UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS DE CURITIBA
CURSO DE ENGENHARIA INDUSTRIAL ELÉTRICA
ÊNFASE AUTOMAÇÃO

EDUARDO KOJI HIEDA
LUIS PAULO SCOLARO CORDEIRO
MARCOS VINÍCIUS CAMPOS RIBEIRO

IMPLEMENTAÇÃO DIGITAL DE ALGORITMO DE SINCRONISMO APLICADO AO CONTROLE DE CONVERSORES ESTÁTICOS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2013

EDUARDO KOJI HIEDA
LUIS PAULO SCOLARO CORDEIRO
MARCOS VINÍCIUS CAMPOS RIBEIRO

IMPLEMENTAÇÃO DIGITAL DE ALGORITMO DE SINCRONISMO APLICADO AO CONTROLE DE CONVERSORES ESTÁTICOS

Trabalho de Conclusão de Curso de Graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do curso de Engenharia Industrial Elétrica – Ênfase em Automação do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Eduardo Félix Ribeiro Romaneli, Dr. Eng.

Co-Orientador: Amauri Amorin Assef, M. Eng.

CURITIBA
2013

EDUARDO KOJI HIEDA
LUIS PAULO SCOLARO CORDEIRO
MARCOS VINÍCIUS CAMPOS RIBEIRO

IMPLEMENTAÇÃO DIGITAL DE ALGORITMO DE SINCRONISMO APLICADO AO CONTROLE DE CONVERSORES ESTÁTICOS

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro Eletricista do curso de Engenharia Industrial Elétrica com ênfase em Automação do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 09 de Maio de 2013.

_____

Prof. Paulo Sérgio Walênia, Esp.
Coordenador de Curso
Engenharia de Controle e Automação

_____

Prof. Marcelo de Oliveira Rosa, Dr.
Coordenador dos Trabalhos de Conclusão de Curso
de Engenharia Elétrica do DAELT

ORIENTAÇÃO

_____

Prof. Eduardo Félix Ribeiro Romaneli, Dr. Eng.
Universidade Tecnológica Federal do Paraná
Orientador.

_____

Prof. Amauri Amorin Assef, M. Eng.
Universidade Tecnológica Federal do Paraná
Co-orientador

BANCA EXAMINADORA

_____

Prof. Amauri Amorin Assef, M. Eng.
Universidade Tecnológica Federal do Paraná

_____

Prof. Marco José da Silva, Dr. Ing.
Universidade Tecnológica Federal do Paraná

_____

Profa. Rosângela Winter, M. Eng.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia de Controle e Automação

Dedicamos este trabalho a todas as pessoas que nos auxiliaram no decorrer de nossa jornada e participaram no nosso processo de evolução.

All is one, and that one is love/light, light/love, the Infinite Creator. (Ra)

You create your own Reality. (Seth)

RESUMO

HIEDA, Eduardo; CORDEIRO, Luis; RIBEIRO, Marcos. Implementação digital de algoritmo de sincronismo aplicado ao controle de conversores estáticos. 2013. 95f. Trabalho de conclusão de curso (Engenharia Industrial Elétrica, ênfase em Automação) – Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Este trabalho apresenta a implementação de um PLL (*Phase Locked Loop*) em *firmware* utilizando-se de um dsPIC, visto que este DSP possui funções e interrupções pré-definidas em sua biblioteca interna facilitando o desenvolvimento do *firmware*. O PLL implementado foi testado com o auxílio de um protótipo montado no laboratório LPEE (Laboratório de Processamento Eletrônico de Energia) da UTFPR, sendo que a referência utilizada no desenvolvimento deste foi a rede elétrica da COPEL (Companhia Paranaense de Energia), com uma tensão de 127 V e frequência de 60 Hz. Verificou-se que o sistema de controle implementado é muito eficiente nessas condições, mantendo o sincronismo constante com precisão e dentro dos limites impostos pela concessionária de energia.

Palavras-chave: *Phase Locked Loop*. DSP. Algoritmo. *Firmware*.

ABSTRACT

HIEDA, Eduardo; CORDEIRO, Luis; RIBEIRO, Marcos. Development of digital synchronization algorithm applied to static converter control. 2013. 95f. Trabalho de conclusão de curso (Engenharia Industrial Elétrica, ênfase em Automação) – Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

This research presents the implementation of a PLL firmware using a dsPIC, since this DSP has predefined functions and interruptions in its internal library assisting the PLL firmware development. The implemented PLL was tested on a prototype developed on LPEE at UTFPR. The reference used in the development of this prototype was the COPEL electrical grid, this one working on a 127 V voltage and 60 Hz frequency. It has been found that the implemented control system is efficient in these conditions, maintaining accurately synchronism in the electrical grid within the limits imposed by the electric company.

Keywords: Phase Locked Loop. DSP. Algorithm. Firmware.

ÍNDICE DE FIGURAS

SUMÁRIO

LISTA DE SIGLAS

ADPLL	PLL 100% digital
A/D	Analógico / Digital
ALU	Unidade lógica aritmética
AM	Amplitude Modulada
CI	Circuito Integrado
DAG	Gerador de Endereços
D/A	Digital / Analógico
DCO	Oscilador Controlado Digitalmente
DIP	*Dual In-Line Package*
DMA	Acesso Direto à Memória
DPLL	PLL digital
DSP	Processador Digital de Sinais
FIFO	*First In First Out*
FM	Frequência Modulada
I/O	Entrada / saída
JK	*Flip-Flop Set-Reset*
JTAG	*Joint Test Action Group*
LPEE	Laboratório de Processamento Eletrônico de Energia
LPLL	PLL analógico
MIPS	Milhões de Instruções por Segundo
MOSFET	Transistor de Semicondutor de Oxido Metálico por Efeito de Campo
MPPT	Rastreador do Ponto de Máxima Potência
NCO	Oscilador Controlado Numericamente
NPN	Transistor Lógica Negativa
PI	Controlador Proporcional Integral
PLL	*Phase Locked Loop*
PNP	Transistor Lógica Positiva
PV	Painel Fotovoltaico
PWM	Modulação por Largura de Pulso
PWM1H	PWM 1 *High*
PWM3H	PWM 3 *High*
PWM3L	PWM 3 *Low*

QFP	*Quad Flat Package*
RAM	Memória de Acesso Aleatório
SRAM	Memória Estática de Acesso Aleatório
TV	Televisão
UART	*Universal Asynchronous Receiver/Transmitter*
UTFPR	Universidade Tecnológica Federal do Paraná
VCO	Oscilador Controlado por Tensão
XOR	OU Exclusivo

LISTA DE SÍMBOLOS

CA	Corrente Alternada
CC	Corrente Contínua
Crede	Capacitor do secundário do *flyback* de dois secundários
D1	Diodo do *flyback* elementar
Drect1	Diodo 1 do secundário do *flyback* de dois secundários
Drect2	Diodo 2 do secundário do *flyback* de dois secundários
I	Corrente
Lrede	Indutor do secundário do *flyback* de dois secundários
Sac1	Chave 1 do secundário do *flyback* de dois secundários
Sac2	Chave 2 do secundário do *flyback* de dois secundários
Spv	Chave do primário do *flyback* de dois secundários
V	Tensão
Vac	Tensão da rede elétrica
Vcc	Tensão Continua

1 INTRODUÇÃO

O PLL ou *Phase Locked Loop* é um importante circuito encontrado em aplicações eletrônicas de diversos tipos. Estes circuitos são encontrados desde receptores de AM/FM, MODEMs, celulares, telefones sem fio, sintetizadores de frequência, instrumentos digitais e analógicos além de uma infinidade de aplicações onde frequências estejam presentes.

O PLL é um circuito que tem como função manter a sincronização de um sistema em relação a um determinado sinal. O sistema basicamente opera produzindo um sinal de saída em que a fase esta relacionada com a fase do sinal de entrada de referência, sendo que no momento em que os dois sinais estão em sincronismo, o erro de fase entre a saída de sinal do oscilador e o sinal de referência é zero, ou mantém-se constante (BEST, 2007).

A primeira estrutura PLL foi implementada em 1932 pelo engenheiro francês de Bellescize, porém a sua ampla difusão ocorreu somente a partir da década de 1960 com a diminuição dos custos de componentes e circuitos integrados (ANALOG, 1999), sendo que a partir da década de 1980 com a popularização dos processadores digitais e a diminuição de preço ocorreu um maior interesse na área de desenvolvimento de estruturas PLL digitais (WANG & SUN, 2004). As estruturas PLL consistem basicamente de um elo fechado com três componentes: um detector de fase, o filtro e um VCO (oscilador controlado por tensão), sendo estas estruturas PLL divididas em quatro tipos de sistema de controle:

* PLL Linear ou Analógico: o detector de fase, o filtro e o VCO são analógicos.
* PLL Digital: PLL analógico com o detector de fase digital.
* PLL 100% Digital: o detector de fase, filtro e o oscilador são digitais, com a utilização de um oscilador controlado numericamente (NCO).
* PLL por *Firmware*[1]: os blocos funcionais são implementados em *firmware* em um microcontrolador ou microprocessador.

1.1 TEMA

Apesar da existência de vários tipos de estruturas analógicas e digitais, o PLL por *firmware* vem ganhando destaque nos últimos 20 anos devido à popularização e, consequentemente, redução de custos dos processadores digitais (FISCHETTE, 2009). As

[1] Um *software* passa a ser chamado de *firmware* a partir do momento que o mesmo é inserido em algum dispositivo programável.

vantagens deste sistema de controle digital frente aos analógicos são a possibilidade de implementação de leis de controle sofisticadas, as quais podem se ajustar a não-linearidades, variações de parâmetros ou às tolerâncias do projeto por meios de análises e estratégias de adequação e compatibilização, algo muito difícil ou impossível de se implementar de forma analógica (BUSO & MATTAVELLI, 2006).

O controle de conversores estáticos é uma das inúmeras aplicações do PLL por *firmware*, e caso haja uma conexão deste conversor com a rede elétrica, esta interligação deverá ser regulamentada e autorizada. Os requisitos básicos para que esta seja realizada consistem na necessidade de que a amplitude, frequência e fase da tensão do conversor devem estar exatamente em concordância com a tensão da rede (COPEL, 2002). Esses fatores podem ser todos monitorados pelo sistema de controle, o qual assegura que a conexão esteja de acordo considerando a amplitude, frequência e fase da tensão gerada.

1.1.1 Delimitação do tema

Para a adequação aos requisitos da conexão de um conversor à rede elétrica, este necessita de estratégias de controle e sincronismo, os quais são objeto de estudo nesse trabalho mais especificamente com a implementação de um algoritmo PLL. Para isso foi necessário a amostragem do sinal da rede, a sua compatibilização com os limites impostos pelo processador digital, o cálculo de fatores como valores médios e efetivos da onda analisada, as passagens por zero da senóide, a geração de uma onda senoidal sintética a partir de uma tabela programada no processador digital, o controle das chaves de sincronia em fase e frequência com o sinal de entrada, entre outros fatores implantados e posteriormente explanados.

1.2 PROBLEMAS E PREMISSAS

O sistema de sincronismo digital foi otimizado para trabalhar na frequência de 60 Hz, com o intuito de conectar um conversor estático à rede de energia elétrica. Este sistema necessita de um processador digital além de um circuito que possa fornecer uma amostra da rede elétrica para a efetiva comparação e assim sincronização.

Uma das considerações necessárias foi à adequação do formato senoidal da tensão da rede elétrica tanto em amplitude quanto em nível médio. Por exemplo, a tensão da rede passa por valores negativos e os circuitos digitais normalmente não possuem capacidade de efetuar

a leitura por estarem limitados em tensões de leitura positivas. O monitoramento contínuo e a implementação de um *firmware* no processador digital de sinais, possibilita manter o sistema em sincronismo independente da condição inicial ou de alterações nos parâmetros da rede elétrica.

O DSP foi escolhido a partir da disponibilidade de interrupções e rotinas apropriadas especificamente ao controle eletrônico de energia, as quais não estão disponíveis na maioria dos microcontroladores atuais disponíveis comercialmente a baixo custo.

Para efeitos de testes do algoritmo de sincronismo foi desenvolvido um protótipo, o qual permitiu aferir o sincronismo em fase e frequência para o funcionamento adequado do conversor estático através do processamento dos sinais provenientes da rede pelo algoritmo implementado no DSP.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Implementar em DSP de um PLL por *firmware aplicado a um* controle digital de sincronismo aplicado a conversores estáticos, utilizando um DSP, .

1.3.1.1 Objetivos Específicos

- Realizar a revisão bibliográfica sobre controles analógicos e digitais de sincronismo;
- Especificar os componentes usados no protótipo;
- Montar um protótipo operacional;
- Implementar o algoritmo de controle digital;
- Executar os testes de sincronismo em fase e frequência com o sinal de entrada da rede.

1.4 JUSTIFICATIVA

Este trabalho inclui os passos necessários para a implementação de um PLL em *firmware* que pode ajudar a guiar outros alunos e equipes da UTFPR que desejarem utilizar este sistema de sincronismo em seus respectivos trabalhos.

Sendo o PLL um sistema de controle confiável e com número reduzido de componentes, este foi escolhido por ser amplamente usado industrialmente, porém com pouca

ou quase nenhuma adoção no momento pelos alunos da UTFPR, ainda mais se tratando da sua implementação via *firmware* em DSP.

O controle digital é mais confiável e preciso que o controle analógico, visto que, este pode ser centralizado num único componente. Devido a este motivo, o número de componentes do circuito diminui consideravelmente, ocasionando, um aumento na confiabilidade do circuito e redução de custo. Além disso, a implementação do controle digital possibilita que ajustes sejam feitos na saída do controlador sem que haja substituição de componentes, pois o DSP pode ser reprogramado para atuar da maneira desejada.

Finalmente o PLL implementado em *firmware* pode ser considerada uma técnica de controle sofisticada que permite a implementação de correções de não-linearidades intrínsecas aos componentes e circuitos.

