench*. As portas de saídas são a freqüência estimada e a sua derivada no tempo). No corpo da arquitetura desta entidade é executado o laço principal do código, que se refere ao condicionamento dos sinais, a movimen-

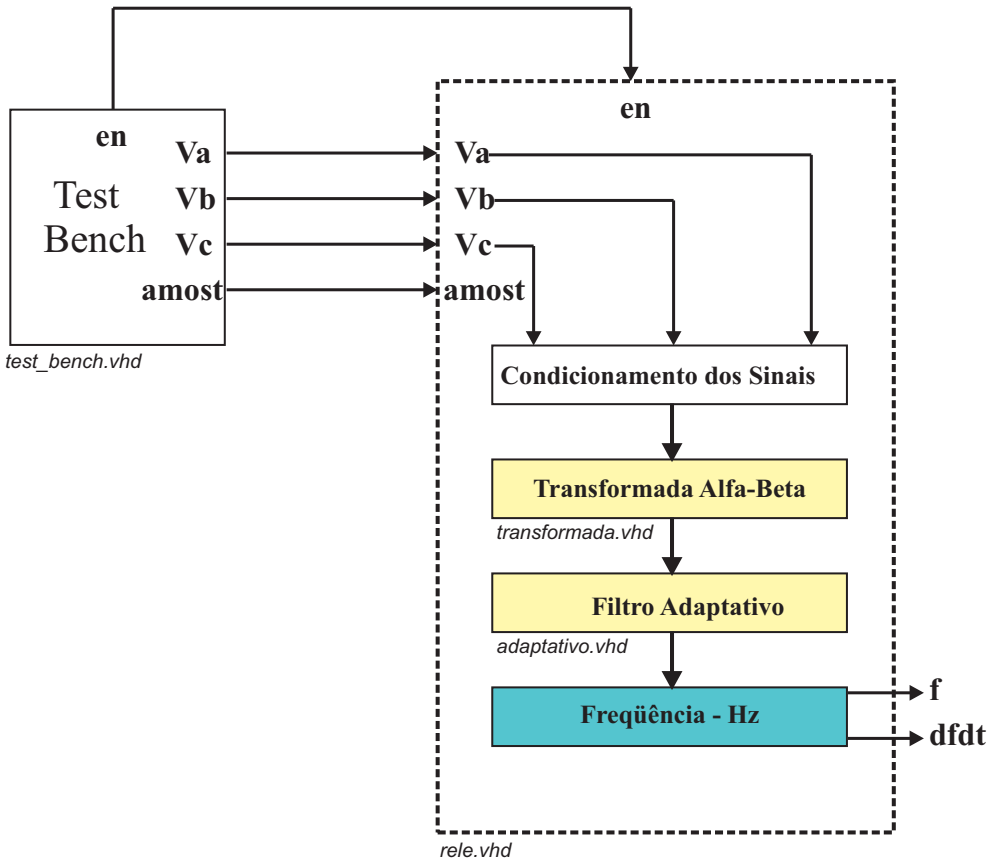

Figura 4.4: Composição em sub-sistemas do algoritmo de estimação da freqüência.

tação da janela de dados e as habilitações/desabilitações para a transformada $\alpha\beta$ e para o filtro adaptativo.

4.3.1 Bibliotecas implementadas

As definições de protótipos das funções/procedimentos e tipos de variáveis que são usados especificamente no projeto foram declaradas em bibliotecas de tal forma que todos os módulos do projeto tenham acesso aos mesmos.

Na biblioteca principal são declarados os componentes e as variáveis (tais como aquela que armazena os dados das tensões trifásicas) que são usados na elaboração do projeto.

As outras bibliotecas se referem às definições do filtro *Butterworth*, ao conversor analógico-digital e á normalização.

4.3.2 *Test bench*

Em VHDL, pode-se simular usando uma estrutura chamada de *test bench*. Tal estrutura pode ser considerada um gerador de sinais, aonde são agendados eventos aos sinais de entrada do projeto. O código de um *test bench* consiste de um corpo de arquitetura contendo o componente a ser testado e os processos que gerarão a seqüência de estímulos nas entradas de tal componente.

Na implementação do código em VHDL, o *test bench* foi configurado de modo a iniciar o funcionamento do componente sob teste, que neste caso é a entidade que representa o algoritmo estimador da freqüência. Em seguida, é repassado o arquivo de dados que possui os valores das tensões trifásicas ($\mathbf{V}_{a,b,c}$) que são os sinais aplicados na entrada do projeto.

4.3.3 Condicionamento dos sinais

O condicioanemto dos sinais foi realizado similarmente como descrito na Seção 3.2 do Capítulo 3. Os dados trifásicos gerados no *test bench* são registrados para as etapas da filtragem *anti aliasing*, digitalização e normalização. Este estágio visa a representação dos sinais para que estes sejam processados de modo coerente nas etapas posteriores, buscando à aproximação para um relé comercial.

4.3.4 Transformada $\alpha\beta$

Após o condicionamento dos sinais trifásicos, ocorre a conversão para a forma complexa do problema, aonde são calculadas as componentes α e β da tensão complexa. De acordo com a sua formulação, tal tensão é da forma de um vetor com tamanho igual a de uma janela de dados. No código em VHDL, os elementos dos vetores $\mathbf{V}_\alpha(n)$ e $\mathbf{V}_\beta(n)$ são obtidos de forma paralela, como mostrado na Figura 4.5.

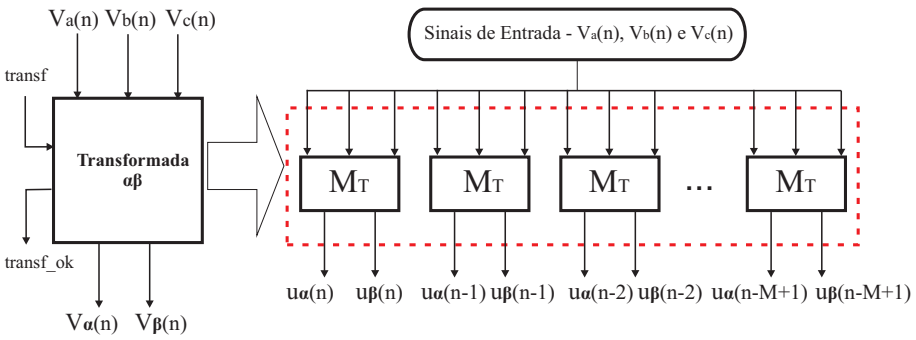

Figura 4.5: Composição estrutural da Transformada $\alpha\beta$.

Tal modelagem foi realizada aplicando a geração de estruturas iterativas. Esta ferramenta pode ser empregada quando é necessário replicar um determinado sub-sistema, sendo que cada um opera de maneira concorrente. A entidade que representa o módulo da Transformada $\alpha\beta$ possui como entradas as tensões trifásicas ($\mathbf{V}_{a,b,c}$) e o sinal $transf$, e as saídas definidas como as componentes α e β e o sinal $transfok$.

O sinal $transf$ indica que uma nova janela de dados, proveniente da normalização, está pronta para ser processada, implicando na habilitação dos cálculos realizados nesta etapa. Tais cálculos se referem a aplicação da matriz de transformação (M_T) nos sinais trifásicos de modo a obter a tensão complexa. Em seguida, a saída $transfok$ indica que a tensão complexa foi computada para ser aplicada no filtro adaptativo.

4.3.5 Filtro adaptativo

Este módulo representa a etapa mais importante do algoritmo proposto, aonde é retornada a freqüência calculada para uma janela fixa de dados. A entidade possui como pinos de entrada um sinal de habilitação do filtro adaptativo e as componentes α e β. As saídas são a freqüência estimada e o sinal de indicação que o filtro atingiu a convergência.

O corpo da arquitetura possui um laço (processo) principal aonde são verificados os critérios de parada em relação a convergência do algoritmo e, também, a estimação da freqüência, o cálculo do erro, as habilitações para a atualização dos coeficientes e para a obtenção da saída. Além disso, foram implementados outros processos, em que são computados os coeficientes e a saída do filtro. Na Figura 4.6 é mostrada a divisão da arquitetura em termos dos processos citados.

A atualização dos coeficientes do filtro, que corresponde ao *processo adaptativo*, foi realizada de forma similar ao cálculo das componentes α e β descrito na seção 4.3.4, ou seja, aplicando o uso da geração de estruturas iterativas, no qual cada estrutura executa os cálculos dos novos valores destes coeficientes pela Equação 3.19.

Em seguida, os coeficientes são direcionados a um combinador linear que foi dividido em duas partes. A primeira realiza a multiplicação entre as entradas e os coeficientes, sendo que estas multiplicações são feitas paralelamente por meio do uso da geração de estruturas iterativas. A outra representa um acumulador de modo que a saída seja calculada pela soma das multiplicações obtidas anteriormente.

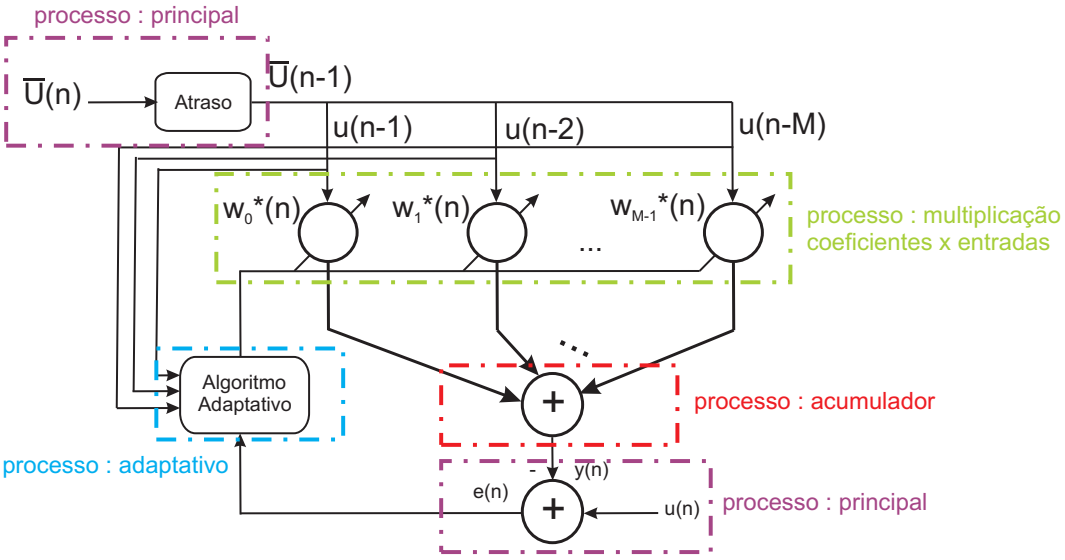

Figura 4.6: Composição estrutural do filtro adaptativo.

4.3.6 Sinais de freqüência

Com o intuito de reduzir a característica oscilatória proveniente da técnica utilizada e em função dos transitórios observados nas tensões de entrada, o sinal de freqüência resultante do MMQ é filtrado por um filtro digital passa-baixa com aproximação *Butterworth* de segunda ordem e freqüencia de corte de $5Hz$.

Capítulo 5

Field Programmable Gate Array - FPGA

A tecnologia *Field Programmable Gate Array* (FPGA) surgiu no início da década de 80 com o intuito de preencher uma lacuna entre os dispositivos lógicos programáveis (PLDs - *Programmable Logic Devices*), que não suportavam funções altamente complexas, e os circuitos integrados de aplicação especifíca (ASICs - *Application-Specific Integrated Circuits*), que por sua vez podiam implementar funções com alto grau de complexidade, porém, possuiam um alto custo (Maxfield, 2004). Uma das vantagens desta tecnologia vem da capacidade de poder ser programada e testada rapidamente, sem que haja a necessidade de re-manufaturar fisicamente o dispositivo. Além disso, um mesmo *chip* de FPGA pode ser utilizado em vários projetos, reduzindo o custo de inventário.