1.5 PROCEDIMENTOS METODOLÓGICOS

O método adotado para a realização do presente trabalho consistiu em inicialmente realizar a pesquisa bibliográfica dos temas a serem abordados, na qual se buscou obter um grande número de informações e fontes a respeito dos controles de sincronismo analógicos e digitais, métodos de sincronização PLL, controladores e compensadores além de uma visão geral sobre o DSP.

Em seguida, foi iniciada a fase de desenvolvimento do projeto, o qual foi montado e testado no laboratório LPEE, começando pela topologia e o dimensionamento dos componentes utilizados, além da escolha do DSP adequado às necessidades e a programação de seu *firmware*.

Finalizando os procedimentos metodológicos, foi realizada a análise dos testes com o protótipo montado e foi comprovado o sincronismo de fase e frequência em relação à rede. Os resultados obtidos são analisados e apresentados ao término deste trabalho, em sua conclusão.

1.6 ESTRUTURA DO TRABALHO

No capítulo 1, Introdução, é apresentado o objetivo deste estudo, o problema que o motivou e os métodos de pesquisas que foram utilizados para obtenção do resultado desejado.

No capítulo 2, onde se encontra a Revisão Bibliográfica, é apresentada uma bibliografia a respeito de circuitos de sincronismo, além dos princípios básicos de constituição e seu funcionamento. No mesmo capítulo são abordados métodos de

implementação do PLL analógicos e digitais. Por fim, a utilização do sincronismo digital e a importância e aplicação do mesmo em conversores estáticos é apresentada.

No Capitulo 3 está abordado o desenvolvimento do *firmware*, processo de montagem do protótipo e os resultados dos testes executados.

No capítulo 4 são apresentadas as Conclusões Gerais deste trabalho e sugestões para trabalhos futuros

2 REVISÃO BIBLIOGRÁFICA

2.1 DSP

Os DSPs - processadores digitais de sinais - são circuitos integrados que ocupam um espaço intermediário entre os microcontroladores e os microprocessadores, pois têm os periféricos típicos de um microcontrolador, como PWM (Modulação por largura de pulso), conversor A/D (Analógico/Digital) e *timers*, porém com uma unidade lógica e aritmética reforçada (MICROCHIP, 2011). Já segundo a BDTi (2002) a diferença essencial entre um DSP e um microcontrolador está no ponto que o processador digital de sinais é mais especializados em tarefas numéricas repetitivas, também devido a possuir sequências de instruções diferenciadas, os quais permitem diversas operações serem codificadas em uma única instrução. Nesse mesmo foco de discussão Stolze e Fengler, (2008) citam que apesar dos microcontroladores modernos possuírem poder computacional suficiente no cálculo de algoritmos complexos, estes não são tão especializados, mantendo um campo mais generalista, enquanto os DSPs possuem alta performance em campos específicos de aplicação, com os mencionados autores concluindo que o dispositivo ideal deveria possuir a versatilidade de um microcontrolador com o poder de um DSP.

Dessa forma, os DSPs são dispositivos que foram projetados com o intuito de otimizar o processamento de sinais, possuindo um desempenho superior aos microcontroladores, tratando-se de rotinas intensivas e repetitivas. Na comparação com um microprocessador, o processador digital de sinais possui como vantagens o seu baixo custo, baixo consumo e tamanho reduzido, mas como desvantagem uma menor velocidade de processamento.

Segundo Piccioni e Oliveira (2002), o desempenho superior dos DSPs em relação aos microcontroladores se deve a:

- Capacidade de multiplicação e registro no acumulador em um único ciclo. DSPs de alto desempenho possuem dois multiplicadores que permitem realizar duas multiplicações ao mesmo tempo por ciclo de instrução. Alguns possuem quatro ou mais multiplicadores;
- Modos especializados de endereçamento: pré e pós-modificação dos ponteiros de endereços e endereçamento circular;
- Arquitetura de múltiplo acesso à memória, que possibilita o acesso completo a memória *on-chip* em um único ciclo;

- Instruções especiais de controle de execução, como por exemplo, instrução de *loop*, não necessitando assim instruções de atualização e teste de contadores de *loop*;
- Conjunto diferenciado de instruções, o que geralmente possibilita mais de uma operação em um único ciclo de instrução (por exemplo, em um DSP com arquitetura de 32 *bits*, 16 podem ser destinados a operações como multiplicação e adição e o resto para os dados manipulados).

Os DSPs podem ser divididos em duas famílias de processadores aritméticos de ponto fixo e de ponto flutuante, sendo que na maioria dos casos o uso de processadores de ponto flutuante resulta em soluções técnicas melhores para controladores digitais (TORRES, 2006).

No entanto, se o custo é crítico, os processadores de ponto fixo podem ser uma opção de baixo custo se o desempenho do controlador digital não for drasticamente afetado (TI, 1992).

O DSP acima de tudo é um dispositivo programável que detêm seu próprio código de instruções. Cada empresa que cria o seu processador, cria também o ambiente de desenvolvimento próprio para aquele *chip*, tornando desta forma a programação do *firmware* mais fácil e rápida (ALBUQUERQUE & NUNES, 2006).

O diagrama de blocos da Figura 1 apresenta um exemplo da visão geral de um *chip* DSP básico. Neste exemplo, o *chip* é composto por cinco blocos dominantes, sendo que os DSPs em sua maioria possuem a mesma arquitetura (TORRES, 2006):

- Processador principal;
- Porta SRAM;
- Emulador JTAG;
- Porta para comunicação externa;
- Processador do tipo I/O além de seus componentes internos.

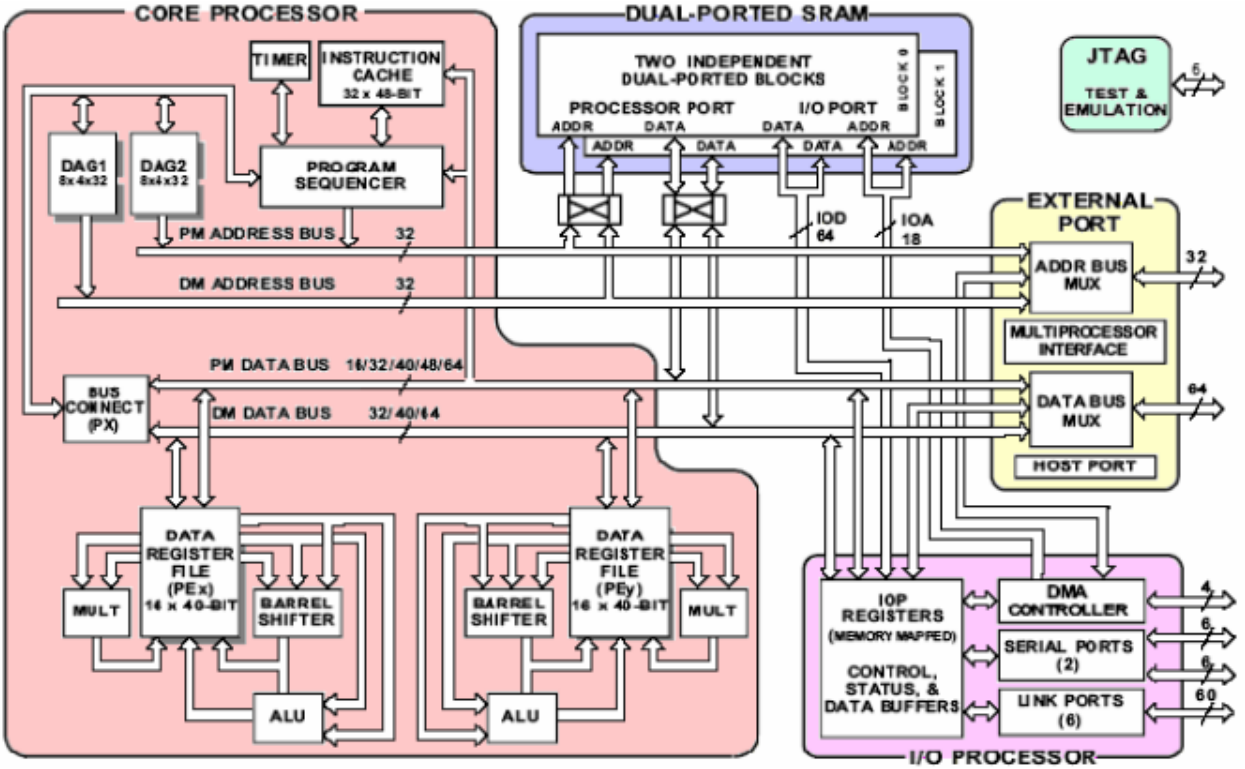


Figura 1 - Diagrama de blocos de um *chip* DSP básico
Fonte: Albuquerque e Nunes (2006).

Na parte interna ao processador principal encontram-se importantes sub-grupos. Os geradores de endereço (DAG) são responsáveis pelo fornecimento de endereçamento imediato entre a memória e os registradores. As unidades aritméticas lógicas (ALU) são as responsáveis pelas operações lógicas e aritméticas, podendo ser em ponto fixo ou ponto flutuante. O *timer* programável pode ser utilizado para realizar interrupções periódicas, ou na ativação de funções ou rotinas (ALBUQUERQUE & NUNES, 2006).

As unidades de acesso direto a memória (DMA) podem operar de forma independente ao processador principal. O DMA além de ser importante na comunicação entre a memória interna e as portas seriais, pode ser usado em programas onde é necessária a liberação do processador principal para a realização de outras rotinas (TORRES, 2006).

Um exemplo de DSP encapsulado em QFP-44 é mostrado na Figura 2:

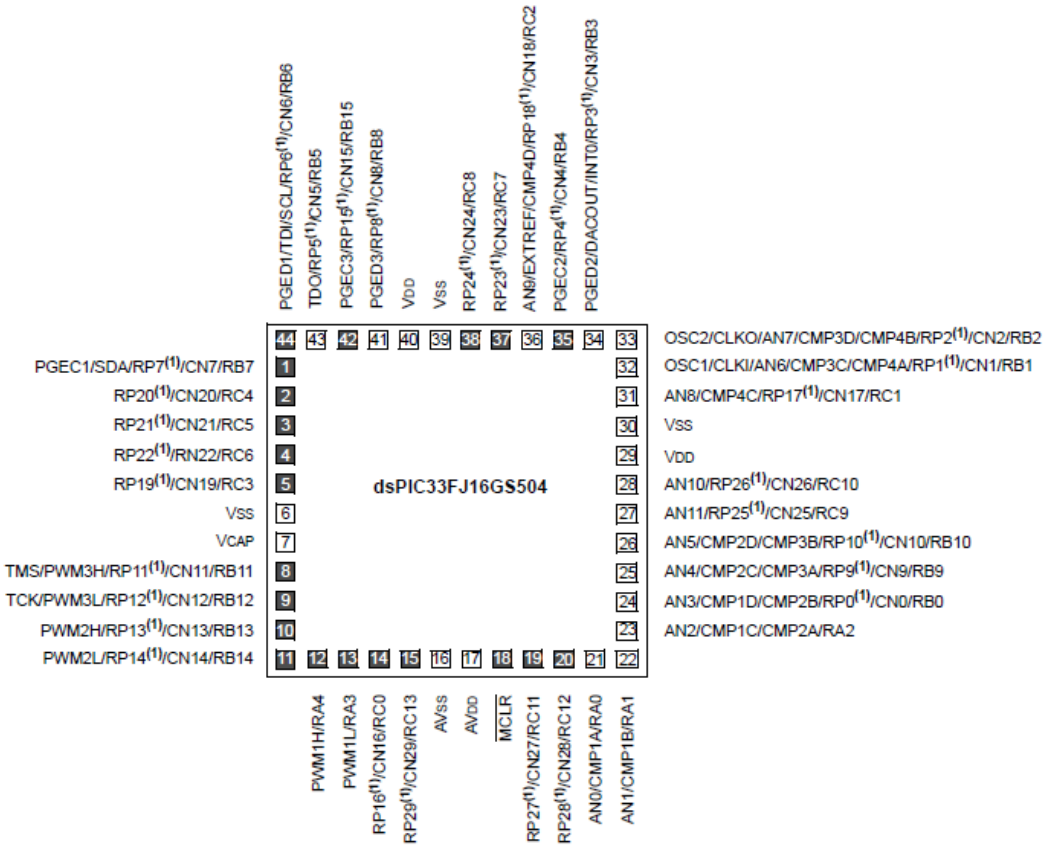


Figura 2 - Pinagem do dsPIC33F
Fonte: Microchip (2011).

A Figura 2 mostra a pinagem do microcontrolador Microchip dsPIC33F, o qual tem como características:

- 35 Pinos I/O;
- Velocidade de 40 MIPS (milhões de instruções por segundo);
- Memória *Flash*;
- 2 kbytes de RAM;
- *WatchDog Timer;*
- *Input* para oscilador externo de 4 MHz até 10 MHz com PLL ativo (x4, x8 ou x16 o valor do oscilador externo);
- Canais PWM com *timer* dedicado, possibilidade de uso em modo complementar;
- PWM com possibilidade de configuração para *center aligned* ou *edge aligned;*
- Razão cíclica individual para cada um dos oito PWMs.

2.2 PLL

O PLL é utilizado há várias décadas, sendo que dentre suas aplicações iniciais estavam a sincronia de fase em geradores, e uma ampla utilização em sistemas de telecomunicações, como a modulação e demodulação de sinais FM, e sincronia de sinais de TV (BANERJEE, 2006).

O desenvolvimento de estruturas PLL começou na década de 1930, quando a primeira aplicação considerada foi utilizada na sincronização dos sinais horizontais e verticais de um aparelho de televisão (BEST, 2007). Porém, devido aos altos custos na época, a estrutura PLL era pouco difundida, fato este que mudou na década de 1960 com a invenção dos circuitos analógicos integrados, os quais permitiam o encapsulamento da estrutura, porém ainda de forma analógica. A partir da década de 1980, com a popularização dos microprocessadores e a diminuição de preço, ocorre um maior interesse na área de desenvolvimento de estruturas PLL digitais (WANG & SUN, 2004).

Basicamente, o PLL é um sistema de controle em malha fechada que gera um sinal de saída em que a fase está relacionada com a fase de um sinal de entrada de referência.

2.2.1 Tipos de PLL

As estruturas de PLL podem ser divididas primariamente em dois grandes grupos: O PLL analógico e o PLL digital. Os PLLs analógicos têm como principal desvantagem em relação aos digitais a necessidade de utilização de componentes externos para o ajuste da malha, o que pode acarretar problemas e uma menor confiabilidade. Entre os problemas existentes podem ser mencionados a saturação dos componentes, os erros de deslocamento, também chamado de *offset*, o envelhecimento dos componentes externos, além do fator que um maior número de componentes diminui a confiabilidade do sistema (FISCHETTE, 2009). Outro problema dos PLLs analógicos frente aos digitais é ainda a necessidade de ajuste inicial da estrutura.

2.2.2 PLL Analógico ou Linear (LPLL)

O PLL Analógico ou Linear básico consiste em um elo fechado com três componentes:

Detector de Fase: fornece uma tensão de saída na qual a componente contínua é proporcional a diferença de fase entre os sinais de entrada e o sinal do oscilador controlado por tensão. O detector de fase comanda a operação dos outros blocos do PLL aumentando ou diminuindo a frequência proveniente do oscilador controlado por tensão;

Filtro: define o comportamento dinâmico do circuito, em fatores como tempo de resposta, estabilidade, fator de amortecimento, ruído de fase, entre outros;

VCO: oscilador controlado por tensão, o qual gera um sinal de frequência dependente da tensão de controle.

O circuito PLL básico é mostrado na Figura 3.

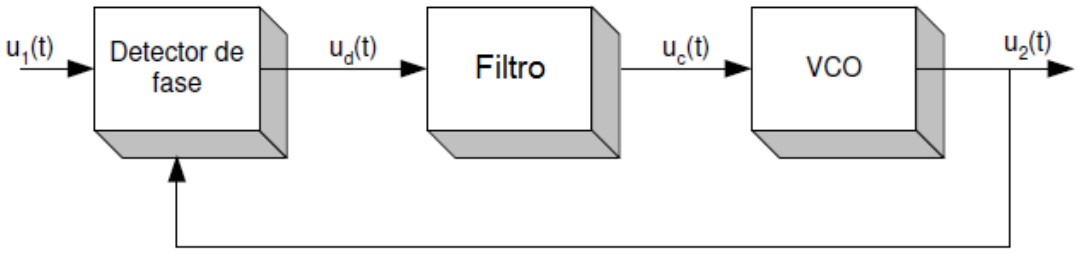


Figura 3 - Modelo simplificado de uma malha PLL
Fonte: Adaptado de Best (2007).

Nesta malha (Figura 3) o detector de fase irá gerar um sinal de erro proporcional à diferença de fase entre o sinal de entrada e a senóide gerada internamente pelo PLL. Este sinal de erro é geralmente uma correção de frequência, e quanto maior a diferença de fase entre os sinais de entrada e o sinal do PLL, maior será a correção da frequência. O filtro tem como função remover os distúrbios indesejáveis que possam afetar o comportamento do VCO além de definir o comportamento transiente ou dinâmico do sistema. Assim, o sinal $u_d(t)$ após filtrado irá controlar a frequência gerada pelo VCO, o qual basicamente efetua a síntese de uma senóide cuja frequência é diretamente proporcional a um sinal de controle. Deste modo, sucessivamente, o PLL irá sintetizar uma senóide sincronizada em fase e frequência com o sinal de entrada (BEST, 2007).

Um exemplo de aplicação para o PLL analógico é a demodulação de sinais FM, a qual pode ser visto na Figura 4. Neste caso quando o PLL está travado em um sinal do tipo frequência modulada, a tensão do controlador VCO torna-se proporcional à frequência.

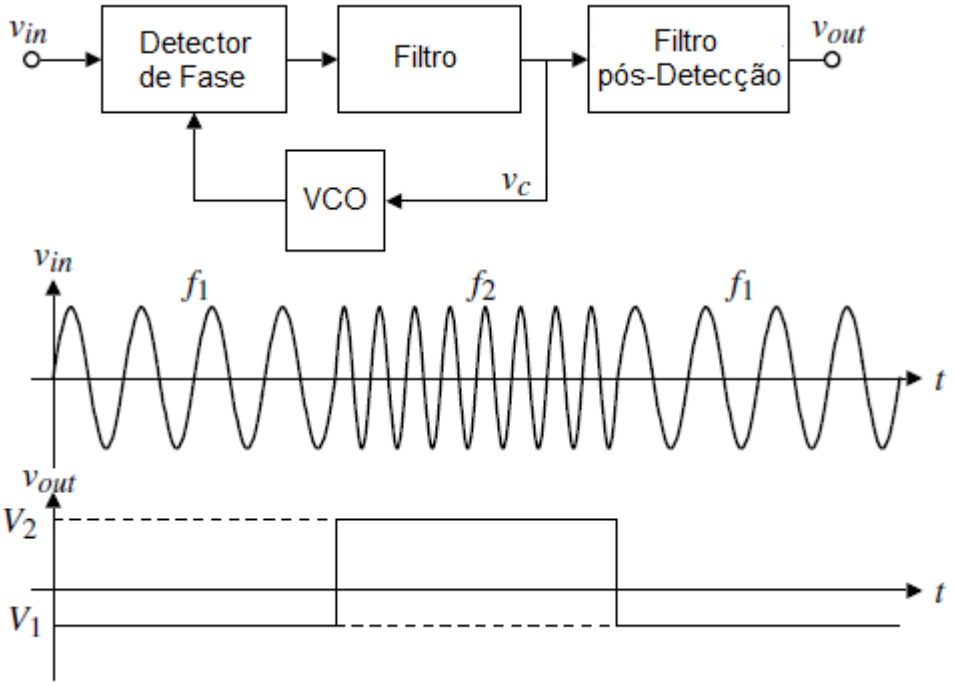


Figura 4 - Demodulação de sinal em frequência modulada
Fonte: Adaptador de Allen (2003).

Este tipo de PLL pode também ser utilizado na síntese de frequências, onde divisores colocados no elo de resposta e/ou na entrada permitem a geração de frequências baseadas numa referência estável. A Figura 5 mostra o diagrama de blocos da síntese de frequências.

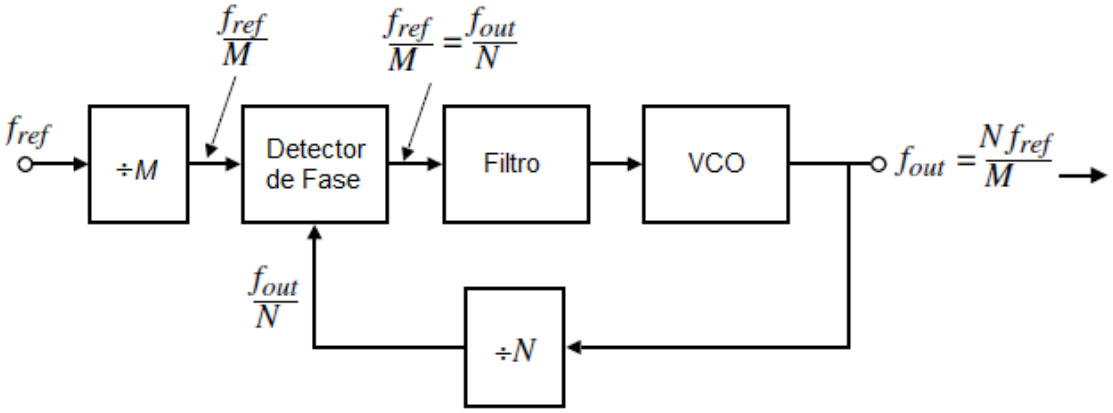


Figura 5 - Síntese de frequências utilizando PLL
Fonte: Adaptado de Wickert (2011).

Na Figura 5 quando o detector de fase está travado, as duas frequências são iguais, assim:

$$\frac{f_{ref}}{M} = \frac{f_{out}}{N} \rightarrow f_{out} = \frac{N}{M} \cdot f_{ref} \quad (2.1)$$

Um exemplo de equacionamento básico para o PLL analógico está disponível no anexo A, o qual tem como propósito demonstrar as equações necessárias à aplicação deste tipo de sistema.

2.2.3 PLL Digital (DPLL)

O PLL digital consiste em um PLL analógico com detector de fase digital, podendo esse ser XOR, detector de frequência de fase, JK, entre outros. Esta topologia pode incluir um divisor digital na malha. O DPLL é considerado um sistema híbrido, sendo muito popular em aplicações com sintetizadores (WICKERT, 2011).

A Figura 6 mostra um PLL digital clássico. A diferença deste PLL em relação ao analógico é a presença de um detector de fase digital, o qual possui características de funcionamento e equacionamento de um sistema discreto, sendo este modelado a partir das equações no espaço do tempo.

Um exemplo de equacionamento básico para o PLL digital está disponível no anexo B, o qual tem o propósito de demonstrar as diferenças entre as equações deste em relação ao PLL analógico.

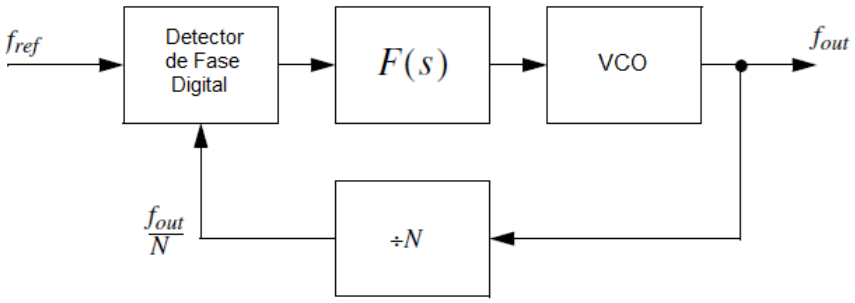


Figura 6 - DPLL clássico com detector de fase digital
Fonte: Adaptado de Wickert (2011).

2.2.4 PLL 100% Digital (ADPLL)

Nesta topologia o detector de fase, filtro e o oscilador são digitais. O ADPLL se utiliza de oscilador controlado numericamente (NCO) ao invés de um oscilador controlado por tensão. Este tipo de PLL pode ser encontrado implementado em CIs da família 74xx297. A Figura 7 mostra um exemplo de ADPLL implementado no CI 74HCT297, onde f_{ref} corresponde ao sinal de entrada da estrutura, Mf_0 é um divisor e f_{out} corresponde à saída do PLL.

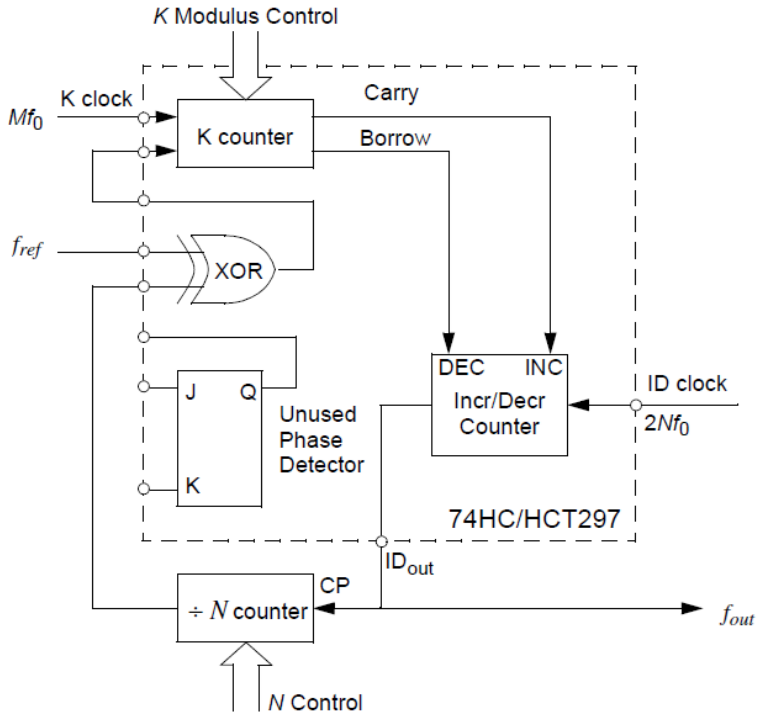


Figura 7 - ADPLL implementado no CI 74HCT297
Fonte: Adaptado de TI (2003).

Entre as aplicações possíveis do ADPLL estão o condicionamento e a filtragem de sinais. Através da aplicação do PLL, este pode operar como um filtro de forma a selecionar um específico sinal desejado entre a presença de sinais indesejados. A Figura 8 mostra o exemplo de condicionamento de sinal.

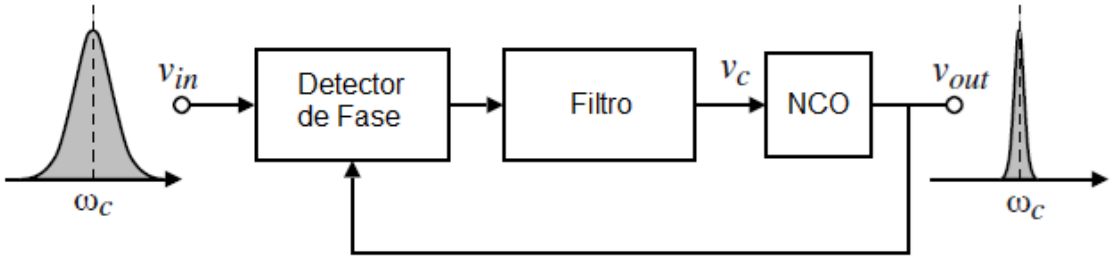


Figura 8 - Condicionamento de sinal
Fonte: Adaptado de Allen (2003).

O condicionamento de sinais possui uma relação entre a faixa de captura possível do PLL e a largura de banda na aquisição do sinal, sendo que quanto maior a largura de banda, maior será a faixa possível de captura do sinal, e consequentemente será melhor a resolução na busca pelo sinal desejado.