A principal característica do FPGA consiste no fato que a funcionalidade do seu núcleo pode ser programada (ou re-programada) exclusivamente pelo usuário final e no campo de aplicação, em oposição aos dispositivos que são configurados pelos fabricantes. Na indústria, a estrutura de um FPGA varia de acordo com o fabricante, porém a arquitetura geral contém três partes principais, como mostrado na figura 5.1 : blocos lógicos configuráveis (CLB - *Configurable Logic Blocks*), blocos de entrada/saída (IOB - *Input/Output Block*) e recursos de interconexão.

Um FPGA não utiliza portas *OR* e *AND* em seu interior. Ao invés destas, são empregados blocos lógicos que podem ser configurados para implementar uma determinada função. Cada CLB é constituído por células chamadas de elementos lógicos (LE - *Logic Element*), nos quais podem ser formados, em sua maioria, por *lookup tables* (LUT), *flip-flops* ou *latches* e multiple-

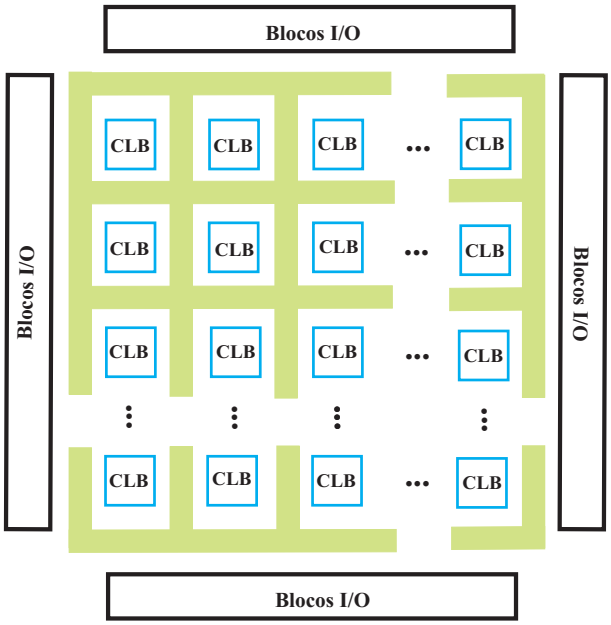

Figura 5.1: Arquitetura geral de um FPGA.

xadores. Na figura 5.2 está representado o esquema básico de um LE com três entradas.

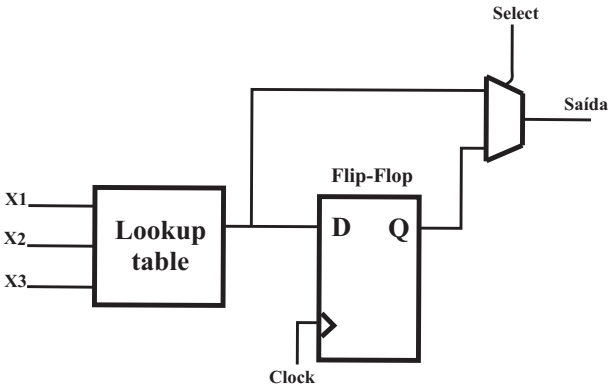

Figura 5.2: Elemento lógico.
(Brown e Vranesic, 2005)

Uma LUT é um arranjo de células de memória SRAM (*Static-Random Allocation Memory*) que são usadas para implementar uma tabela verdade. Cada endereço da SRAM representa uma combinação dos sinais de entradas do LE, sendo que o valor armazenado neste endereço corresponde ao nível lógico (0 ou 1) para aquela combinação de entrada. Desse modo, uma função com n-entradas requer uma SRAM com 2^n locações. Já os *flip-flops* são combinados em forma de registradores e agem junto aos multiplexadores, pois o elemento da memória é conectado com a saída de acordo com o sinal de *clock* e o sinal de habilitação (Wolf, 2004). Os dispositivos que utilizam a LUT são normalmente baseados em SRAM, porém existem outras maneiras para implementar funções lógicas em FPGA como é o caso da tecnologia *anti-fuse*, na qual se emprega

um elemento de alta resistência que pode ser considerado um circuito aberto. Pela tecnologia *anti-fuse*, a programação é realizada somente uma única vez e de forma permanente. Para que tal elemento seja programado são aplicados pulsos de tensão relativamente altos de modo que o mesmo seja curto-circuitado (Maxfield, 2004). A figura 5.3 mostra a configuração de um CLB.

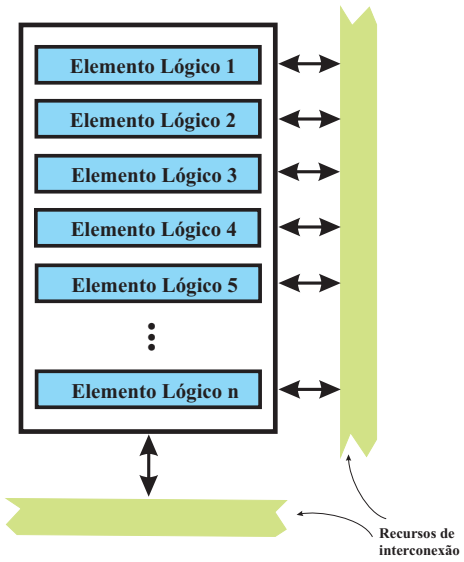

Figura 5.3: Esquema de um bloco lógico configurável - CLB.

Os recursos de interconexão são canais de roteamento organizados na horizontal e na vertical, entre os CLBs. Tais canais são formados por linhas (ou fios) e chaves programáveis que permitem conectar os CLBs que estão fisicamente longe sem induzir grande atraso de tempo. As chaves são geralmente do tipo CMOS e atrelam as linhas de comunicação entre si, além de realizar a conexão entre CLBs e IOBs. Já os fios são utilizados na transmissão de uma determinada informação. Tais fios podem ser organizados em diferentes categorias dependendo da estrutura e do tipo de aplicação (Wolf, 2004):

- **Fios curtos**: são aqueles que conectam somente LEs locais dentro de um mesmo bloco lógico.
- **Fios globais**: são especialmente projetados para comunicação de distância relativamente longa dentro do FPGA.
- **Fios especiais**: são dedicados somente para distribuir os sinais específicos como o de *clock* e de controle de registradores.

Os pinos de entrada e saída têm a função de realizar a interface entre o *chip* e o meio externo, aonde os sinais são inseridos para o interior do FPGA e enviados de volta.

Na implementação de um circuito lógico em FPGA, os CLBs são programados para realizar as funções requeridas e os canais de roteamento são configurados para fazer a interconexão entre os mesmos. Quando são utilizados blocos lógicos baseados em LUT, as células de memória SRAM possuem a propriedade de serem voláteis, ou seja, quando a alimentação do sistema é desligada o conteúdo armazenado é perdido. Desse modo, há a necessidade de reprogramar o FPGA. Freqüentemente é inserido um *chip* de memória, geralmente do tipo PROM (*Programmable Read Only Memory*), para reter esses dados permanentemente, fazendo com que o conteúdo das SRAMs seja recarregado automaticamente quando o sistema é energizado (Brown e Vranesic, 2005).

Dentre as aplicações nas quais o FPGA é frequentemente empregado pode-se citar o uso em protótipos de projetos em ASICs, em processamento digital de sinais, sistemas de navegação, controle de motores, em dispositivos aeroespaciais (satélites e veículos espaciais) e como plataforma hardware com o intuito de verificar a implementação física de algoritmos.

Para a continuidade do trabalho em específico, sugere-se implementar em FPGA o relé de freqüência que irá analisar e responder frente as possíveis operações desejáveis e não desejáveis sobre um SEP. Desta meneira, buscar-se-á por tal implementação viabilizar o desenvolvimento e a aplicação prática do relé via FPGA, permitindo-se compará-lo às técnicas comumente disponíveis nos relés comerciais.

Capítulo 6

Descrição do Sistema Elétrico de Potência Analisado

Este capítulo tem como finalidade descrever e comentar a modelagem do SEP utilizado para validar o método de estimação de freqüência apresentado neste trabalho. A ferramenta computacional empregada para modelar tal SEP foi o *software* ATP (*Alternative Transients Program*) que permite a simulação de condições normais e anormais de operação através da análise de transitórios eletromagnéticos em redes polifásicas (Leuven, 1987). No ATP é possível representar não linearidades, elementos com parâmetros concentrados e distribuídos, transformadores, chaves, geradores, motores elétricos e outros.

A Figura 6.1 mostra o diagrama unifilar do sistema elétrico utilizado neste trabalho, no qual se têm os principais elementos relacionados aos sub-sistemas de geração, transmissão e distribuição de energia elétrica. Tal sistema é composto por um gerador síncrono de 13,8 (kV) e potência aparente de 76 (MVA), um motor trifásico de 4,0 (kV) e potência ativa de 1180 (kW), transformadores trifásicos elevadores de 13,8:138 (kV) e 25 (MVA), transformadores trifásicos abaixadores de 138:13,8 (kV) e 25 (MVA), linhas de transmissão de 80, 100 e 150 (km) e cargas caracterizadas por um fator de potência de 0,92 atrasado e variando entre 5 e 25 (MVA). As tensões trifásicas foram medidas no barramento do gerador síncrono (BGER), uma vez que o interesse deste trabalho é analisar a viablidade da utilização deste algoritmo na proteção de geradores (Barbosa, 2007).

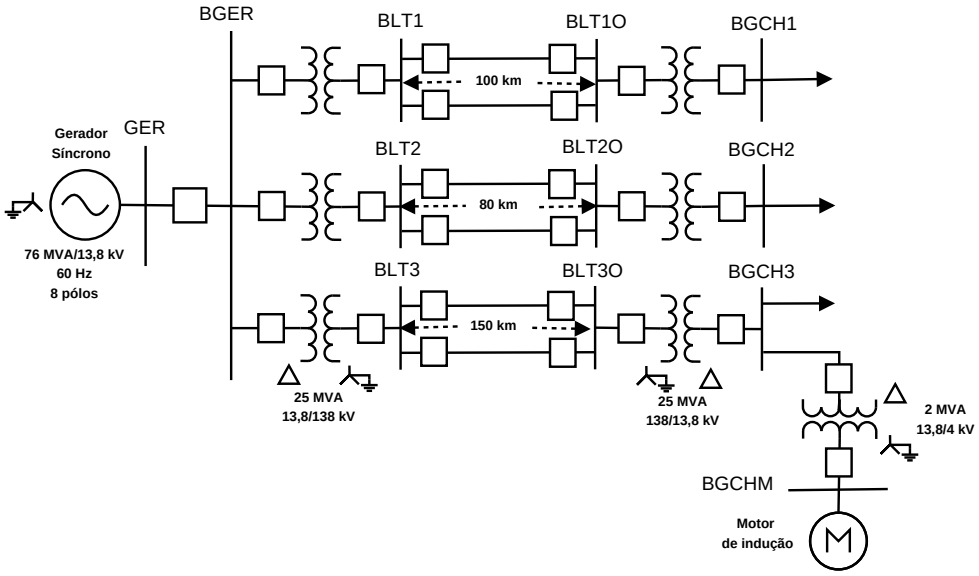

Figura 6.1: Modelo do SEP utilizado dispondo do software ATP.

6.1 Gerador síncrono

Os geradores são equipamentos fundamentais para o sistema elétrico, pois estes são os responsáveis pelo fornecimento da energia elétrica e pela manutenção da potência disponibilizada nos diversos ramos que constituem uma rede elétrica. Os geradores síncronos são utilizados na maioria das aplicações, pois estes, em sistemas isolados, suprem a potência consumida pelas cargas e agem como uma fonte de tensão cuja freqüência é determinada pela velocidade da fonte primária de energia (Fitzgerald et al., 1975).

Para realizar as simulações, foi adotado um sistema com um único gerador síncrono na sua topologia, viabilizando as análises do comportamento do mesmo às variações de carga e alterações estruturais da rede elétrica, como por exemplo, quando da saída de uma determinada linha de transmissão. Os parâmetros utilizados na modelagem deste gerador são apresentados na tabela 6.1 abaixo (Barbosa, 2007).

6.2 Controle do gerador

Além do gerador, vale salientar que foi considerado o controle de velocidade dinâmico para sistemas hidráulicos (Filho, 1984) *apud* (Barbosa, 2007) e o regulador automático de tensão (RAT) (Lee, 1992) que permitem melhor análise dos resultados. O controle da freqüência se dá através do controle de potência ativa (vazão turbinada) e o controle de tensão é feito atuando

Tabela 6.1: Parâmetros do gerador síncrono.