2.2.5 PLL por *Firmware*

No PLL por *firmware* os blocos funcionais são implementados em *software* ao invés de um *hardware* dedicado. É considerado um sub-grupo que ganha destaque nos PLLs digitais, sendo implementado principalmente em microprocessadores e microcontroladores nos últimos 20 anos (FISCHETTE, 2009).

O PLL por *firmware* se utiliza de um algoritmo de implementação, onde o sinal de entrada é discretizado, e toda a detecção de fase, a filtragem, e o oscilador são determinados em um único algoritmo utilizando-se as interrupções do DSP.

Um exemplo da utilização do PLL por *firmware* se dá em casos de recuperação de dados e *clock*, produzindo um sinal estável a partir de possíveis tremulações (*jitter*). A recuperação de *clock* consiste em basicamente duas funções: detecção de borda e a geração de uma saída periódica estável. A Figura 9 mostra o diagrama de blocos da aplicação citada.

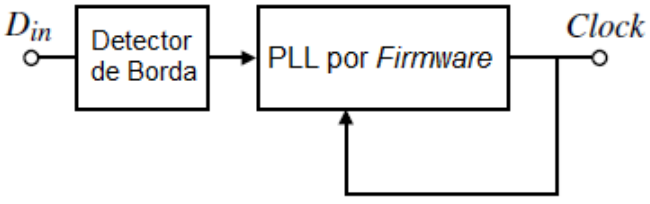


Figura 9 - Recuperação de *clock*
Fonte: Adaptado de Wickert (2011).

Em sistemas de comunicação digitais, os dados transmitidos ou recebidos podem sofrer de erros de tremulação. O circuito de recuperação de *clock* pode ser usado pra regenerar o sinal e remover a tremulação como mostrado na Figura 10.

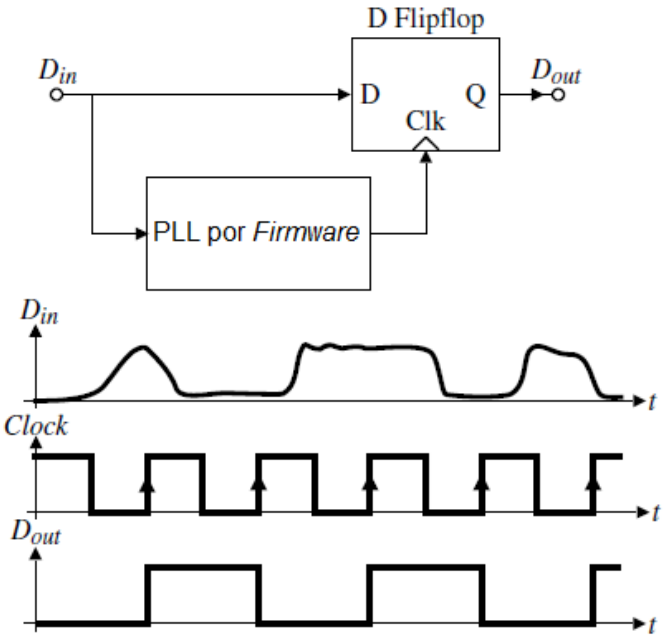


Figura 10 - Remoção de tremulação utilizando PLL por *firmware*
Fonte: Adaptado de Allen (2003).

2.2.6 Exemplos de algoritmos em PLL por *firmware*

Uma das possibilidades de desenvolvimento de um algoritmo de PLL por *firmware* consiste na comparação de uma senóide de referência, esta sendo uma amostra da senóide da rede com um *offset* aplicado e uma tabela com os pontos de uma senóide definida internamente ao DSP. O desenvolvimento deste algoritmo conta com algumas diretrizes principais. A primeira é que a senóide de entrada necessita de um *offset* na sua tensão de entrada para que a mesma possa ser lida pelo DSP, o qual trabalha somente com valores positivos de tensão de leitura. No caso de um DSP que opere com leitura nas suas portas I/O entre 0 e 3,3V, um *offset* de 1,65 V pode ser aplicado, de forma a centralizar o ponto de passagem da senóide neste valor. Assim, caso a onda senoidal de entrada possua uma amplitude de 2 V, esta irá variar entre 0,65 V e 2,65 V.

A segunda diretriz é que a senóide é uma onda simétrica, com isto pode-se dividir a mesma em quatro quadrantes, como pode ser visto na Figura 11.

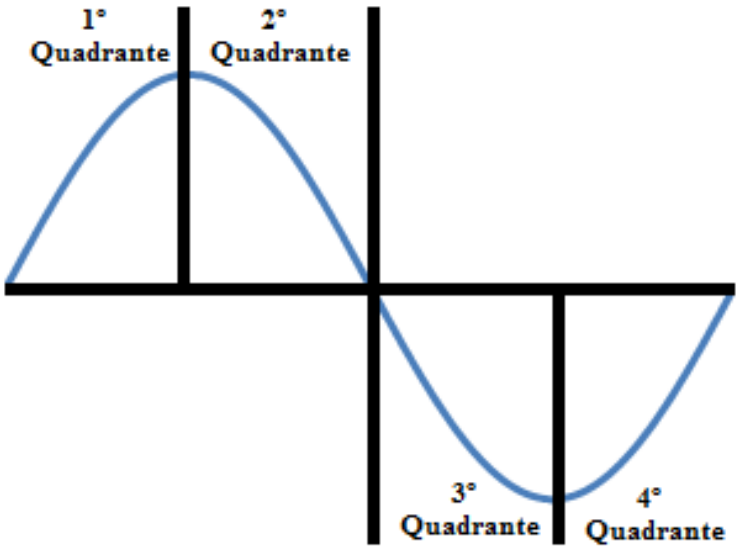


Figura 11 - Quadrantes da senóide
Fonte: Autoria Própria

Visando a economia de memória do dispositivo e aproveitando-se desta simetria, apenas o primeiro quadrante precisa necessariamente ser definido no DSP, com os outros calculados a partir deste.

A terceira premissa é que a senóide da rede pode variar em uma certa frequência, sendo este valor a ser determinado no algoritmo, o qual por exemplo pode ser escolhido como 3 Hz. Assim, a senóide pode variar entre 57 Hz e 63 Hz no *lock range* final. O controlador considera essa variação e caso a onda de saída esteja neste intervalo será considerado que esta se encontra em fase com a rede elétrica.

A quarta e última diretriz é a delimitação de valores máximos e mínimos de frequência, caso algum destes valores sejam ultrapassados o controlador deve interromper o chaveamento, pois é considerado, que ou a rede ou o controlador estão com seu funcionamento normal comprometidos.

Tendo delimitado estes parâmetros o algoritmo pode ser desenvolvido. Seu funcionamento consiste basicamente na identificação da passagem da senóide da rede por zero. Após esta identificada, a passagem do próximo valor medido pelo conversor A/D é comparado com o valor da senóide interna ao controlador. É também a partir deste ponto que é determinado o quadrante da senóide, pois caso este valor seja maior que o *offset* a rede está no primeiro quadrante, caso contrário, a rede estará no terceiro quadrante.

A frequência do sinal de saída só é alterada caso esta seja inferior a 57 Hz, quando o período deve ser decrementado a cada ciclo, assim aumentando-se a frequência, ou caso passe

de 63 Hz quando o processo contrário deve ser realizado. Os valores mínimo e máximo de funcionamento do programa podem, por exemplo, ser estabelecidos entre 45 Hz e 90 Hz, e no caso da ultrapassagem destes, o processo de chaveamento é interrompido. Este só é retomado caso o controlador identifique que a rede está funcionando normalmente e após uma reinicialização do programa.

Este funcionamento pode ser melhor observado com o auxílio do gráfico da Figura 12. Nele estão representadas as senóides com frequência mínima em vermelho, e a frequência máxima em azul. O intervalo ideal de funcionamento é entre as ondas roxa e amarela, que apresentam uma frequência de 63 Hz e 57 Hz respectivamente.

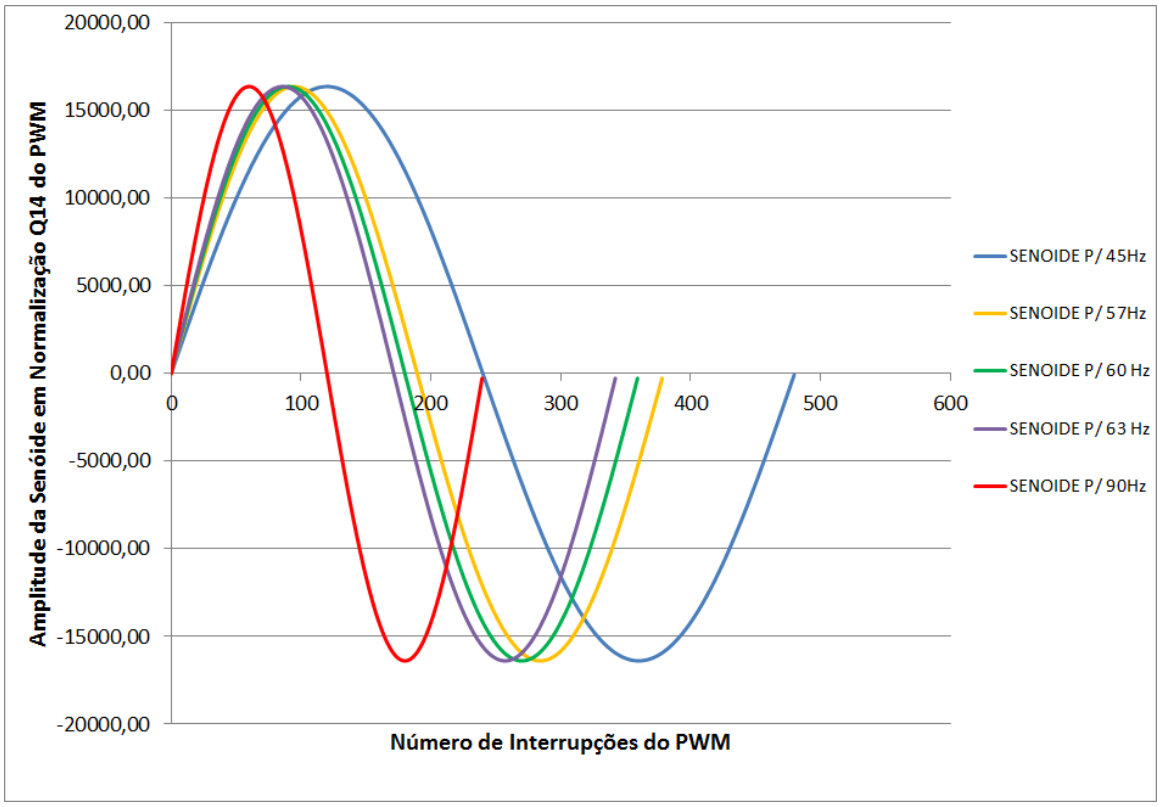


Figura 12 - Exemplo de *lock range* e *capture range* no PLL em *firmware*
Fonte: Autoria Própria.

No gráfico (Figura 12) caso a onda da senóide esteja em valor inferior a 57 Hz, deve-se aumentar a frequência, fato este que é ajustado ao decrementar o período da senóide e após isso efetuar o *set* de *flag* de fase. Caso a onda da senóide esteja em valores superiores a 63 Hz, o período da senóide deve ser incrementado. A frequência pode ser verificada por um contador de interrupções do PWM, e comparando o valor do contador a cada passagem por zero pode ser verificado se o valor é maior ou menor que o registro do período do PWM.

Neste exemplo de implementação, para uma frequência do PWM de 20 kHz tem-se um período de 50 µs, e a normalização do PWM em Q14 corresponde ao valor de 2^{14}. Estes dados complementam o caso do desenvolvimento da Figura 12, tal que para 400 pontos estabelecidos em uma tabela senoidal equivalentes a 60 Hz, o período para 90 Hz corresponde a 266 aquisições, enquanto o período para 45 Hz corresponde a 533 aquisições. Assim, a fase da onda gerada em relação à onda de entrada é atingida justamente quando o contador implementado no algoritmo coincidir com o período da senóide de entrada, onde para a verificação de sincronismo o contador de aquisições é utilizado. A partir do momento em que o sincronismo é atingido, o período do contador do PWM não necessita ser incrementado ou decrementado.

A Figura 13 demonstra o fluxograma do algoritmo citado, onde estão os passos mínimos necessários para a implementação deste em um DSP. Várias sub-rotinas podem ser adicionadas ao algoritmo, como por exemplo, a soma ou decremento de duas unidades do período para ajustar o PLL mais rapidamente. Outro cuidado a ser tomado é em relação ao número de interrupções do PWM, pois um número muito alto pode sobrecarregar as rotinas de forma a afetar o funcionamento do DSP e até mesmo trava-lo. Neste caso é interessante a implementação de uma rotina de verificação do estado do algoritmo.

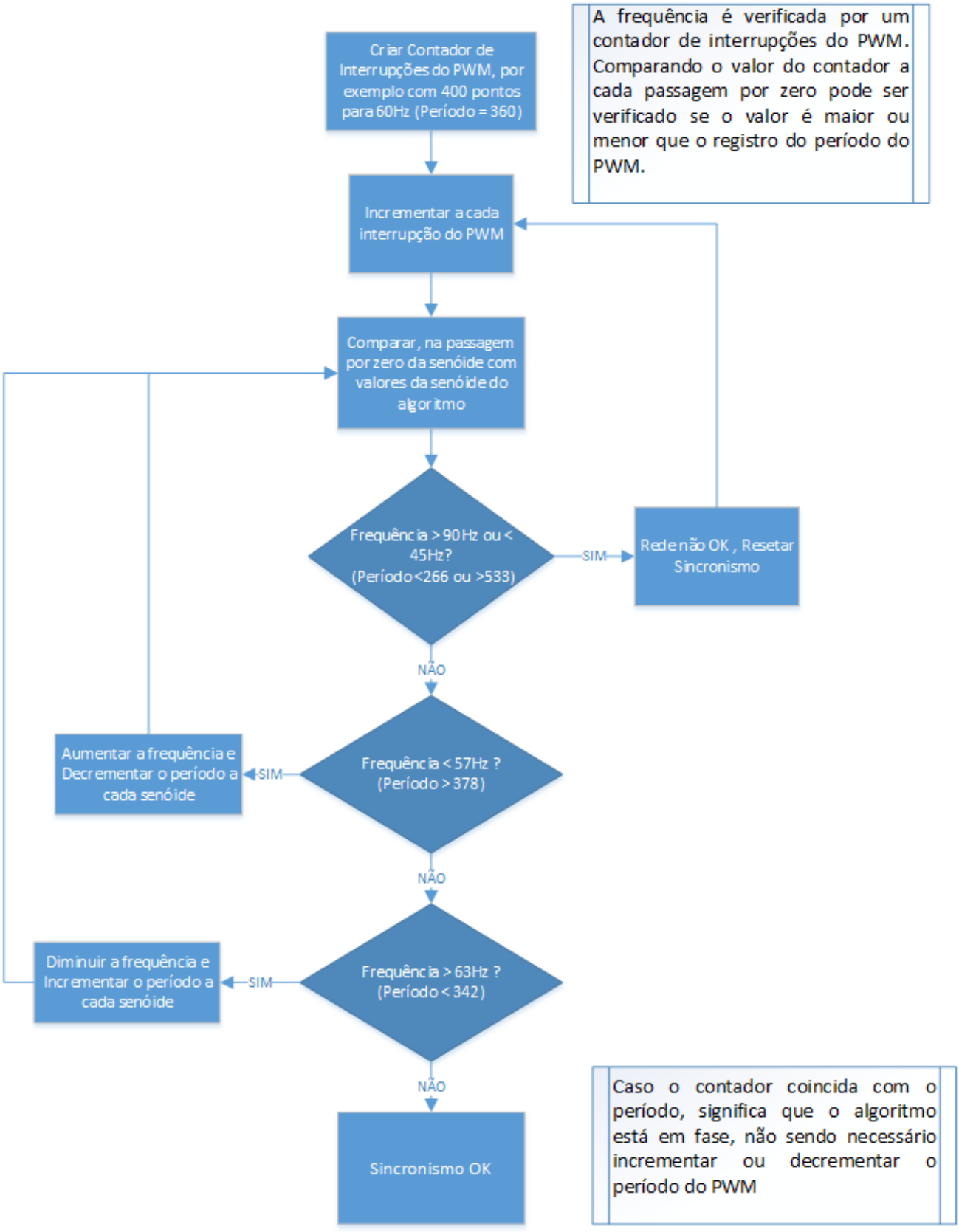


Figura 13 - Fluxograma do algoritmo do PLL em *firmware*
Fonte: Autoria Própria.

Outra possibilidade de implementação de algoritmo PLL em *firmware* consiste na inclusão de um contador no DSP com, por exemplo, 360 pontos iniciando em 0 e com valor final de 359, os quais são os ângulos da senóide e base para utilização em todo o restante do *firmware*. Nesse caso o teste para gerar a onda quadrada de sincronia se configura em testar se os valores do contador no DSP estão em valor inferior a 180.

Indexado por esse contador existe uma tabela de cossenóide, a qual é multiplicada em toda a interrupção do *timer* pela tensão de entrada, que tem uma fundamental senoidal de 60 Hz. Assim é feito o cálculo médio dessa correlação – dividindo a somatório por 360, resultando assim no detector de fase - onde o resultado é zero caso o contador que indexa a tabela estiver em fase com a rede. Se este não estiver em fase, o resultado dessa operação irá gerar valores positivos ou negativos, sendo esses alimentados em um PI, que altera a frequência do *timer* onde o contador indexador do DSP é incrementado e são feitas as leituras de tensão de entrada.

Este controlador PI (Proporcional Integral) é implementado diretamente no algoritmo através de linhas de código e tem como função procurar zerar a defasagem entre as ondas de referência e a onda de saída do algoritmo.

Por exemplo, para 360 pontos de uma senóide em 60 Hz, o *timer* tem uma interrupção de aproximadamente 46 μs, este sendo o *timer* o qual tem o período alterado para buscar o sincronismo. Da mesma forma que a rede, esse controlador PI está em 60 Hz.

A Figura 14 mostra um exemplo de simulação matemática desse algoritmo PLL (sem incluir o compensador PI) com uma das possibilidades de senóide fora de sincronismo. Nesse caso o defasamento inicial simulado é de -0,5 em radianos, com os pontos da senóide estabelecidos, e a partir disso é efetuado o cálculo do valor médio com a integração da senóide e cossenóide.

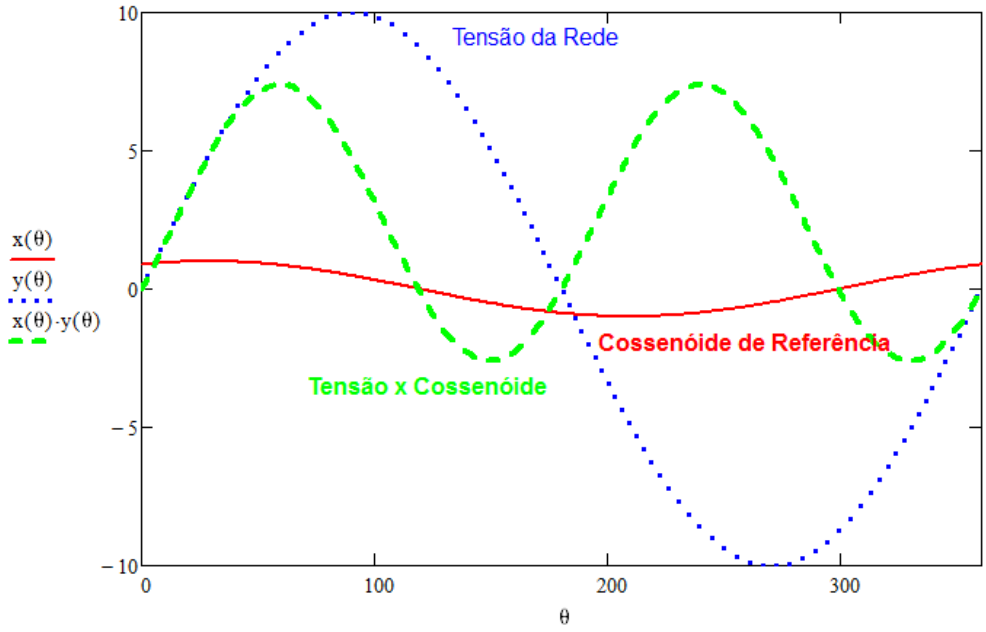


Figura 14 - Exemplo de simulação do algoritmo com defasamento em -0,5 radianos
Fonte: Autoria Própria.

Na Figura 14, a onda em cor vermelha (linha contínua) corresponde a cossenóide com uma pequena amplitude e a defasagem já citada, enquanto a onda azul (linha pontilhada) corresponde à tensão da rede, enquanto a onda verde (linha tracejada) remete ao valor médio da integração da multiplicação da senóide pela cossenóide, na qual o nível médio depende diretamente da defasagem, sendo esta a onda que sofre a alteração no algoritmo. Neste exemplo o valor médio calculado foi maior que zero, e este depois de filtrado passa por um controlador PI que tenta mantê-lo em zero, conforme já explicado.

No caso desse algoritmo possuir um defasamento de +0,5 Radianos, e considerando-se as mesmas premissas da figura anterior, o valor médio calculado é inferior à zero, o que faz o controlador PI atuar na busca do sincronismo. A Figura 15 expõe este caso.

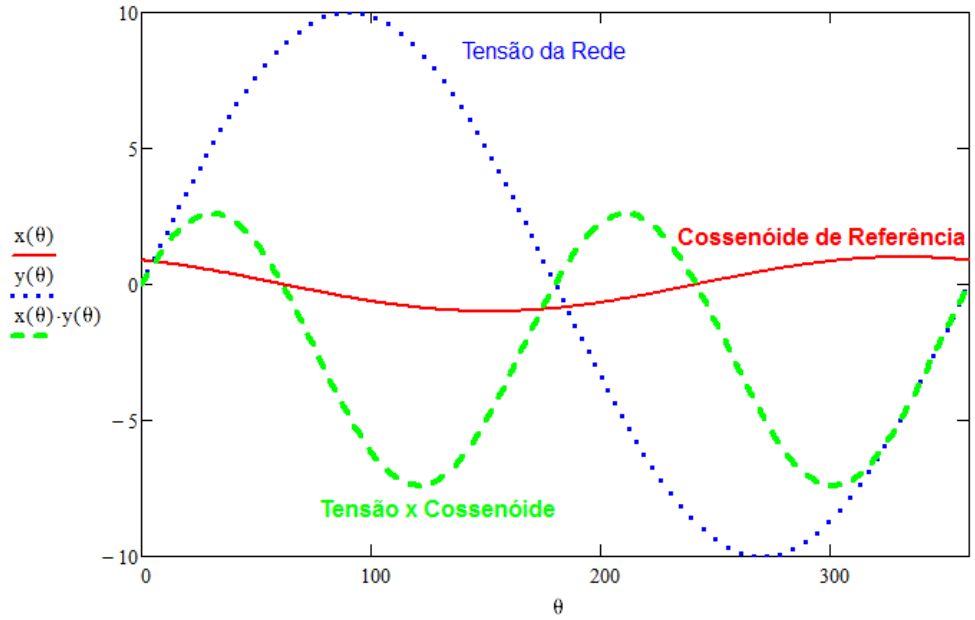


Figura 15 - Exemplo de simulação do algoritmo com defasamento em +0,5 radianos
Fonte: Autoria Própria.

Porém no caso do sincronismo eficaz do PLL com a rede, o cálculo do valor médio se torna nulo, o que assegura que a senóide gerada sinteticamente está em fase com a senóide de referência, fato esse evidenciado na Figura 16, a qual mostra a simulação matemática – sem o compensador PI - desse algoritmo em sincronismo.

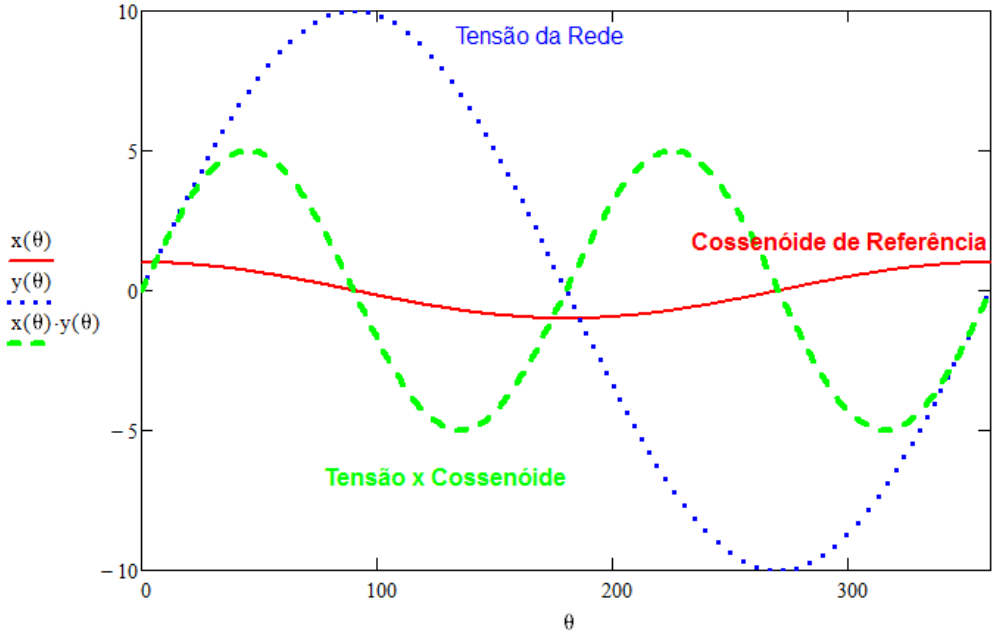


Figura 16 - Exemplo de simulação do algoritmo no caso de sincronismo
Fonte: Autoria Própria.

2.2.7 Parâmetros de desempenho

O PLL pode ser implementado com uma grande variedade de indicadores de desempenho, os quais simplificadamente são citados a seguir:

- Tipo e Ordem;
- *Lock Range*: O *range* de frequência que o PLL é apto a estar travado em sincronismo, valor este definido principalmente pelo VCO;
- *Capture Range*: O *range* de frequência que o PLL é apto a estar sincronizado, partindo de uma condição fora de sincronia;
- A velocidade da malha de controle;
- Resposta transiente: *Overshoot* e tempo de estabilização;
- Erros de temporização;
- Ruídos de fase;
- Parâmetros gerais: Consumo, amplitude de saída, entre outros.

2.2.8 Vantagens do PLL *Firmware* em relação ao PLL Analógico

O processamento analógico de sinais em um PLL analógico é feito através do uso de componentes analógicos, tais como:

- Resistores;
- Capacitores;
- Amplificadores operacionais.

As tolerâncias inerentes desses componentes podem afetar a eficiência do circuito apenas com variações de temperatura, tensão e vibrações mecânicas, além de componentes analógicos serem mais suscetíveis a ruídos, e possuirem um consumo de potência geralmente superior dado a um maior número de componentes em comparação ao PLL por *firmware*, além de gerar uma menor confiabilidade geral no sistema (PELLENZ, 2005).

Outro fato pertinente é a maior quantidade de elementos no circuito em um sistema de sincronia com a rede elétrica utilizando-se um PLL Analógico no lugar de um PLL por *firmware*, sendo que este somente necessita do sinal de entrada da rede dentro dos parâmetros de leitura enquanto um sistema analógico se utiliza de vários comparadores, além de componentes ativos e passivos. Pode-se atribuir a menor quantidade de componentes no sistema de sincronismo digital ao fato que todo o PLL, os filtros e compensadores são programados diretamente no DSP.

Porém uma desvantagem existente ao PLL por *firmware* é a limitação da frequência em que este apto ao sincronismo, a qual depende diretamente da frequência da aquisição do sinal pelo DSP (PELLENZ, 2005).