Descrição	**Valor**
Potência nominal (S)	76 (MVA)
Número de pólos (N_p)	8
Tensão nominal de linha (V_L)	13,8 (kV)
Freqüência (f)	60 (Hz)
Corrente de campo (I_{FD})	250 (A)
Resistência de armadura (R_a)	0,004 (p.u.)
Reatância de dispersão não saturada (X_l)	0,175 (p.u.)
Reatância de seqüência zero não saturada (X_O)	0,175 (p.u.)
Reatância síncrona de eixo direto não saturada (X_d)	1,150 (p.u.)
Reatância síncrona de eixo em quadradura não saturada (X_q)	0,685 (p.u.)
Reatância transitória de eixo direto não saturada (X'_d)	0,310 (p.u.)
Reatância subtransitória de eixo direto não saturada (X''_d)	0,210 (p.u.)
Reatância subtransitória de eixo em quadratura não saturada (X''_q)	0,182 (p.u.)
Constante de tempo transitória em vazio do eixo direto (τ'_{do})	5,850 (s)
Constante de tempo subtransitória em vazio do eixo direto (τ''_{do})	0,036 (s)
Constante de tempo subtransitória em vazio do eixo em quadratura (τ'_{do})	0,073 (s)
Momento de inércia (HICO)	3,333065 ($kg.m^2.10^6$)

sobre a potência reativa fornecida pelo gerador. A Figura 6.2 ilustra como se dá o controle de um gerador.

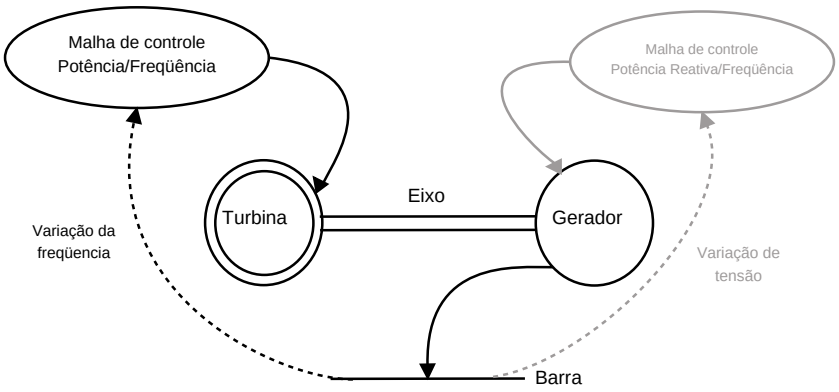

Figura 6.2: Controle do gerador síncrono.

6.2.1 Regulador de velocidade

As características desejadas de um gerador são que este forneça tensão constante e freqüência constante independente da carga elétrica a ele conectado, mas na verdade estes parâmetros sofrem influência direta da carga. Quando ocorre a adição de uma carga, alterando o equilíbrio consumo/geração, caso o gerador não possua um controle que atue sobre a turbina para que ele possa suprir a carga adicional, esse acréscimo será suprido pelo decréscimo da velocidade cinética

do gerador, e conseqüentemente a freqüência decairá. Em seguida, o sistema irá se estabilizar em um novo ponto de operação, porém, com uma freqüência menor, caracterizando assim a regulação própria do sistema. Desse modo fica clara a necessidade de um controle que atue sobre a turbina buscando manter a freqüência constante.

A equação 6.1 mostra a função de transferência do regulador de velocidade utilizado, no qual se têm a relação entre a posição ($\eta(s)$) do servo motor que aciona a turbina e o desvio de freqüência (ΔF(s)) (Barbosa, 2007).

$$\frac{\eta(s)}{\Delta F(s)} = -\frac{1}{R}.\frac{1+sT_r}{(1+sT_g)(1+s\frac{r}{R}T_r)} \tag{6.1}$$

A Tabela 6.2 apresenta os valores dos parâmetros utilizados para melhor representar o regulador de velocidade.

Tabela 6.2: Parâmetros do regulador de velocidade.

Descrição	**Valor**
Constante de tempo (T_g)	0,600 (s)
Constante de tempo Dashpot (T_r)	0,838 (s)
Estado transitório (r)	0,279
Estado permanente (R)	0,100

6.2.2 Regulador automático de tensão (RAT)

Como acontece geralmente na prática, é empregado um sistema de controle da excitatriz e o ajuste automático da corrente de campo, de forma a obter tensões cujo o valor eficaz seja constante, independente da variação da capacidade do gerador. A Figura 6.3 mostra o diagrama de blocos do controle de tensão implementado.

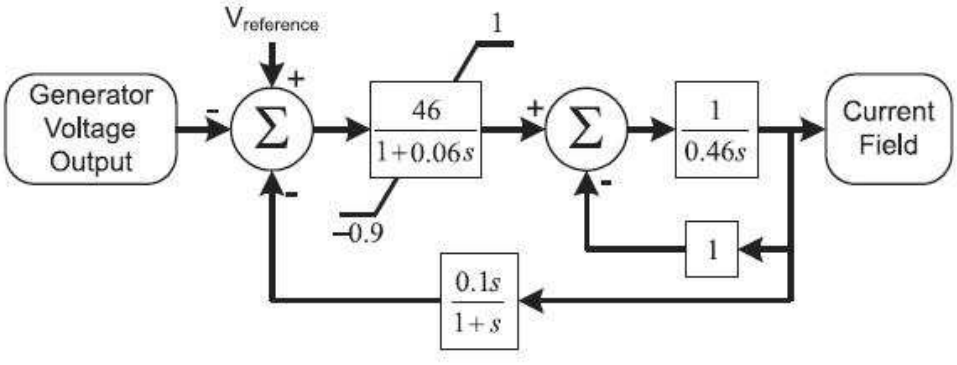

Figura 6.3: Diagrama de blocos simplificado do sistema de excitação (Barbosa et al., 2010).

6.3 Transformadores

Os transformadores de potência apresentam uma grande importância para o SEP, uma vez que estes são os responsáveis pela interligação dos três principais ramos dos sistemas de potência (geração, transmissão e distribuição), além de determinar os níveis de tensões desejados, elevando ou diminuindo a tensão de rede ao longo do sistema.

Devido a grande relevância destes equipamentos para a composição do modelo do sistema elétrico utilizado, empregou-se uma modelagem completa utilizando o ATP para representar o comportamento destes, incluindo a relação de transformação, curva de saturação e conexões dos rolamentos delta e estrela.

A curva de saturação do transformador foi obtida pela utilização da rotina SATURATION, a qual fornece os dados de corrente e de fluxo a partir dos valores de tensão e de corrente do transformador provenientes de ensaios, possibilitando a utilização dessas informações na caracterização do equipamento (EEUG, 1987). A Tabela 6.3 mostra as tensões aplicadas aos transformadores e a Tabela 6.4 informa as características elétricas do transformador simulado (Barbosa, 2007).

Tabela 6.3: Dados dos transformadores

Função	**Potência**	**Tensão Primário**	**Tensão Secundário**
Elevador	25 MVA	13,8 kV (Δ)	138,0 kV (Y)
Abaixador	25 MVA	138,0 kV (Y)	13,8 kV (Δ)
Abaixador/Motor	2 MVA	13,8 kV (Δ)	4,0 kV (Y)

Tabela 6.4: Parâmetros dos cabos

Descrição	R_+ **(Ω)**	L_+ **(mH)**
Imped. primária do trafo elevador	1,048	1,514
Imped. secundária do trafo elevador	0,017	91,577
Imped. primária do trafo abaixador	0,017	91,577
Imped. secundária do trafo abaixador	1,048	1,514
Imped. primária do trafo do motor	0,922	88,519
Imped. secundária do trafo do motor	0,258	4,238

6.4 Linhas de transmissão

As linhas de transmissão são os elementos que transportam a energia de seu ponto de geração ao centro de consumo, em muitos casos as linhas são extensas dado que o local de geração de energia estar distante geograficamente do local de consumo, ou mesmo ainda quando se deseja a interconexão de sistemas, com a intenção de que um sistema forneça energia quando outro está

com *déficit*, aumentando a estabilidade do sistema como um todo, o que pode ser observado hoje com o sistema interligado nacional.

A modelagem da linha de transmissão foi efetuada considerando-se parâmetros distribuídos e variantes com a freqüência, o que possibilita um estudo mais detalhado do comportamento da mesma perante ao dinamismo do SEP. Para tal situação, empregou-se a rotina *JMARTI* incluída no *software* ATP. Além disso, foi considerada uma linha transposta, uma vez que essa situação compensa os desequilíbrios magnéticos entre os condutores, as estruturas metálicas e o solo sob a linha de transmissão (Barbosa, 2007).

As linhas escolhidas para o sistema são trifásicas com extensão de 80, 100 e 150 (km), com circuito duplo mutuamente acoplados e tensão de linha de 138 (kV). As especificações dos cabos condutores estão na Tabela 6.5 e as dimensões da torre estão na Figura 6.4 (Barbosa, 2007). A resistência de terra considerada foi de 1000 $\frac{\Omega}{m}$.

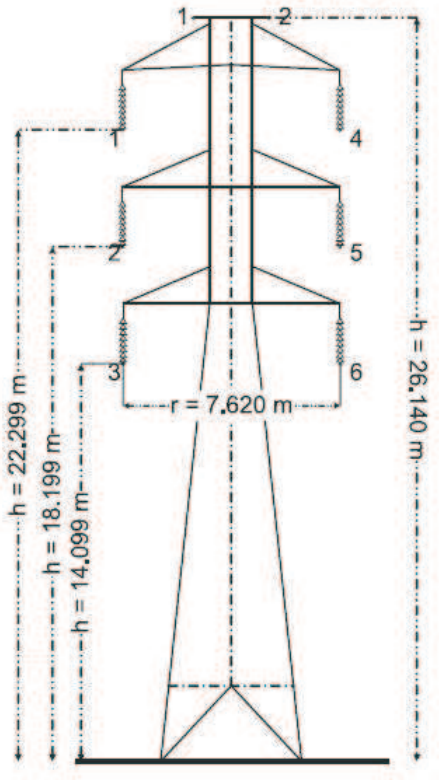

Figura 6.4: Torre de transmissão modelada.

Tabela 6.5: Parâmetros dos cabos.

Condutor	**Fase (Linnet)**	**Cabo guarda (EHS 5-16")**
Diâmetro externo (mm)	18,313	7,9375
Diâmetro interno (mm)	10,106	-
Resistência($\frac{\Omega}{m}$)	0,1901	5,5920

6.5 Motor de indução

Utilizando o modelo de máquina universal do ATP, o motor de indução trifásico (MIT) foi modelado com parâmetros reais, no qual diversas características elétricas e mecânicas foram consideradas, conforme as apresentadas na Tabela 6.6. A Tabela 6.7 mostra os dados do compressor conectado ao MIT utilizado (Gibelli, 2009).

Tabela 6.6: Parâmetros do MIT

Descrição	**Valor**
Tensão Nominal (V_L)	4 (kV)
Potência Nominal (S)	1180 (kW)
Velocidade Nominal (n)	187,24 (rad/s)
Rendimento Nominal (η)	94,8 (%)
Fator de Potência Nominal (fp)	0,91
Torque Nominal (τ)	6297 (Nm)
Momento de Inércia do Rotor (GD)	167 ($kgfm^2$)
Tempo de Rotor Bloqueado (t_{block})	20 (s)

Tabela 6.7: Parâmetros do Compressor conectado ao MIT.

Descrição	**Valor**
Torque Nominal (τ_N)	5675 (Nm)
Momento de Inércia do Rotor (GD)	828 ($kgfm^2$)

6.6 Cargas conectadas ao SEP

A variação dinâmica entre as potências gerada e a consumida pelas cargas produz alguns fenômenos no sistema elétrico, entre os quais se destaca a variação da freqüência. Nesse contexto, e a fim de consolidar a simulação realizada, optou-se pela utilização de um conjunto de cargas com características específicas que serão inseridas à rede elétrica em tempos distintos, com o objetivo de verificar o comportamento dinâmico e a resultante variação da freqüência. Os valores de resistência e indutância das cargas utilizadas na simulação são apresentados na Tabela 6.8 (Barbosa, 2007).