2.3 CONCLUSÕES

Neste capítulo, foram abordados os aspectos técnicos necessários para a implementação de um algoritmo PLL para sincronismo em conversores estáticos. Inicialmente, foi feita uma breve explicação a respeito dos microcontroladores e DSPs, os quais serão suporte fundamental para este projeto, seguindo-se pela explanação das possíveis topologias de PLL - estas divididas primariamente em analógicas e digitais – e, finalmente, foram expostas soluções para a implementação do sincronismo digital.

3 DESENVOLVIMENTO DO PROTÓTIPO

3.1 PREMISSAS PARA O PROJETO DO PROTÓTIPO E SINCRONISMO

O objetivo inicial desse trabalho é a implementação de um algoritmo de sincronismo em um DSP de forma a efetuar o sincronismo em frequência e fase com a rede elétrica, para tanto foi adotado como *capture range* as frequências entre 45 Hz e 90 Hz com o *lock range* estabelecido para frequências entre 47 Hz e 66 Hz nos testes iniciais, com a otimização final do *lock range* situando-se entre 57 Hz e 63 Hz devido a rede elétrica estar centrada em 60 Hz.

O formato de onda da rede em que o protótipo será testado é considerado uma senóide pura, com leves variações na sua frequência, possuindo uma tensão de pico de aproximadamente 180 V. A frequência de chaveamento do PWM da senóide gerada sinteticamente pelo algoritmo será fixada em um valor de 24 kHz, valor este adotado por não ser captado pelo ouvido humano.

O protótipo será testado junto à rede elétrica, porém não estará conectado diretamente a esta no caso de fornecimento de energia, pois no caso de conexão à rede, a amplitude do sinal de saída deve estar em valores similares a esta, o que foge do objeto de estudo deste trabalho.

3.2 MONTAGEM DO PROTÓTIPO

O protótipo montado possui uma variedade de circuitos que podem ser demonstrados no diagrama de blocos da Figura 17. Estes circuitos tem como função a alimentação dos circuitos integrados e do DSP, a aquisição e condicionamento do sinal da rede, o processamento do sinal junto ao algoritmo PLL e o chaveamento do conversor montado para os testes.

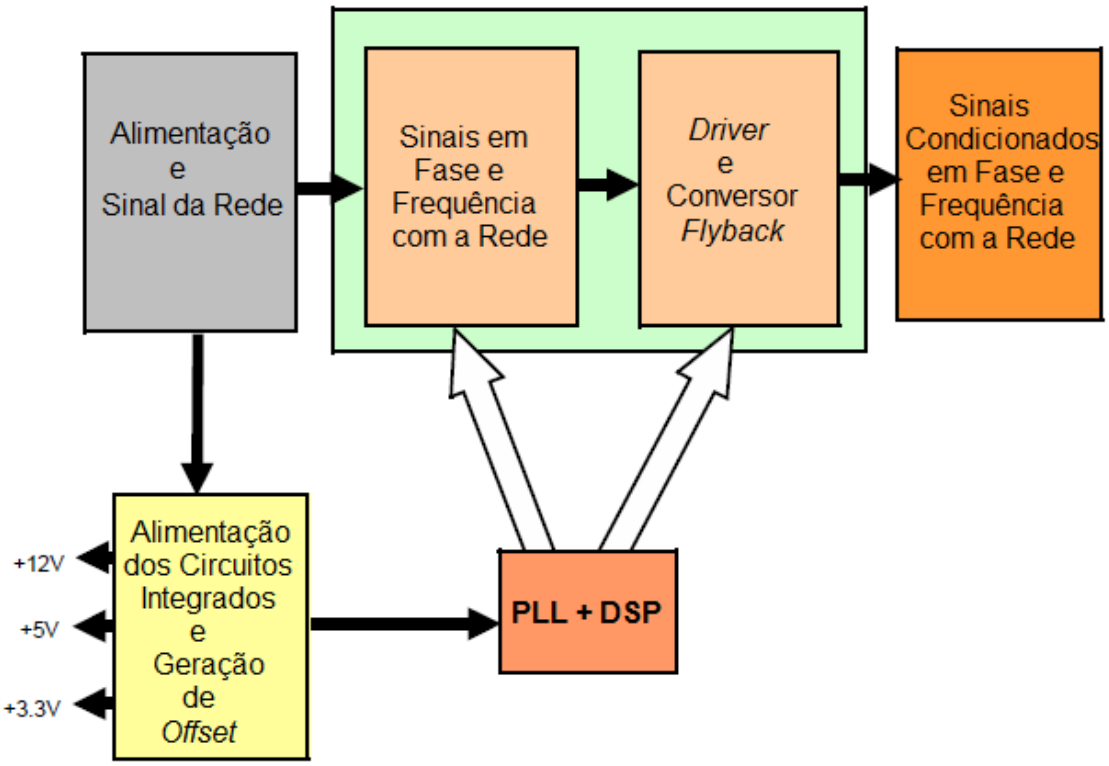


Figura 17 - Diagrama de blocos da montagem do protótipo
Fonte: Autoria Própria.

3.2.1 Circuitos de alimentação dos circuitos integrados e DSP

O protótipo é alimentado a partir de uma fonte que deve fornecer tensão e corrente em valores adequados, na qual o valor de 15 V foi escolhido, pois este é suficiente para alimentar todos os circuitos integrados além dos outros componentes ativos e passivos. Esta tensão também possibilita alimentar as chaves no caso de teste com a rede elétrica.

A alimentação dos circuitos integrados é proveniente do circuito mostrado na Figura 18, onde o LM7805 é um regulador linear, que fornece a tensão de 5 V para os circuitos integrados, enquanto o LM317 com o ajuste dos resistores R_{130} e R_{131} fornece 3,3 V para a alimentação do DSP.

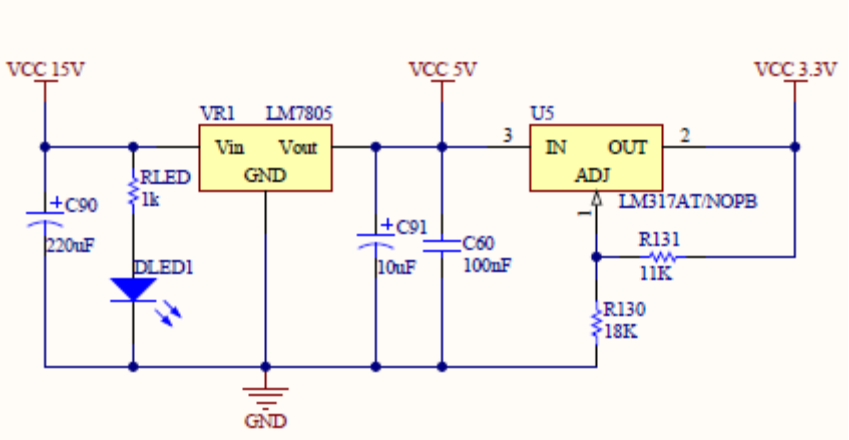


Figura 18 - Circuito para alimentação dos CIs e DSP
Fonte: Autoria Própria.

3.2.2 Circuito de condicionamento da senóide e *offset*

Em um primeiro momento, o PLL necessita de uma referência de tensão e frequência da rede para que o mesmo estabeleça uma relação entre o sinal de entrada coletado e possa fornecer um sinal de saída em fase e frequência com o sinal da rede.

Para isso é necessário um circuito que forneça uma amostra do sinal da rede, e que este sinal esteja condicionado à leitura pelo DSP. Assim deve-se atenuar o sinal senoidal da rede a níveis propícios para a leitura do mesmo pelas entradas analógicas do DSP. Uma das soluções possíveis é o uso de um transformador 127-12 V_{CA}, o qual além de diminuir a amplitude da tensão da rede, funciona como isolamento entre a rede e o circuito de controle.

Outro ponto importante é que o DSP deve ler toda a senóide da rede quando se está utilizando uma estrutura de controle de PLL por *firmware*, ou seja, deve-se aplicar um *offset* no sinal da rede para que ambos os semiciclos positivos e negativos da senóide estejam acima da tensão de 0 V e inferiores à tensão máxima suportada pela entrada analógica do DSP, nesse caso, 3,3 V.

A Figura 19 demonstra tal circuito de condicionamento, com o amplificador operacional na configuração de seguidor de tensão fornecendo um *offset* de 2,5 V, valor este devido à escolha dos resistores R_8 e R_9 os quais possuindo a mesma resistência e conectados ao pino 3 do CI fazem com que a tensão de saída seja metade da entrada, pois na alimentação deste foi utilizada a tensão de 5 V. Como a saída do transformador é em 12 V_{CA}, se faz necessário o uso dos resistores R_{120} e R_{121} como divisores de tensão, de forma a atenuar a tensão deste para aproximadamente 1,2 V_{CA}, e da mesma forma são utilizados os resistores R_{122} e R_{123} em outro divisor de tensão de forma a limitar a tensão em valores que o DSP pode

efetuar a leitura, no caso até 3,3 V, com o *offset* da senóide nesse caso em 1,65 V, após o uso dos resistores R_{122} e R_{123}.

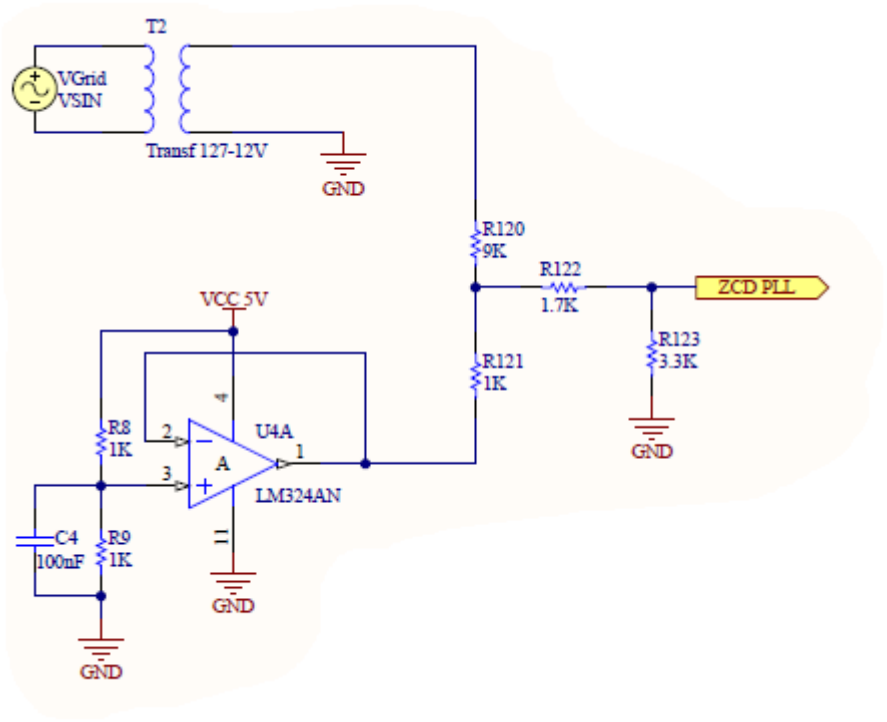


Figura 19 - Circuito utilizado para gerar um sinal de referência ao PLL
Fonte: Autoria Própria.

Os circuitos das Figuras 18 e 19 foram montados na mesma placa padrão e são demonstrados na Figura 20.

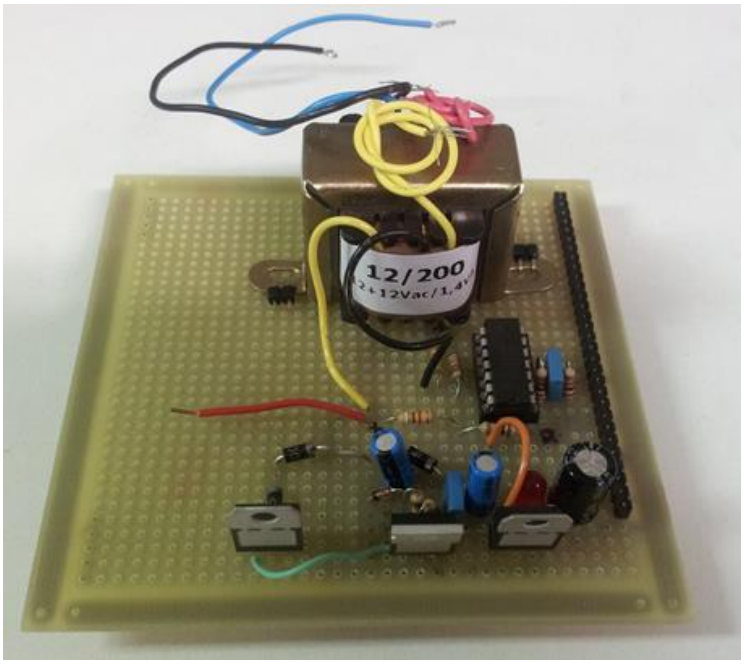


Figura 20 - Circuitos de alimentação e condicionamento montados em placa padrão
Fonte: Autoria Própria.

3.2.3 Circuito de gravação do DSP

Para a gravação e testes de algumas partes do algoritmo foi criado e utilizado um circuito especificamente destinado a esse fim, sendo este também necessário para que o DSP pudesse ser gravado *on-board*. O DSP escolhido neste projeto tem como peculiaridade a necessidade de ser alimentado por uma fonte externa até no momento que está sendo gravado, neste caso por um gravador do tipo ICD2 da Microchip junto ao circuito de gravação montado. O esquemático do circuito de gravação do DSP é mostrado na Figura 21.