Tabela 6.8: Dados das cargas conectadas ao SEP.

Potência (MVA)	**Resistência (Ω)**	**Indutância (mH)**
25,00	7,008192	7,919224
15,00	11,68032	13,19871
10,00	17,52048	19,79806
5,00	35,04096	39,59612

É importante ressaltar que as cargas simuladas nesse sistema são compostas por elementos resistivos e indutivos, dimensionados para que o conjunto tenha um fator de potência de 0,92 indutivos. Também vale destacar que a potência total instalada e observada pelo gerador varia de 25 a 110% da potência nominal do mesmo.

Capítulo 7

Resultados

A análise dos resultados decorrentes da aplicação do algoritmo estimador da freqüência foi realizada em duas etapas. A primeira consistiu em testes aonde os sinais de entrada foram obtidos por formulação matemática, de forma sintetizada, via a aplicação do *software* Matlab®. A segunda corresponde ás situações de distúrbios simuladas sobre o sistema elétrico de potência ilustrado pela Figura 6.1, dispondo do *software* ATP. Para ambas as etapas, todas as formas de ondas de entrada foram geradas com freqüência de amostragem de 10 kHz.

7.1 Sinais sintetizados por formulação matemática

Nesta etapa, os sinais que representam as tensões trifásicas foram gerados a partir de formulação matemática através do *software* Matlab®. As formas de ondas sintetizadas são representadas pelas Equações 3.8, 3.9 e 3.10, aonde V_{rms} é dado por 120 (V), ϕ é igual a 0°e a freqüência de amostradem de 10 (kHz).

7.1.1 Condicionamento dos sinais

Cabe ressaltar que todo o pré-processamento dos sinais, até a sua consideração pelo algoritmo proposto, foi considerado nesta abordagem. A Figura 7.1, por exemplo, ilustra o atraso inserido na fase "A"dos sinais de entrada no processo de filtragem para se evitar a sobre-posição de espectros, o efeito *aliasing*. Para tal, foi utilizada a aproximação por *Butterworth* de segunda ordem, com uma freqüência de corte igual á 200 (Hz). Em seguida, o sinal filtrado é digitalizado por um CAD de 16 *bits* e frequência de reamostragem de 1920 (Hz), por onde se busca uma

aproximação aos relés comerciais. A Figura 7.2, mostra o sinal digitalizado, onde se observa a sua fidelidade com a entrada apresentada.

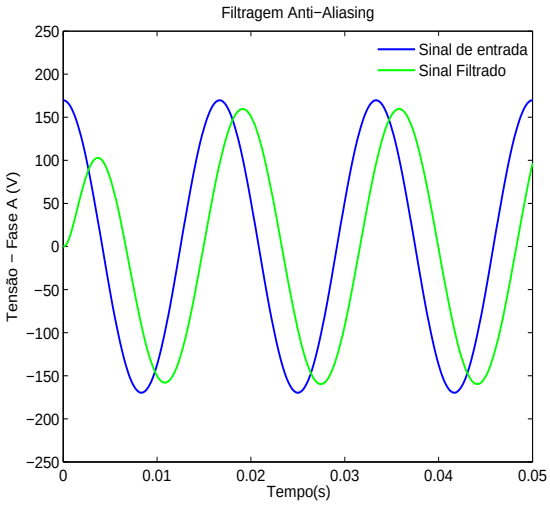

Figura 7.1: Filtragem da tensão fase "A".

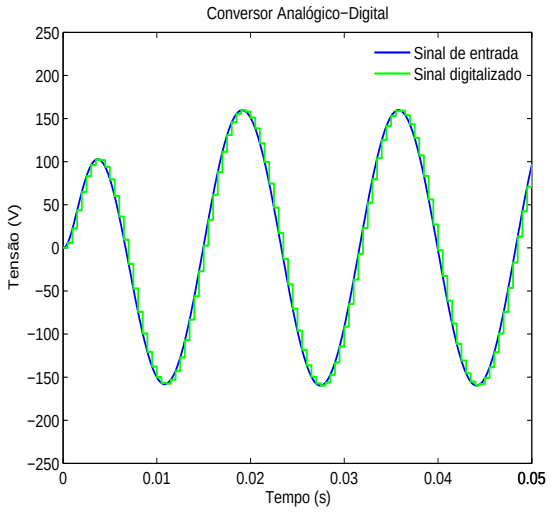

Figura 7.2: Digitalização do sinal filtrado.

7.1.2 Freqüência constante

Para este estudo de caso, considerou-se um sistema operando com uma freqüência constante igual á 60 (Hz), o que representa um perfeito sincronismo entre a geração e o consumo de energia. A oscilação presente no início do processamento corresponde à resposta ao degrau do filtro de saída, que cessa após 150 (ms). Da Figura 7.3, observa-se que a resposta do método descrito é plenamente satisfatória quanto á precisão. Nesta e nas demais figuras, apresenta-se também o

erro apresentado pela lógica implementada na estimação da frequência.

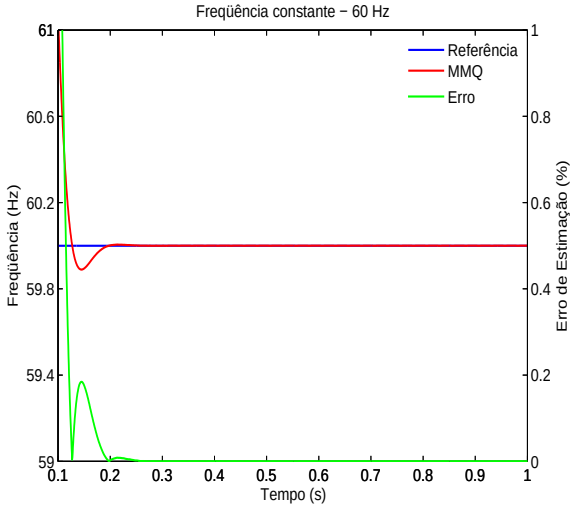

Figura 7.3: Teste considerando a freqüência fundamental como constante.

7.1.3 Distorções harmônicas

Para esta situação operativa, os sinais de tensão incorporam componentes harmônicas, aonde a distribuição percentual individual de distorção com relação a componente fundamental na composição das ondas é apresentada na Tabela 7.1.

Tabela 7.1: Composição das Componentes Harmônicas

Ordem	3	5	7	11
%	6,5	7,5	6,5	4,5

Os valores apresentados na Tabela 6.1 foram retirados a partir das referências apresentadas no Módulo 8 do *PRODIST* (ANEEL, 2008) para valores eficazes de tensão menores do que 1 (kV). A partir da Figura 7.4, buscou-se analisar o comportamento do algoritmo proposto quando os sinais de tensão apresentam distorções harmônicas. Observa-se que, neste caso, a resposta obtida foi similar ao teste da seção 7.1.2 em que foi considerado somente a freqüência fundamental, uma vez que as distorções das componentes harmônicas são atenuadas por causa da freqüência de corte da etapa de filtragem *anti-aliasing* e pela omissão da influência do terra na transformada $\alpha\beta$.

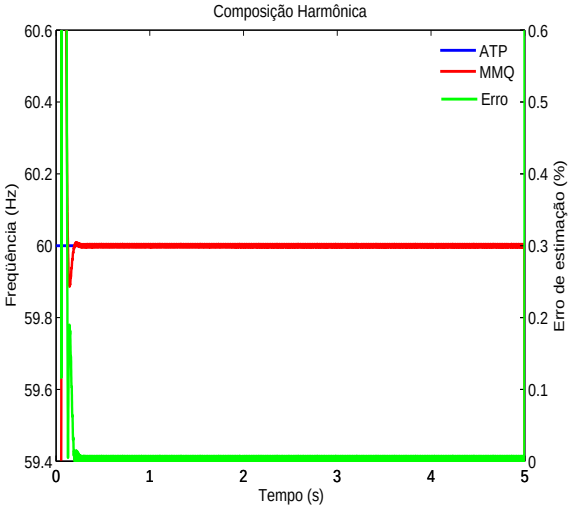

Figura 7.4: Resposta do algoritmo apresentado na presença da composição harmônica caracterizada.

7.1.4 Variação abrupta na freqüência

Este caso visa avaliar o comportamento do algoritmo MMQ quando da variação abrupta da freqüência de 60 (Hz) para 61 (Hz), o que pode, por exemplo, significar o desligamento de um grande bloco de cargas conectado ao SEP em análise. Este tipo de mudança representa uma situação irreal, uma vez que em situações práticas a mesma não pode variar instantaneamente. Na transição ocorrida, a resposta obtida apresentou um atraso de 50 (ms), o qual se deve a busca pela nova solução do algoritmo. Ressalta-se que é necessário uma janela de dados para rastrear corretamente o valor da freqüência, uma vez que, enquanto nas janelas de transição entre pré e pós-distúrbio não há uma freqüência característica. Conforme observado na Figura 7.5, o método descrito apresentou boa precisão e velocidade de estimação.

7.1.5 Variação dinâmica na freqüência

Este teste considera a freqüência fundamental oscilando de forma cossenoidal com amplitude de 4 (Hz) e um período igual á 0,5 (s). Tal situação pode representar um desequilíbrio periódico na relação geração-carga do sistema. Da Figura 7.6, observa-se que a resposta obtida segue o comportamento da referência com um atraso de aproximadamente 30 (ms).

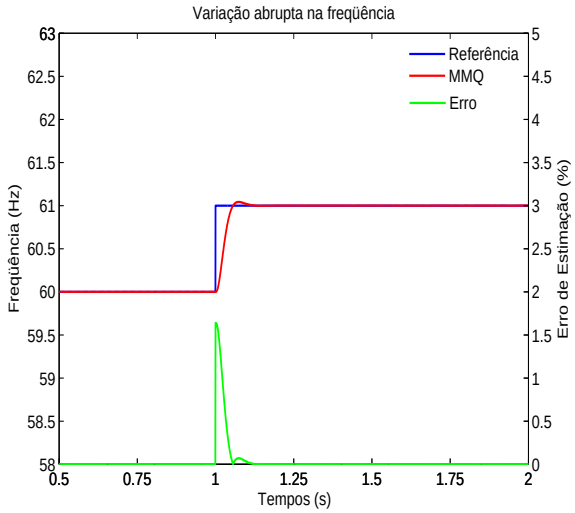

Figura 7.5: Teste considerando uma variação abrupta na freqüência.

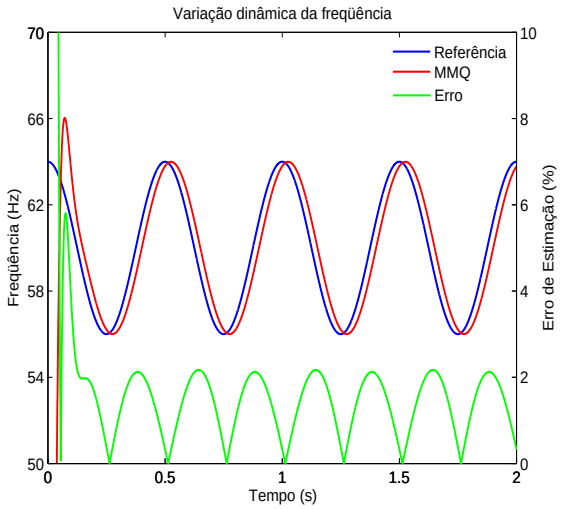

Figura 7.6: Teste considerando uma variação dinâmica na freqüência.

7.2 Sinais gerados através de simulações via o *software* ATP

Nesta sessão serão apresentados os resultados obtidos através da simulação no *software* ATP do SEP proposto no capítulo 6 (Figura 6.1).

7.2.1 Entrada e saída de um bloco de carga

A Figura 7.7 mostra o comportamento da frequência do gerador síncrono para a entrada de uma carga de 25 (MVA) no barramento BGCH3 (no instante de tempo de 3 (s)), com a sua respectiva remoção no instante de 7 (s). Como era esperado, a frequência do gerador varia

com a entrada e a saída de carga, sendo que quando ocorre o sobrecarregamento no sistema, a frequência tende a decair, enquanto que quando um bloco de carga é removido, a situação de excesso de geração é atingida, e a frequência tende a aumentar. Vale frisar que a resposta obtida pela lógica implementada é plenamente satisfatória, quanto a precisão e a velocidade de resposta do MMQ.