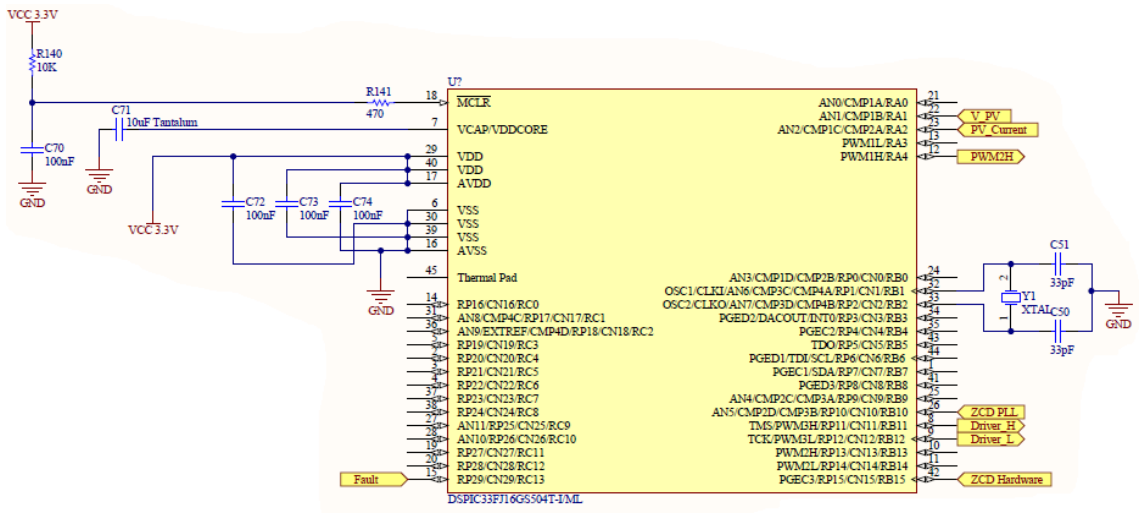


Figura 21 - Circuito de gravação e testes do dsPIC33F
Fonte: Autoria Própria.

Outro fato importante é que devido ao encapsulamento do DSP escolhido, foi necessário o desenvolvimento da placa da Figura 21, a qual passa de um formato QFP para o DIP normalmente usado nas placas padrão, e que foram a base da montagem do protótipo. O circuito de gravação montado em placa padrão é visualizado na Figura 22.

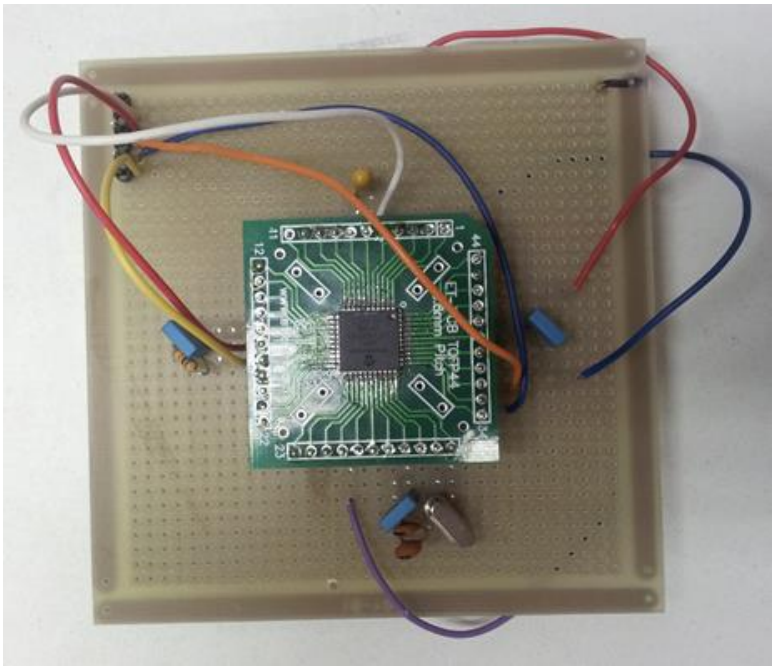


Figura 22 - Circuito de gravação do DSP montado em placa padrão
Fonte: Autoria Própria.

3.2.4 Circuitos de sincronismo

O controle de sincronismo é basicamente implementado pela comutação das chaves de sincronismo – comandadas pelos PWMs - que estão em fase e frequência com o sinal de entrada, além da geração da senóide sintética seguindo o algoritmo PLL implementado no *firmware* no DSP. Um exemplo de possível topologia de circuito onde um sistema de sincronismo pode ser aplicado é mostrado na Figura 23.

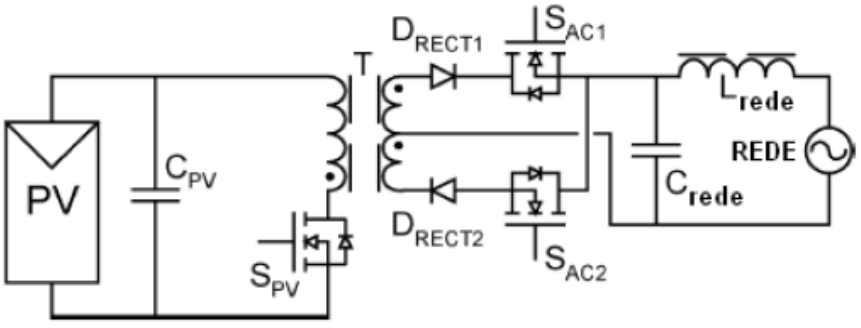


Figura 23 - Topologia de conversor a qual pode ser aplicado um sistema de sincronismo
Fonte: Adaptado de Kasa e Iida (2002).

No circuito da Figura 23 pode ser visualizado um exemplo de inversor, que pode processar a energia proveniente de um painel fotovoltaico e conecta-se a uma rede elétrica monofásica.

Neste exemplo PV corresponde a um painel fotovoltaico, $D_{RECT1,2}$ aos diodos retificadores, L_{REDE} e C_{REDE} à filtragem LC, S_{PV} corresponde a chave que da senóide gerada pelo algoritmo PLL, enquanto S_{AC1} e S_{AC2} são as chaves de sincronismo com a rede comandadas pelos PWMs a partir do algoritmo PLL.

Para este exemplo de circuito, os formatos de onda esperados são apresentados na figura 24.

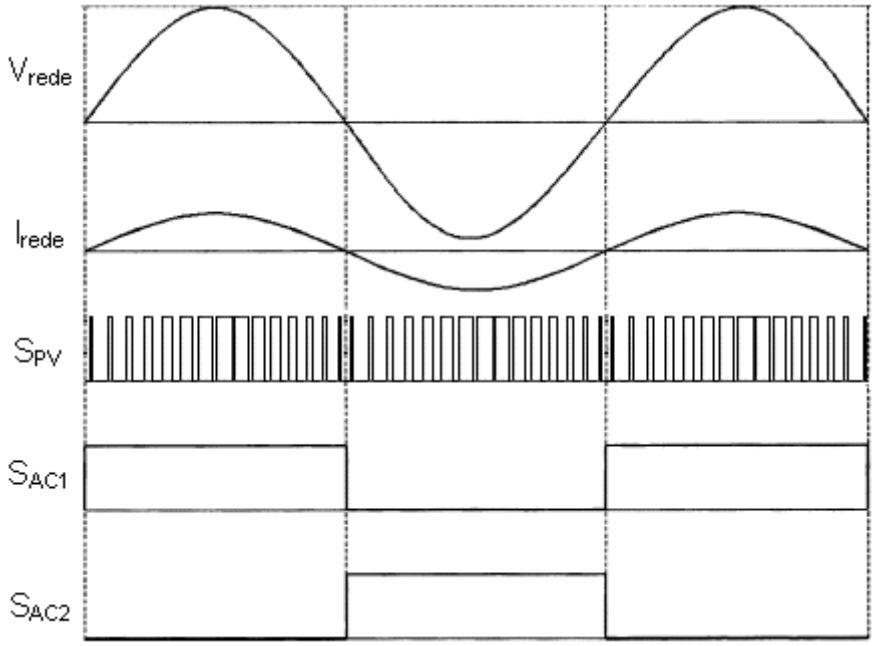


Figura 24 - Formatos de onda em sistema de sincronia com a rede elétrica
Fonte: Adaptado de Kasa e Iida (2002).

Na Figura 24, por meio do controle das chaves S_{AC1} e S_{AC2} que é feita a sincronia com a rede, onde no semiciclo positivo a chave S_{AC1} permanece aberta enquanto a chave S_{AC2} permanece fechada, enquanto no semiciclo negativo a chave S_{AC1} está fechada e a chave S_{AC2} é aberta. Através dessa comutação é possível o sincronismo. O PWM da chave S_{PV} também estará sincronizado com a rede, de forma a gerar a senóide sintética sincronizada em fase e frequência com a rede elétrica.

3.2.5 *Driver* de acionamento das chaves de sincronismo

A tensão de saída dos pinos dos PWMs - os quais controlam as chaves de sincronismo - é insuficiente para acionar os MOSFETs caso se deseje testar o sistema de sincronismo utilizando propriamente um conversor em conexão à rede elétrica, o que faz necessário a utilização de um circuito de interface entre os PWMs de controle e as chaves de potência. Foi utilizado neste projeto um circuito *Driver* baseado no circuito integrado HCPL-316J, o qual é isolado opticamente e possui saídas de *fault* no caso da ocorrência de algum erro por sobretensão, subtensão ou proteção por saturação da tensão V_{DS} do MOSFET. O esquemático simplificado do *driver* e o seu circuito de acionamento são apresentados na Figura 25, onde PWM *High* e PWM *Low* correspondem aos sinais de sincronismo de entrada, *Fault* ao sinal de algum erro no circuito de potência e *Reset* ao sinal necessário para que o *driver* tenha seu funcionamento regularizado após algum erro.

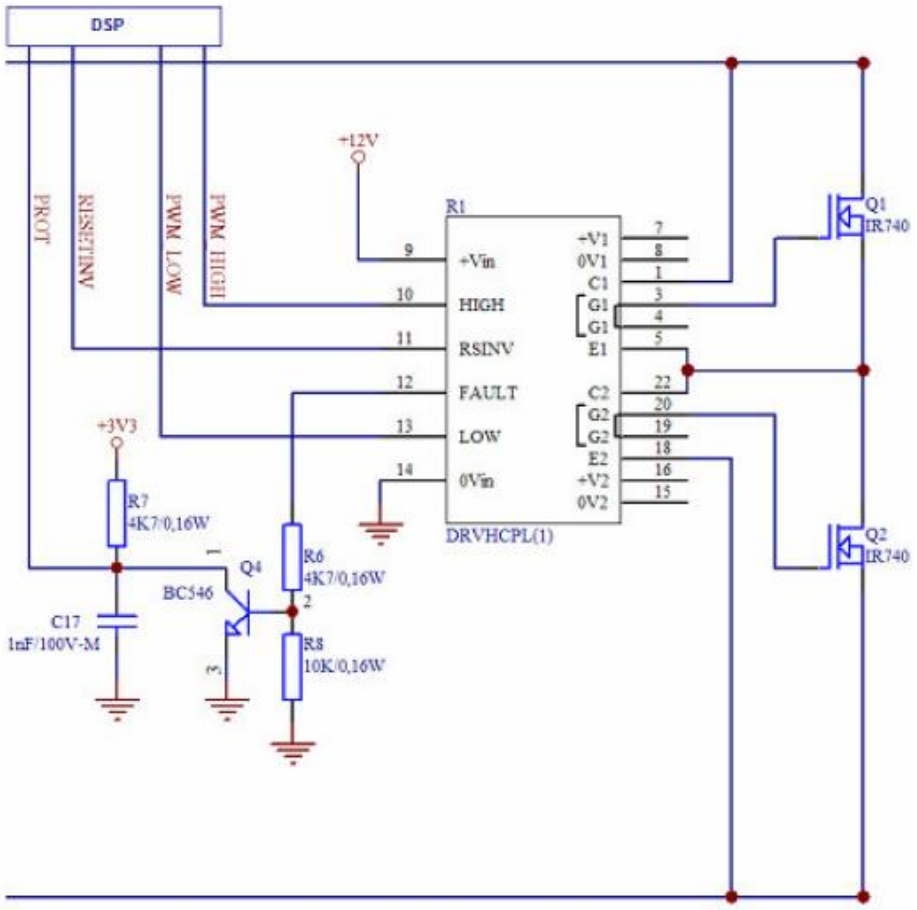


Figura 25 - Esquemático de interface entre o DSP e o acionamento das chaves de sincronismo S_{AC1} e S_{AC2}
Fonte: Avago (2011).

3.2.6 *Driver* de acionamento da chave da senóide de sincronismo

Da mesma forma que os PWMs de sincronismo, o PWM da senóide de saída necessita de um circuito *driver* caso se deseje fazer o acionamento de uma chave de sincronismo. Dessa forma foi implementado um circuito composto por um transistor PNP e três transistores NPN com o intuito de amplificar o sinal de 3,3 V proveniente do dsPIC para o valor de 15 V compatível com o acionamento da chave da senóide de sincronismo. Tal circuito é apresentado na Figura 26.

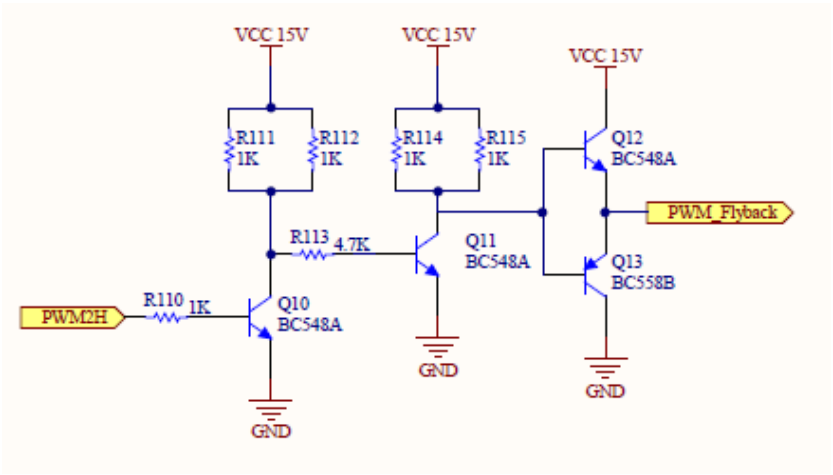


Figura 26 - Circuito para ampliação do sinal PWM da chave da senóide de sincronismo
Fonte: Autoria Própria.