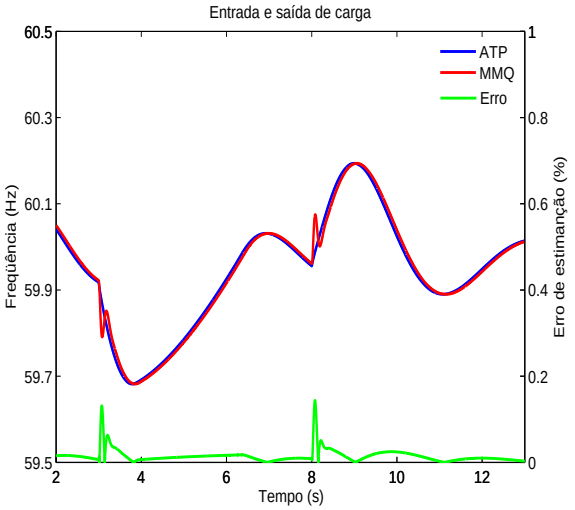

Figura 7.7: Entrada e saída súbita de carga.

7.2.2 Entrada e saída do MIT

Neste teste buscou-se avaliar a influência da entrada e saída do motor de indução trifásico (MIT) no comportamento da freqüência. No instante 7 (s) acontece a entrada do MIT, e no instante 12 (s), a sua desconexão do sistema. É possível afirmar que o MIT possui pouca influência na dinâmica do sistema. A resposta obtida possui grande fidelidade e segue aproximadamente a variação da frequência do sistema.

7.2.3 Curto-circuito franco fase-terra

Nesta situação, foi considerado um curto-circuito franco sustentado envolvendo a fase "A" com a conexão terra (AT) na linha de transmissão LT1 (Figura 6.1), no instante de 0,5 (s). Pela Figura 7.9, observa-se que o gerador conseguiu suprir a falta em tempo hábil, fazendo com que não houvesse a perda do sincronismo. Além disso, apesar do transiente no início da estimação, a resposta do método descrito é plenamente satisfatória quanto á precisão e a velocidade, pois veio a acompanhar o comportamento do SEP analisado.

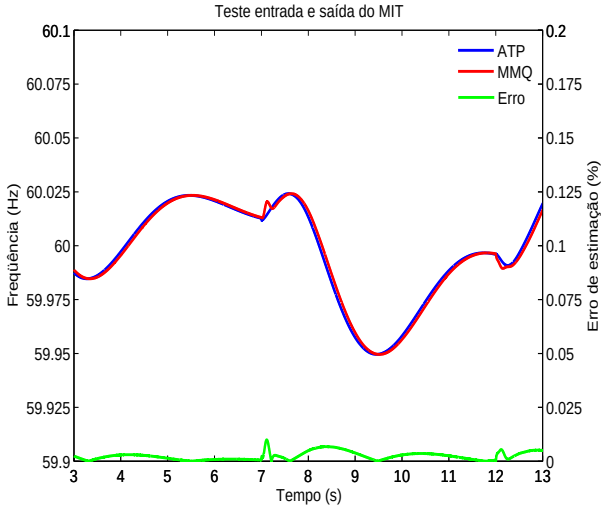

Figura 7.8: Entrada e saída do MIT.

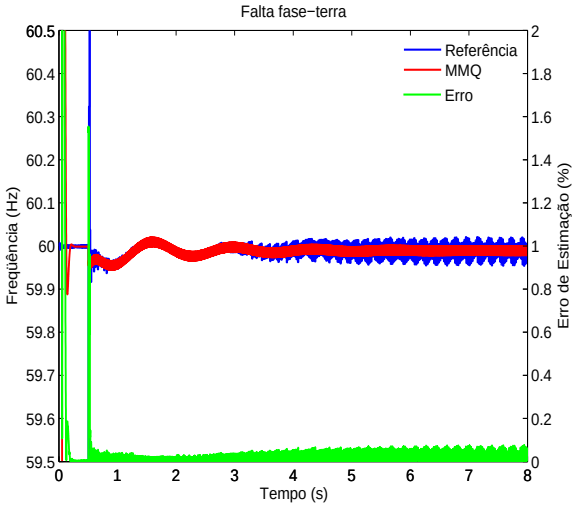

Figura 7.9: SEP sujeito a um curto-circuito franco do tipo fase-terra.

7.2.4 Curto-circuito bifásico na linha LT1

A Figura 7.10 ilustra o comportamento do algoritmo MMQ frente a uma situação de curto-circuito do tipo fase-fase a 50% do comprimento da linha de transmissão LT1 envolvendo as fases "A" e "B", aplicada no instante de 2,5 (s) e com duração de 80 (ms). Verifica-se que o SEP apresentou a tendência de recuperar o sincronismo com o gerador, porém a freqüência ultrapassou os limites de operação estabelecidos para o sistema nacional. A resposta obtida possui grande fidelidade e segue aproximadamente a variação da freqüência do sistema.

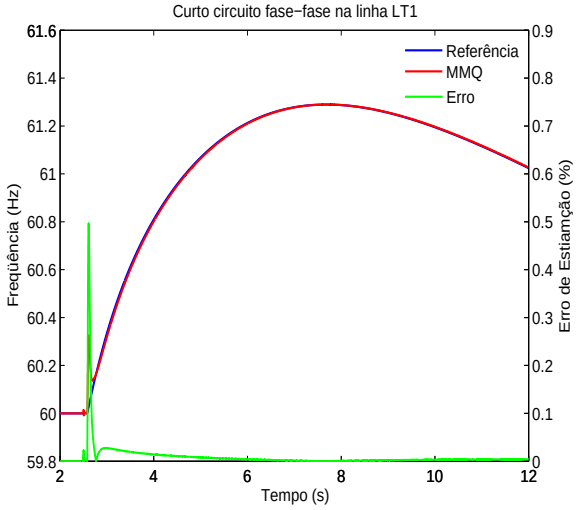

Figura 7.10: Estimação da freqüência para uma falta fase-fase na LT1.

7.2.5 Falta interna no transformador TR2E

Para esta situação, o teste executado foi para a situação de curto-circuito interno no transformador TR2E á 50% do enrolamento no lado de alta tensão, aplicada no instante de 3 (s) e com duração igual á 35 (ms). Na Figura 7.11, observa-se a tendência do SEP em retomar o sincronismo do gerador. Por fim, a resposta do algoritmo é precisa e acompanha a dinâmica do sistema.

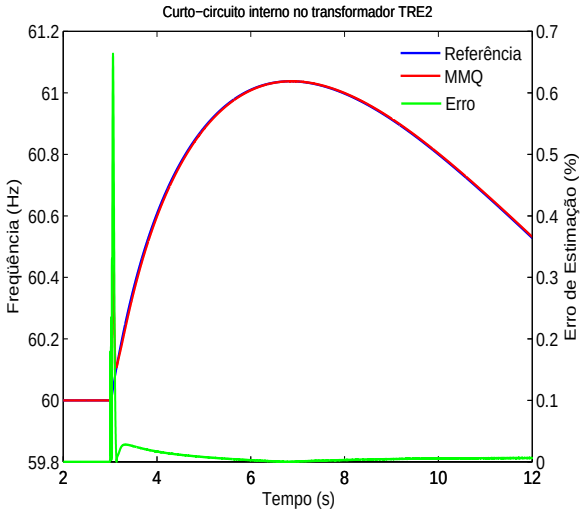

Figura 7.11: Curto-circuito no transformador TRE2.

Capítulo 8

Conclusões

Esta pesquisa apresentou um método de estimação da frequência em SEP através da filtragem adaptativa baseada no MMQ. O uso de filtros adaptativos permitiu um processamento rápido e preciso dos dados, possibilitando uma adequação mais rápida dos seus parâmetros.

O início deste projeto ocorreu com o levantamento bibliográfico das técnicas mais comuns para estimar o comportamento da freqüência, com o intuito de analisar os possíveis resultados e soluções para o problema. No estudo da linguagem de programação VHDL e da estrutura do FPGA, houve uma inserção do aluno nos conceitos de linguagem de descrição em *hardware* e às tecnologias que empregam elementos de lógica programável. Em seguida, ocorreu a conversão do algoritmo em linguagem de programação *C* para a linguagem VHDL, sendo esta fase a proposta principal deste trabalho.

O método implementado em VHDL possui todos os estágios típicos de um relé micropocessado de freqüência. Os resultados decorrentes da aplicação, mostraram-se precisos, apresentado robustez frente às variações na freqüência dos sinais de entrada. Desse modo, há o encorajamento na continuidade dos estudos de maneira a viabilizar e analisar a sua aplicação em um relé usando a tecnologia FPGA.

Cabe salientar que foram cumpridas todas as etapas referentes ao cronograma inicialmente proposto para o trabalho de conclusão de curso.

Coloco que enquanto aluno do curso de Engenharia Elétrica, com ênfase em Sistemas de Energia e Automação, observei um bom e promissor aproveitamento na realização desta pesquisa, pois pude me familiarizar com temas como a proteção digital de sistemas elétricos de potência, método dos mínimos quadrados, linguagens de programação em C, VHDL e em FPGAs que,

provavelmente, poderão fazer parte de futuros trabalhos acadêmicos e/ou de outras atividades profissionais.

Para firmar a qualidade e o empenho nas atividades realizadas, ressalta-se que os resultados decorrentes deste tema de pesquisa foram publicados nos seguintes eventos técnico-científicos:

- ROCHA, R. L. S.; BARBOSA, D.; OLESKOVICZ, M.; COURY, D. V. A estimação da freqüência em um sistema elétrico de potência pelo método dos mínimos quadrados. In: VIII Latin-American Congress on Electricity Generation and Transmission, 2009, Ubatuba - SP. VIII CLAGTEE - VIII Latin-American Congress on Electricity Generation and Transmission, 2009.
- ROCHA, R. L. S.; BARBOSA, D.; OLESKOVICZ, M.; COURY, D. V. Filtros adaptativos baseados no método dos mínimos quadrados para a estimação da frequência em um sistema elétrico de potência. In: Simpósio Internacional de Iniciação Cientifica da USP, 2009, São Carlos - SP. 17º SIICUSP, 2009.

Apêndice A

Arquivos do Algoritmo em VHDL

Este anexo apresenta os códigos implementados em linguagem VHDL.

A.1 Código Biblioteca Principal - filtro.vhd

```
1  library IEEE;
   use IEEE.std_logic_1164.all;
   use ieee.math_real.all;
   use ieee.math_complex.all;
   use std.textio.all;
6
   package filtro is
   constant tjanela :integer := 8;
   constant tfiltro : integer := 5;
   constant NPULSE : integer := 6;
11 constant linha : integer := 1;
   constant coluna : integer := 2;

   subtype t_controle is std_logic_vector (0 to tjanela-1);
   subtype t_controle2 is std_logic_vector (0 to tfiltro-1);
16

   type complex_ptr is access complex;
   type vetor is array (integer range<>) of real;
   type vetor_ptr is access vetor;
   type U_vetor is array (0 to tjanela-1) of real;
21 type coef_vetor is array (0 to tfiltro-1) of real;
   type matriz is array (integer range<>, integer range<>) of real;
   type matriz_ptr is access matriz;
   type trans_matriz is array (0 to linha, 0 to coluna) of real;

26 type dados is record
     temp : vetor_ptr;
     Veta : vetor_ptr;
     Vetb : vetor_ptr;
     Vetc : vetor_ptr;
31   famos : real;
     npontos : integer;
   end record dados;

   component filtro_adaptativo is
```

```
  generic(tamanho : integer;
          fs : real);
   port( adap : in std_ulogic;
         V_re : in  U_vetor;
         V_im : in  U_vetor;
         f :  out real;
         adap_ok :  out std_ulogic);
end component filtro_adaptativo;

 component transf_modal is
   generic(tamanho : integer);
   port(transf : in std_ulogic;
       U_norm_a : in U_vetor;
       U_norm_b : in U_vetor;
       U_norm_c : in U_vetor;
       U_re : out U_vetor;
       U_im : out U_vetor;
       transf_ab : out std_ulogic);
 end component transf_modal;
end package filtro;

package body filtro is
end package body filtro;
```

A.2 Biblioteca de Impressão dos Dados - imprime.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.math_real.all;
use ieee.math_complex.all;
use std.textio.all;

package imprime is
procedure plot_xy(variable arq : in vetor_ptr;
                  variable arq2 : in vetor_ptr;
                  constant l : in integer);
procedure plot_xyy(variable arq : in vetor_ptr;
                   variable arq2 : in vetor_ptr;
                   variable arq3 : in vetor_ptr;
                   constant l : in integer);
procedure plot_xyyy(variable entr : in dados);
end package imprime;

package body imprime is
procedure plot_xyyy(variable entr : in dados) is
  variable i : integer := 0;
  file outfile : text is out "arquivo_de_saida.txt";
  variable outline : line;
  begin
      for i in 0 to (entr.npontos - 1) loop
        write(outline,entr.temp(i));
        write(outline, string'("  "));
        write(outline,entr.Veta(i));
        write(outline, string'("  "));
        write(outline,entr.Vetb(i));
        write(outline, string'("  "));
        write(outline,entr.Vetc(i));
        writeline(outfile, outline);
     end loop;
end procedure plot_xyyy;
```

```
procedure plot_xyy(variable arq : in vetor_ptr;
                   variable arq2 : in vetor_ptr;
                   variable arq3 : in vetor_ptr;
                   constant l : in integer) is

  file outfile : text is out "arquivo_de_saida.txt";
  variable outline : line;
  variable i : integer := 0;
  begin
      for i in 0 to (l - 1) loop
        write(outline,arq(i));
        write(outline, string'("  "));
        write(outline,arq2(i));
        write(outline, string'("  "));
        write(outline,arq3(i));
        writeline(outfile, outline);
      end loop;
end procedure plot_xyy;

procedure plot_xy(variable arq : in vetor_ptr;
                  variable arq2 : in vetor_ptr;
                  constant l : in integer) is

  file outfile : text is out "arquivo_de_saida.txt";
  variable outline : line;
  variable i : integer := 0;
  begin
      for i in 0 to (l - 1) loop
        write(outline,arq(i));
        write(outline, string'("  "));
        write(outline,arq2(i));
        writeline(outfile, outline);
      end loop;
end procedure plot_xy;
end package body imprime;
```

A.3 Biblioteca de Normalização dos Dados - normalização.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.math_real.all;
use work.filtro.all;

package normaliz_dados is
procedure norm_dados(variable ent : dados;
                     pontos : integer;
                     pos : integer;
                     signal Ua : out U_vetor;
                     signal Ub : out U_vetor;
                     signal Uc : out U_vetor);
end package normaliz_dados;

package body normaliz_dados is

procedure norm_dados(variable ent : dados;
                     pontos : integer;
                     pos : integer;
                     signal Ua : out U_vetor;
                     signal Ub : out U_vetor;
                     signal Uc : out U_vetor) is
  variable i, k : integer := 0;
```

```
  variable max_a, max_b, max_c : real := 0.0;
  begin

  for i in 0 to pontos-1 loop
    Ua(i) <= ent.Veta(i+pos)/14000.0;
    Ub(i) <= ent.Vetb(i+pos)/14000.0;
    Uc(i) <= ent.Vetc(i+pos)/14000.0;
   end loop;

end procedure norm_dados;
end package body normaliz_dados;
```

A.4 Biblioteca CAD - cad.vhd

```
use STD.textio.all;
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.math_real.all;
use ieee.math_complex.all;
use std.textio.all;

package cad is
procedure cad_16(variable entrada : in dados;
                 constant  fam : real := 1920.0;
                 constant  ampl : integer;
                 constant  numbits : integer;
                 variable saida : out dados);
end package cad;

package body cad is
procedure cad_16(variable entrada : in dados;
                 constant  fam : real := 1920.0;
                 constant  ampl : integer;
                 constant  numbits : integer;
                 variable saida : out dados) is

   variable i,j : integer :=0;
   variable preamos, namost, nnivel : integer;
   variable nqa, nqb, nqc, preamos1, param, npasso : real;
   variable nqa1, nqb1, nqc1 : integer;

    begin
      param := entrada.famos/fam + 0.5;
      preamos := integer(floor(entrada.famos/fam + 0.5));
      namost := integer(entrada.npontos/preamos);
      saida.npontos := namost;

      saida.temp := new vetor(0 to namost);
      saida.Veta := new vetor(0 to namost);
      saida.Vetb := new vetor(0 to namost);
      saida.Vetc := new vetor(0 to namost);

      nnivel := 2**numbits;
      npasso := real(ampl)/nnivel;

      j := 0;
      i := 0;
      while ( i < namost and j < entrada.npontos) loop
          saida.temp(i) := entrada.temp(j);
          nqa := entrada.Veta(j)/npasso;
          nqb := entrada.Vetb(j)/npasso;
```

```
            nqc := entrada.Vetc(j)/npasso;
            nqa1 := integer(nqa + 0.5);
            nqb1 := integer(nqb + 0.5);
            nqc1 := integer(nqc + 0.5);

            if (nqa1 < (- nnivel)) then
                nqa1 := - nnivel;
            end if;
            if (nqb1 < (- nnivel)) then
               nqb1 := - nnivel;
            end if;
            if (nqc1 < (- nnivel)) then
                nqc1 := - nnivel;
            end if;
            if (nqa1 >=  nnivel) then
                nqa1 := nnivel;
            end if;
            if (nqb1 >=  nnivel) then
                nqb1 := nnivel;
            end if;
            if (nqc1 >=  nnivel) then
                nqc1 := nnivel;
            end if;

            saida.Veta(i) := npasso*real(nqa1);
            saida.Vetb(i) := npasso*real(nqb1);
            saida.Vetc(i) := npasso*real(nqc1);

            i := i + 1;
            j := j + preamos;
        end loop;
        saida.famos := 1.0/(entrada.temp(preamos) - entrada.temp(0));

 end procedure cad_16;
end package body cad;
```

A.5 Biblioteca do Filtro *Butterworth* - butterworth.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.math_real.all;
use work.filtro.all;

package butterworth is

procedure filtro_butter_1(variable ent : in dados;
                          ordem : in integer;
                          f_corte : in real;
                          variable saida : out dados);
procedure filtro_butter_2(variable ent : in vetor_ptr;
                          ordem : in integer;
                          f_corte : in real;
                          pontos : in integer;
                          famos : in real;
                          variable saida : out vetor_ptr;
                          signal f_filt : out real);
procedure calc_coef(ordem : in integer;
                    fs : in real;
                    fc : in real;
                    variable coef : out matriz_ptr);
procedure ndetcoef(ordem : in integer;
```

```
                    fs :in real;
                    fc : in real;
                    n : out vetor_ptr);
end package butterworth;

package body butterworth is
procedure filtro_butter_1(variable ent : in dados;
                          ordem : in integer;
                          f_corte : in real;
                          variable saida : out dados) is

  variable i,j,p : integer := 0;
  variable coe_filtro : matriz_ptr;
  variable s,t : integer;
  variable aux : dados;
begin

  aux.npontos := ent.npontos;
  aux.temp := new vetor(0 to ent.npontos);
  aux.Veta := new vetor(0 to ent.npontos);
  aux.Vetb := new vetor(0 to ent.npontos);
  aux.Vetc := new vetor(0 to ent.npontos);
  aux.famos := ent.famos;

  for p in 0 to ent.npontos - 1 loop
    aux.temp(p) := ent.temp(p);
    aux.Veta(p) := 0.0;
    aux.Vetb(p) := 0.0;
    aux.Vetc(p) := 0.0;
 end loop;

  coe_filtro := new matriz(0 to 1, 0 to ordem);
  calc_coef(ordem, ent.famos, f_corte, coe_filtro);

  for i in ordem to aux.npontos - 1 loop
    aux.Veta(i) := coe_filtro(1,0)*ent.Veta(i);
    aux.Vetb(i) := coe_filtro(1,0)*ent.Vetb(i);
    aux.Vetc(i) := coe_filtro(1,0)*ent.Vetc(i);

    for j in 1 to ordem loop
      aux.Veta(i) := aux.Veta(i)+coe_filtro(1,j)*ent.Veta(i-j)-coe_filtro(0,j)*aux.Veta(i-j);
      aux.Vetb(i) := aux.Vetb(i)+coe_filtro(1,j)*ent.Vetb(i-j)-coe_filtro(0,j)*aux.Vetb(i-j);
      aux.Vetc(i) := aux.Vetc(i)+coe_filtro(1,j)*ent.Vetc(i-j)-coe_filtro(0,j)*aux.Vetc(i-j);
    end loop;

    aux.Veta(i) := aux.Veta(i)/coe_filtro(0,0);
    aux.Vetb(i) := aux.Vetb(i)/coe_filtro(0,0);
    aux.Vetc(i) := aux.Vetc(i)/coe_filtro(0,0);
  end loop;

  saida := aux;
end procedure filtro_butter_1;

procedure filtro_butter_2(variable ent : in vetor_ptr;
                          ordem : in integer;
                          f_corte : in real;
                          pontos : in integer;
                          famos : in real;
                          variable saida : out vetor_ptr;
                          signal f_filt : out real) is

  variable i,j,s,k : integer := 0;
  variable coe_filtro : matriz_ptr;
  variable aux : vetor_ptr;
```

```
88
begin

  aux := new vetor'(0 to pontos-1 => 0.0);
  coe_filtro := new matriz(0 to 1, 0 to ordem);
93  calc_coef(ordem, famos, f_corte, coe_filtro);

  for i in ordem to pontos-1 loop
    aux(i) := coe_filtro(1,0)*ent(i);
    for j in 1 to ordem loop
98    aux(i) := aux(i) + coe_filtro(1,j)*ent(i-j) - coe_filtro(0,j)*aux(i-j);
    end loop;
    aux(i) := aux(i)/coe_filtro(0,0);
    f_filt <= aux(i) after 10 ns;
  end loop;
103
saida := aux;

end procedure filtro_butter_2;

108 procedure calc_coef(ordem : in integer;
                       fs : in real;
                       fc : in real;
                       variable coef : out matriz_ptr) is

113  variable A, B, C, D, E, F : real := 0.0;
  variable aa, bb, cc, dd, ee, ff : real := 0.0;
  variable na, nb : vetor_ptr;
  variable prew, T : real := 0.0;

118  begin
  T := 1.0/fs;
  ndetcoef(ordem, fc, fc, na);
  prew := (2.0*fs)*tan((math_pi*fc)/(2.0*fs));
  coef := new matriz(0 to 1, 0 to ordem);
123
  case ordem is
    when 2 =>
      aa := na(0)/(prew**2.0);
      bb := na(1)/(prew**1.0);
128   cc := na(2);

      A := cc*(T**2.0) + 2.0*bb*T + 4.0*aa;
      B := 2.0*cc*(T**2.0) - 8.0*aa;
      C := cc*(T**2.0) - 2.0*bb*T + 4.0*aa;
133
      coef(0,0) := 1.0;
      coef(0,1) := B/A;
      coef(0,2) := C/A;

138   coef(1,0) := 1.0*cc*(T**2.0)/A;
      coef(1,1) := 2.0*cc*(T**2.0)/A;
      coef(1,2) := 1.0*cc*(T**2.0)/A;

    when 3 =>
143   aa := na(0)/(prew**3.0);
      bb := na(1)/(prew**2.0);
      cc := na(2)/(prew**1.0);
      dd := na(3);

148   A := dd*(T**3.0)+2.0*cc*(T**2.0)+4.0*bb*T + 8.0*aa;
      B := 3.0*dd*(T**3.0)+2.0*cc*(T**2.0)-4.0*bb*T - 24.0*aa;
      C := 3.0*dd*(T**3.0)-2.0*cc*(T**2.0)-4.0*bb*T + 24.0*aa;
      D := dd*(T**3.0)-2.0*cc*(T**2.0)+4.0*bb*T - 8.0*aa;
```

```
153         coef(0,0) := 1.0;
            coef(0,1) := B/A;
            coef(0,2) := C/A;
            coef(0,3) := D/A;

158         coef(1,0) := 1.0*dd*(T**3.0)/A;
            coef(1,1) := 3.0*dd*(T**3.0)/A;
            coef(1,2) := 3.0*dd*(T**3.0)/A;
            coef(1,3) := 1.0*dd*(T**3.0)/A;

163       when 5 =>
            aa := na(0)/(prew**5.0);
            bb := na(1)/(prew**4.0);
            cc := na(2)/(prew**3.0);
            dd := na(3)/(prew**2.0);
168         ee := na(4)/(prew**1.0);
            ff := na(5);

            A := ff*(T**5.0)+2.0*ee*(T**4.0)+4.0*dd*(T**3.0)+8.0*cc*(T**2.0)+16.0*bb*T+32.0*aa;
            B := 5.0*ff*(T**5.0)+6.0*ee*(T**4.0)+4.0*dd*(T**3.0)-8.0*cc*(T**2.0)-48.0*bb*T-160.0*aa;
173         C := 10.0*ff*(T**5.0)+4.0*ee*(T**4.0)-8.0*dd*(T**3.0)-16.0*cc*(T**2.0)+32.0*bb*T+320.0*aa;
            D := 10.0*ff*(T**5.0)-4.0*ee*(T**4.0)-8.0*dd*(T**3.0)+16.0*cc*(T**2.0)+32.0*bb*T-320.0*aa;
            E := 5.0*ff*(T**5.0)-6.0*ee*(T**4.0)+4.0*dd*(T**3.0)+8.0*cc*(T**2.0)-48.0*bb*T+160.0*aa;
            F := ff*(T**5.0)-2.0*ee*(T**4.0)+4.0*dd*(T**3.0)-8.0*cc*(T**2.0)+16.0*bb*T-32.0*aa;

178         coef(0,0) := 1.0;
            coef(0,1) := B/A;
            coef(0,2) := C/A;
            coef(0,3) := D/A;
            coef(0,4) := E/A;
183         coef(0,5) := F/A;

            coef(1,0) := 1.0*ff*(T**5.0)/A;
            coef(1,1) := 5.0*ff*(T**5.0)/A;
            coef(1,2) := 10.0*ff*(T**5.0)/A;
188         coef(1,3) := 10.0*ff*(T**5.0)/A;
            coef(1,4) := 5.0*ff*(T**5.0)/A;
            coef(1,5) := 1.0*ff*(T**5.0)/A;

          when others =>
193         report "Erro no filtro de Butterworth"
            severity error;
        end case;
    end procedure calc_coef;

198 procedure ndetcoef(ordem : in integer;
                       fs :in real;
                       fc : in real;
                       n : out vetor_ptr) is

203   variable tf : real := 0.0;
      variable i : integer := 0;
      variable aux : vetor_ptr;
      begin

208   aux := new vetor(0 to ordem);
      aux(0) := 1.0;
      tf := fc/fs;
      for i in 1 to ordem loop
    aux(i) := (tf)*aux(i-1)*cos(((real(i-1))*math_pi)/(2.0*real(ordem)))/sin((real(i)*math_pi)/(2.0*real(ordem)));
213   end loop;
      n := aux;
    end procedure ndetcoef;
```

```
end package body;
```

A.6 Código Principal - *rele.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.filtro.all;
use work.butterworth.all;
use work.normaliz_dados.all;
use ieee.math_complex.all;
use ieee.math_real.all;

entity ent is
  port(
    rd : in std_logic;
    tempo : in real;
    amost : in integer;
    Va : in real:= 0.0;
    Vb : in real:= 0.0;
    Vc : in real:= 0.0;
    amrz : out std_logic;
    fest_filt : out real;
    dfest : out real);
end entity ent;

architecture behav of ent is

  signal transf: std_ulogic := '0';
  signal f, f_filt : real;
  signal U_re,U_im : U_vetor;
  signal U_norm_a, U_norm_b, U_norm_c : U_vetor;
  signal transf_ok : t_controle;
  signal transf_ab, adap, fitera : std_ulogic;
  signal adap_ok : std_ulogic := '0';
  signal W_re, W_im : coef_vetor;
  signal q : integer;

 begin
  inicio : process is
    variable i,index,j : integer := 0;
    variable limit : integer;
    variable count,N : integer := 0;
    variable freq, freq_filtrado,dfdt,fdfdt : vetor_ptr;
    variable T_delay : time;
    variable dados_digitais,dados_entrada,dados_filtrados,dados_filtrados2    : dados;

  begin
      wait until rd  = '1';
      report "Inicializando...";

      wait on amost;
      dados_entrada.npontos := amost;
      amrz <= '1' after 10 ns;

      dados_entrada.temp := new vetor(0 to dados_entrada.npontos);
      dados_entrada.Veta := new vetor(0 to dados_entrada.npontos);
      dados_entrada.Vetb := new vetor(0 to dados_entrada.npontos);
      dados_entrada.Vetc := new vetor(0 to dados_entrada.npontos);

      index := 0;
```

```
        report "Adquirindo os dados...";
59      while (index < dados_entrada.npontos) loop
          wait on tempo;
          dados_entrada.temp(index) := tempo;
          dados_entrada.Veta(index) := Va;
          dados_entrada.Vetb(index) := Vb;
64        dados_entrada.Vetc(index) := Vc;
          index := index + 1;
        end loop;

        dados_entrada.famos := 1.0/(dados_entrada.temp(1) - dados_entrada.temp(0));
69
        report "Condicionando os sinais Va, Vb e Vc";
        filtro_butter_1(dados_entrada,2,200.0,dados_filtrados);
        cad_16(dados_filtrados, 1920.0, 13800, 16, dados_digitais);
        filtro_butter_1(dados_digitais,2,200.0,dados_filtrados2);
74
        report "Processando...";
        freq := new vetor'(0 to dados_digitais.npontos-1 => 0.0);
        for i in 0 to dados_digitais.npontos-tjanela-1 loop
         norm_dados(dados_filtrados2, tjanela, i, U_norm_a, U_norm_b, U_norm_c );
79       transf <= '1';
         wait until transf_ab = '1';
         transf <= '0';
         wait until transf_ab = '0';
         adap <= '1';
84       wait until adap_ok ='1';
         adap <= '0';
         wait until adap_ok = '0';
         freq(i) := f;
        end loop;
89
        report "Tratando o sinal de saida...";
        freq_filtrado := new vetor(0 to dados_digitais.npontos-1);
        filtro_butter_2(freq,2,20.0,dados_digitais.npontos,dados_digitais.famos,freq_filtrado,f_filt);

94      dfdt := new vetor'(0 to dados_digitais.npontos-1=>0.0);
        fdfdt := new vetor'(0 to dados_digitais.npontos-1=>0.0);
        for j in NPULSE to dados_digitais.npontos-1 loop
        dfdt(j) := abs((freq_filtrado(j)-freq_filtrado(j-1))/(dados_digitais.temp(j)- dados_digitais.temp(j-1)));

99       if((dfdt(j)<1.5) and (dfdt(j-1)<1.5) and (dfdt(j-2)<1.5) and (dfdt(j-3)<1.5) and (dfdt(j-4)<1.5)) then
            fdfdt(j) := freq_filtrado(j);
         else
          fdfdt(j) := fdfdt(j-1);
         end if;
104      dfest <= fdfdt(j) after 10 ns; --after Tpd;
        end loop;

        report "Salvando a saida...";
        plot_xyy(dados_digitais.temp,freq,fdfdt,dados_digitais.npontos);
109    wait on rd;
     end process inicio;

     filt_adap : component filtro_adaptativo
                 generic map(tamanho => tfiltro,
114                          fs => 2000.0)
                 port map(adap => adap,
                          V_re => U_re,
                          V_im => U_im,
                          f => f,
119                       adap_ok => adap_ok);

    freq_filtrada : process (f_filt) is
```

```
begin
    fest_filt <= f_filt;
end process freq_filtrada;
end architecture behav;

configuration filtro_structure of ent is
for behav
  for filt_adap : filtro_adaptativo
    use entity work.filtro_adaptativo(implement);
  end for;
  for transformada_ab : transf_modal
    use entity work.transform_ab(behav);
  end for;
end for;
end configuration filtro_structure;
```

A.7 Código da Transformada $\alpha\beta$ - transformada.vhd

```
library IEEE;
use ieee.std_logic_1164.all;
use work.filtro.all;
use ieee.math_complex.all;
use ieee.math_real.all;

entity transform_ab is
  generic(tamanho : integer);
  port(transf : in std_ulogic;
       U_norm_a : in U_vetor;
       U_norm_b : in U_vetor;
       U_norm_c : in U_vetor;
       U_re : out U_vetor;
       U_im : out U_vetor;
       transf_ab : out std_ulogic);
end entity transform_ab;

architecture behav of transform_ab is

signal transf_ok : t_controle;
constant M : trans_matriz :=
               (0 => (0=>sqrt(2.0/3.0),1=>-0.5*sqrt(2.0/3.0),2=>-0.5*sqrt(2.0/3.0)),
                1 => (0=>0.0          ,1=>sqrt(2.0)/2.0     ,2=>-sqrt(2.0)/2.0));
begin
transform : for L in 0 to  tamanho - 1 generate
 begin
   calc : process is
   begin

    transf_ok(L) <= '0';
    wait until transf = '1';
     U_re(L) <= U_norm_a(L)*M(0,0) + U_norm_b(L)*M(0,1) + U_norm_c(L)*M(0,2);
     U_im(L) <= U_norm_a(L)*M(1,0) + U_norm_b(L)*M(1,1) + U_norm_c(L)*M(1,2);
     transf_ok(L) <= '1';
    wait on transf;
   end process calc;
  end generate transform;

  and_tjanela : process is
   variable p : integer := 0;
   variable result : std_ulogic;
     begin
```

```
      for p in 1 to tjanela - 1 loop
        result := transf_ok(p) and transf_ok(p-1);
      end loop;

      transf_ab <= result;--after 10 ns;
      wait on transf_ok;
end process and_tjanela;
end architecture behav;
```

A.8 Código do Filtro Adaptativo - adaptativo.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.filtro.all;
use ieee.math_complex.all;
use ieee.math_real.all;

entity filtro_adaptativo is
  generic(tamanho : integer := 8;
          fs : real:=2000.0);
  port( adap : in std_ulogic;
        V_re : in  U_vetor;
        V_im : in  U_vetor;
        f : out real;
        adap_ok : out std_ulogic := '0');
end entity filtro_adaptativo;

architecture implement of filtro_adaptativo is

signal XW_re, XW_im  : coef_vetor;
signal X_re, X_im : coef_vetor;
signal W_re, W_im : coef_vetor := (0 to tfiltro-1 => 0.0);--cond. iniciais
signal Y_atual, Y_ant : complex;
signal fir,saida_ok,pes_ok,coef_calc : std_ulogic;
signal e_atual, e_ant,pp : complex;
signal mi : real := 0.0;
signal atualiza_coef : std_ulogic := '0';
signal mult_pes,coef_ok : t_controle2;
signal fitera : std_ulogic;

constant pi : real := 3.14159265358979323846264338327 95;
constant precisao : real := 0.0001;
constant n_itera : integer := 1000;
constant lambida : real := 0.97;
constant rho : real := 0.99;
constant mi_sup : real := 0.18;
constant mi_inf : real := 0.1;
constant psi : real := 0.01;
constant rmu_max_min : real := 0.0001;

begin
   processo_principal : process is
  variable erro_atual, erro_ant,p, T : complex;
  variable k,l,cont : integer := 0;
  variable H : real;
  variable f1 : real;
  begin
    adap_ok <= '0';
    wait until adap = '1';
    erro_atual := cmplx(1.0,1.0);
```

```
    p := cmplx(0.0,0.0);
    H := fs/(2.0*pi);
    cont := 0;

    while (abs(erro_atual) > precisao) and (cont < n_itera) loop
      for l in (tfiltro-1) to tjanela-1 loop
        erro_ant := erro_atual;

        for k in 0 to (tfiltro-1) loop
          X_re(k) <= V_re(l-k);
          X_im(k) <= V_im(l-k);
        end loop;

        Y_ant <= Y_atual after 10 ns;

        fir <= '1';
        wait until saida_ok = '1';
        fir <= '0';
        wait until saida_ok = '0';

        erro_atual := cmplx(V_re(l),V_im(l))-Y_atual;
        e_atual <= erro_atual;-- after 10 ns;
        p := rho*p + (1.0-rho)*erro_atual*erro_ant;
        mi <= lambida*mi + psi*(p.re*p.re + p.im*p.im);

        if mi > mi_sup then
          mi <= mi_sup;
        else
          if mi < mi_inf then
            mi <= mi_inf;
          end if;
        end if;

        atualiza_coef <= '1';
        wait until coef_calc = '1';
        atualiza_coef <= '0';
        wait until coef_calc = '0';

        T := Y_atual*conj(Y_ant);

        if (T.re = 0.0 and T.im = 0.0) then
                f1 := 0.0;
        else
         f1 := H*arctan(T.im,T.re);
        end if;
      end loop;
      cont := cont+1;
    end loop;

    f <= f1;
    adap_ok <= '1';-- after 10 ns;
    wait on adap;
end process processo_principal;

adaptativo : for L in 0 to tamanho-1 generate
   begin

   atuali : process is
   variable coef : complex;

   begin
    coef_ok(L) <= '0';
     wait until atualiza_coef = '1';
      coef := cmplx(W_re(L),W_im(L)) + (mi*(conj(e_atual)*cmplx(X_re(L), X_im(L))));
```

```
        W_re(L)<= coef.re;
        W_im(L)<= coef.im;
        coef_ok(L) <= '1';
       wait on atualiza_coef;
 end process atuali;
 end generate adaptativo;

 and_coef : process is
 variable p : integer := 0;
 variable result : std_ulogic;

  begin
   for p in 1 to tfiltro - 1 loop
    result := coef_ok(p) and coef_ok(p-1);
  end loop
  coef_calc <= result after 10 ns;
  wait on coef_ok;
 end process and_coef;

  and_filtro : process is
  variable p : integer := 0;
  variable result : std_ulogic;
  begin

  for p in 1 to tfiltro - 1 loop
    result := mult_pes(p) and mult_pes(p-1);
  end loop;

  pes_ok <= result after 10 ns;
  wait on mult_pes;
end process and_filtro;

 mult_entr_pes : for i in 0 to tamanho-1 generate

   begin
     pes : process is
     begin
      mult_pes(i) <= '0';
      wait until fir = '1';
      XW_re(i) <= X_re(i)*W_re(i) + X_im(i)*W_im(i);
      XW_im(i) <= X_im(i)*W_re(i) - X_re(i)*W_im(i);
      mult_pes(i) <= '1';
     wait on fir;
   end process pes;
end generate mult_entr_pes;

 acumulador : process is
 variable i : integer := 0;
 variable a,b : complex;
 variable y : complex;

  begin
   saida_ok <= '0';
   wait until pes_ok = '1';
   y := cmplx(0.0,0.0);
   for i in 0 to tamanho -1 loop
     y :=  y + cmplx(XW_re(i), XW_im(i));
   end loop;
   Y_atual <= y;
   saida_ok <= '1';
   wait on pes_ok;
 end process acumulador;
end architecture implement;
```

A.9 Código do *Test Bench* - testbench.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.filtro.all;

entity test_bench is
end entity test_bench;

architecture test of test_bench is
signal rd,amrz : std_logic;
signal amost : integer;
signal tempo, Va, Vb, Vc, fest_filt, dfest :  real;

begin
        dut : entity work.ent(behav)
              port map(rd, tempo, amost, Va,Vb, Vc,amrz, fest_filt,dfest);

        stimulus : process is

                  file infile : text;
                 variable inline : line;
                  variable p,count :integer := 0;
                  variable Tpd1, Tpd2 : time; --Tpd1~=10Tpd2
                  variable t, ta, tb, tc : real := 0.0;
        begin
            rd <= '1' after Tpd2;

            file_open(infile, "arquivo_de_entrada", read_mode);
      while (not endfile(infile)) loop
       readline(infile,inline);
       count := count +1;
      end loop;

      amost <= count after Tpd1;
      wait until amrz = '1';
      file_close(infile);

      file_open(infile, "arquivo_de_entrada", read_mode);

                while (not endfile(infile) and p < count) loop
                         readline(infile, inline);
                                read(inline,t);
                                read(inline,ta);
                                read(inline,tb);
                                read(inline,tc);

                               tempo <= t;
                                Va <= ta;
                                Vb <= tb;
                                Vc <= tc;
                                wait for 20 ns;
                                p := p + 1;
                 end loop;

         wait;
        end process stimulus;
end architecture test;
```

Referências Bibliográficas

Aboulnasr, T. e K. Mayyas (1997). A robust variable step-size lms-type algorithm: analysis and simulations. *IEEE Transactions on Signal Processing 45*(3), 631–639.

Aghazadeh, R., H. Lesani, M. Sanaye-Pasand, e B. Ganji (2005). New technique for frequency and amplitude estimation of power system signals. *IEEE Proc. - Gener. Transm. Distrib.* (3), 435–440.

Akke, M. (1997). Frequency estimation by demodulation of two complex signal. *IEEE Transactions on Power Delivery 12*(1), 157–163.

ANEEL (2008). *Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional (PRODIST) Módulo 8 - Qualidade de Energia*.

Ashenden, P. (2002). *The Designer's Guide to VHDL* (2nd. ed.). Morgan Kaufmann Publishers.

Barbosa, D. (2007). Estimação da frequência em sistemas de elétricos de potência através da filtragem adaptativa. Dissertação de Mestrado, Universidade de São Paulo (USP).

Barbosa, D., M. da Silva, A. Bernades, M. Oleskovicz, e D. Coury (2006). Lógica fuzzy aplicada á proteção de transformadores. *19º SBSE*, 1–5.

Barbosa, D., U. Neto, H. Branco, M. Oleskovicz, e D. Coury (2010). Impacto do paralelismo na proteção diferencial de transformadores de potência. *O Setor Elétrico*.

Blackburn, J. L. (1987). *Protective Relaying: Principles and Applications*. MARCEL DEKKER INC.

Brown, S. e Z. Vranesic (2005). *Fundamentals of Digital Logic with VHDL design*. McGraw Hill.

Caminha, A. C. (1977). *Introdução á proteção dos sistemas elétricos*. São Paulo: Editora Edgard Blücher Ltda.

Chang, K. C. (1997). Digital design and modeling with vhdl synthesis. *IEEE Computer Society*.

Chen, M. e Z. Yu (2003). Determination of frequency variation effect on dft-based algorithms for harmonic and flicker measurements. *Transmission and Distribution Conference and Exposition 1*(2), 422–427.

Darwin, C. (2004). *A origem das espécies*. Ediora.

Dash, P., A. Pradhan, e G. Panda (1999). Frequency estimation of distorted power system signals using extend complex kalman filter. *IEEE Transactions on Power Delivery 14*(3), 761–766.

EEUG (1987). *Alternative Transients Program*.

El-Naggar, K. M. e H. K. M. Youssef (2000). A genetic based algorithm for frequency estimation-relaying applications. *Electric Power Systems Research 55*, 173–178.

Farhang-Boroujeny, B. (1999). *Adaptive Filters : Theory and Applications*. Chichester : John Wiley and Sons Ltd.

Filho, C. V. (1984). *Operação de sistemas de potência com controle automático de geração*. Ed. Campus Ltda.

Fitzgerald, A. E., C. K. Jr., e A. Kusko (1975). *Máquinas Elétricas*. McGraw Hill do Brasil.

Gibelli, G. B. (2009). Resposta dinâmica dos motores de indução trifásicos a afundamentos de tensão na rede de alimentação. Dissertação de Mestrado, Universidade de São Paulo (USP).

Girgs, A. e T. Hwang (1984). Optimal estimation of voltage phasors and frequency estimation deviation using linear and non-linear kalman filtering: theory and limitations. *IEEE Transactions on Power Apparatus and Systems PAS-103*(10), 2943–2948.

Girgs, A. e T. Hwang (1997). An adaptive neural network approach for the estimation of power system frequency. *Electric Power Systems Research 41*, 203–210.

Girgs, A. A. e F. M. Ham (1982). A new fft-based digital frequency relay for load shedding. *IEEE Transactions on Power Apparatus and Systems PAS-101*(2), 433–439.

Haykin, S. (2002). *Adaptive Filter Theory* (4th. ed.). Prentice Hall, Inc.

Horowitz, S. e A. Phadke (1996). *Power System Relaying* (2nd. ed.). Research Studies Press Ltd.

IEEE (1993). *IEEE Standard VHDL Language Reference Manual* (2th. ed.). IEEE. IEEE Std. 1076™-2008.

IEEE (2009). *IEEE Standard VHDL Language Reference Manual* (4th. ed.). IEEE. IEEE Std. 1076™-2008.

Johns, A. e K. Salman (1995). *Digital Protection for Power Systems*, Volume 15th. IEEE Power Series.

Lee, E. C. (1992). *IEEE Recommended practice for excitation system models for power system stability studies*. IEEE Std 421.5-1992.

Leuven, K. U. (1987). *Alternative transients program rule book*.

Lobos, T. e J. Rezmer (1997). Real-time determination of power systems frequency. *IEEE Transactions on Instrumetation and Measurement 46*(4), 877–881.

Lu, S. (2005). Aplication of dft filter bank to power frequency harmonic measurement. *IEEE Proceedings 152*(1), 132–136.

Maxfield, C. (2004). *Design warrior guide's to FPGA*. Newnes.

MentorGraphics (2009). *ModelSim®Tutorial : Software Version 6.5b*.

Naylor, D. e S. Jones (1997). *VHDL : A Logic Synthesis Approach*. Chapman e Hall.

ONS (2001). *Submódulo 10.8 - Norma de Operação - Controle da Geração em Operação Norma*.

Palnitkar, P. (2003). *Verilog HDL : A Guide to Digital Design and Synthesis* (2nd. ed.). Prentice Hall, Inc.

Perry, D. (2002). *VHDL : Programming by Example*. McGraw-Hill.

Pradhan, A., A. Routray, e A. Basak (2005). Power system frequency estimation using least mean square technique. *IEEE Transactions on Signal Processing 20*(3), 1812–1816.

Sachdev, M. e M. Giray (1985). A least error squares technique for determining power system frequency. *IEEE Transactions on Power Apparatus and Systems PAS-104*(2), 437–443.

Salcic, Z. e R. Mikhael (1999). A new method for instantaneous power system frequency measurement using reference points detection. *Electric Power Systems Research 50*, 97–102.

Sanaye-Pasand, M. e V. J. Marandi (2004). Frequency estimation of distorted signals for control and protection of power system. *IEEE Transactions on Power Apparatus and Systems ?*(?), 631–634.

Soliman, S., R. Alammari, e M. El-Hawary (2003). Frequency and harmonics evaluation in power networks using fuzzy regression technique. *Eletrical Power Systems Research 66*, 171–177.

Vargas, E., S. Souza, M. Oleskovicz, e D. Coury (2005). Aplicação de algoritmos genéticos em relés digitais de freqüência. *Sixth Latin-American Congress on Electricity Generation and Transmission (VI CLAGTEE), Mar Del Plata, Argentina, Anais (CD)*.

Whitley, D. (1993). *A Genetic Algorithm tutorial*. Computer Science Department, Colorado State University, Fort Collins.

Widrow, B. e E. Hoff (1960). Adaptive switching circuits. *Proc. IRE WESCON Conv. Rec.*, 96–104.

Widrow, B., J. McCool, e M. Ball (1975). The complex lms algorithm. *Proceedings of the IEEE 63*(4), 719–720.

Wolf, W. (2004). *FPGA-Based system design*. Prentice Hall Professional Technical Reference.

Xue, S. Y. e S. X. Yang (2007). Accurate and fast frequency tracking for power systems signals. *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, 2754–2759.

Yulan, C. e F. Cunyang (2007). A new method of frequency measurement of power system. *Second IEEE Conference on Industrial Eletronics and Applications 1*(2), 2522–2525.

Zivanovic, R. (2005). Eliminating bias in frequency estimantion based on prony's method. *Power Tech IEEE*, 1–4.