3.3 PROTÓTIPO DO CONVERSOR MONTADO

Para testes do sincronismo com a rede, foi desenvolvido um protótipo de um conversor de estágio único baseado em Kasa e Iida (2002), o qual utiliza um indutor modificado como transformador, este chaveado em alta frequência. Este conversor funciona em modo descontínuo, o qual permite garantir que a saída de corrente seja senoidal, e que esta corrente possua pouca distorção harmônica.

A topologia implementada é muito similar a do *flyback* elementar, com a diferença da adição de uma chave e mais um enrolamento no secundário do transformador para a modulação do semiciclo negativo da senóide. A Figura 27 mostra os circuitos do protótipo do conversor.

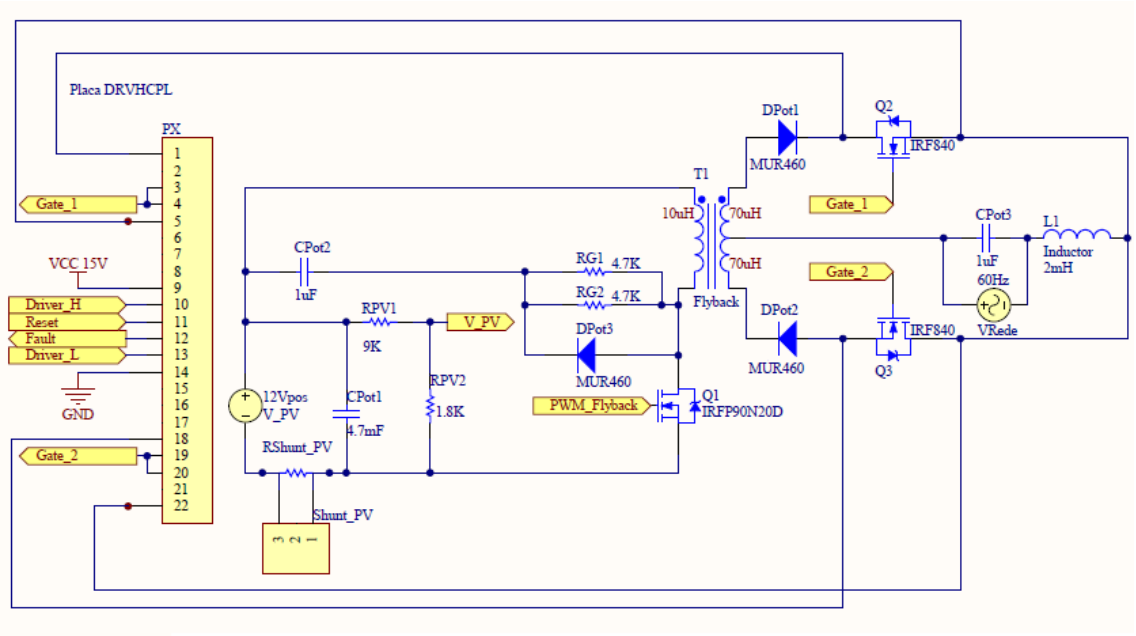


Figura 27 - Circuito do protótipo implementado
Fonte: Autoria Própria.

Neste esquemático, *Driver*_H e *Driver*_L correspondem aos sinais em 60 Hz dos semi-ciclos positivo e negativo dos PWMs de sincronismo antes de serem amplificados, os quais após o devido condicionamento acionam as chaves de sincronismo Q_2 e Q_3 através dos comandos *Gate*_1 e *Gate*_2. Os sinais de *Reset* e *Fault* estão conectados diretamente ao *driver*, no caso de alguma necessidade de interrupção do funcionamento do conversor, e a chave PWM_*Flyback* corresponde ao PWM da senóide de sincronismo chaveada na frequência de 24 kHz, esta sintetizada pelo algoritmo PLL. V_PV é o sinal da tensão de entrada condicionado à leitura do DSP.

A Figura 28 mostra o protótipo do conversor e todos os circuitos utilizados nesse trabalho.

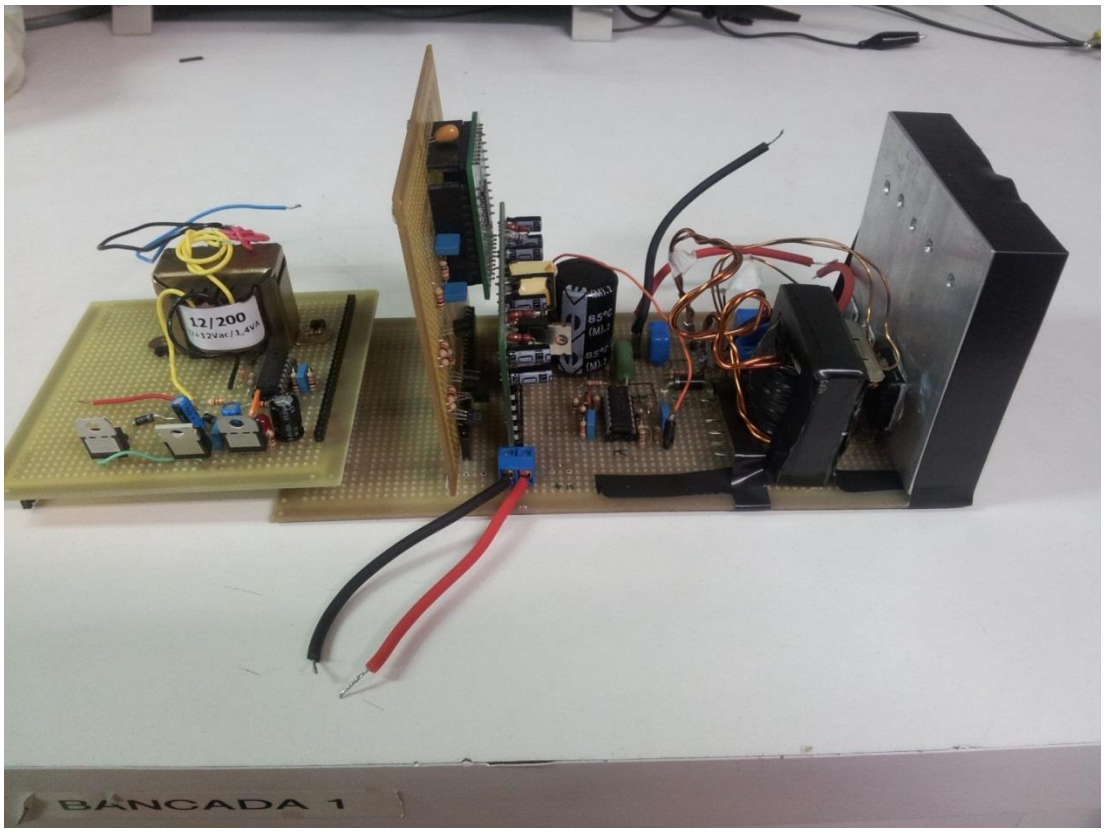


Figura 28 - Protótipo implementado
Fonte: Autoria Própria.

A importância do conversor se dá ao fato que com este o sincronismo pode ser testado efetivamente em relação à rede, de forma a sobrepor o sinal da rede com os sinais de saída do protótipo na visualização do osciloscópio e aferir o sincronismo em fase e frequência.

No Apêndice A pode ser visualizado o esquemático de todos os circuitos implementados neste trabalho.

3.4 O MICROCONTROLADOR DSPIC33FJ16GS

O microcontrolador escolhido para este projeto foi o dsPIC33FJ16GS da Microchip, devido a este possuir um processamento confiável o suficiente em níveis de rotinas e sub-rotinas e uma quantidade adequada de registradores para as funções de controle de potência. Entre as possibilidades deste DSP estão o controle individual adaptivo dos níveis de cada PWM durante a execução do *firmware*, registradores individualizados para cada PWM e seu respectivo nível, entre outros fatores.

O microcontrolador representado na Figura 29 também possui uma boa quantidade de entradas e saídas analógicas e digitais, alta velocidade de processamento além da existência de registradores e interrupções dedicadas à área de processamento de energia. A alimentação

em 3,3 V do DSP ocorre através dos pinos 17, 26 e 40, enquanto os pinos 6, 16, 30 e 39 são utilizados como referência. No pino 26 ocorre a leitura da amostra de tensão senoidal da rede e os PWMs dos pinos 8 e 9 são utilizados para comandar as chaves de sincronismo S_{AC1} e S_{AC2}, respectivamente. O PWM do pino 12 comanda a chave S_{PV}, responsável pela senóide de sincronismo com a rede.

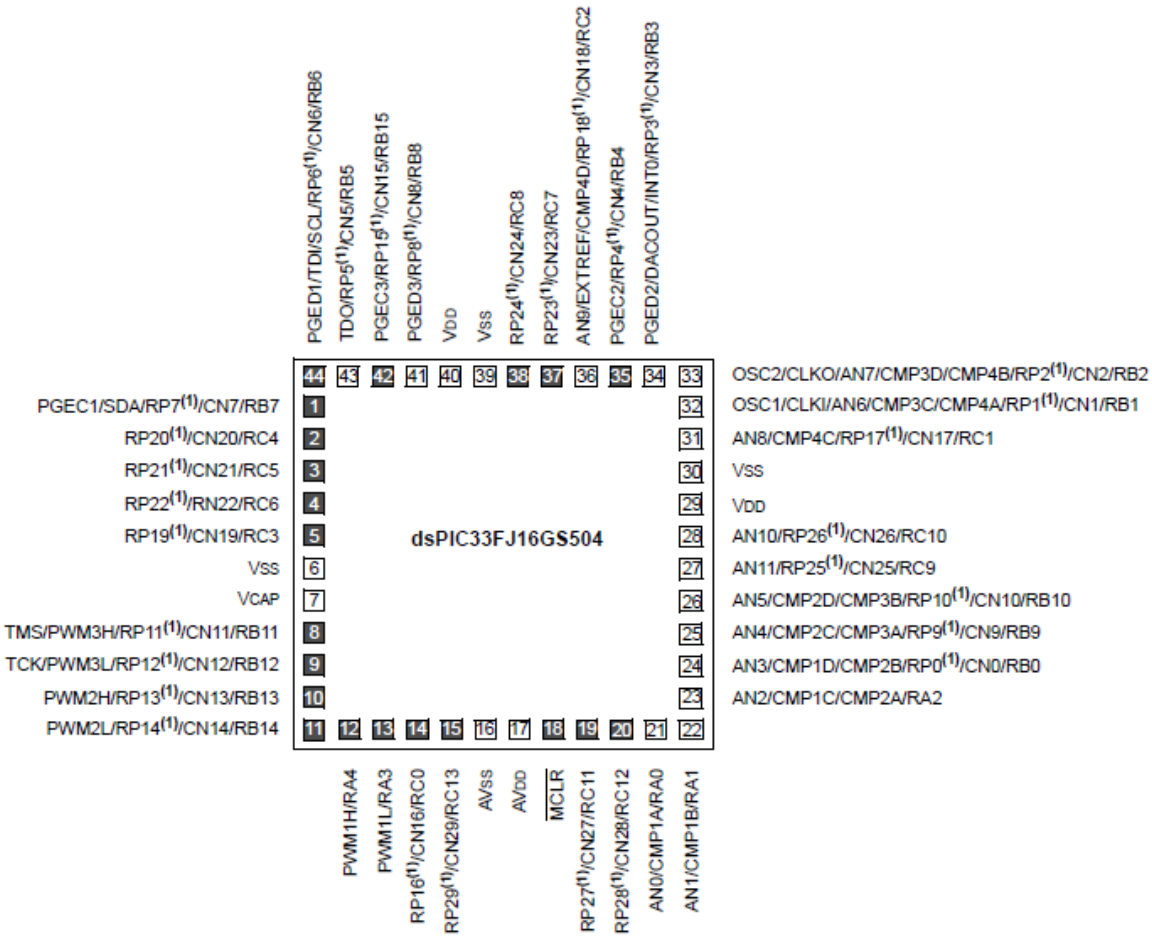


Figura 29 - Pinagem simplificada do dsPIC33FJ16GS
Fonte: Microchip (2011).

A placa padrão montada com o DSP é apresentada na Figura 30.

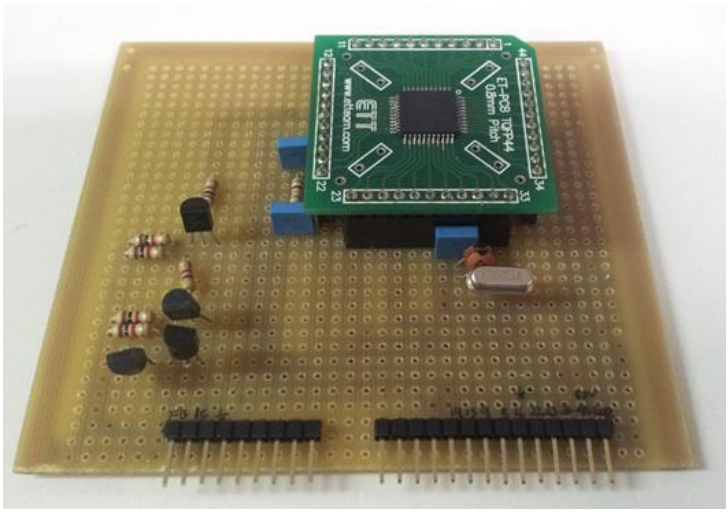


Figura 30 - Placa padrão montada para o DSP
Fonte: Autoria Própria.

3.5 ALGORITMO PLL IMPLEMENTADO

Inicialmente, são feitas as inicializações de *clock*, do oscilador, das portas de entrada e saída, do conversor A/D e do PWM. Devido ao DSP utilizado, o oscilador deve ser configurado em relação ao cristal externo, de forma a fornecer todas as outras bases de *clock* para o *firmware*. A partir do ajuste do oscilador é calculado o valor do período para os PWMs, os quais se utilizam de registradores de tempo além da resolução de *clock* máxima definida no DSP (1,04 ns).

Estes ajustes são peculiaridades da linha de DSP utilizada, o que pode variar de acordo com o fabricante e modelo de DSP utilizado. Neste momento também é escolhida a normalização do PWM, neste caso fixada em formato Q15 (2^{15} = 32768), a qual facilita o trabalho de programação do algoritmo, pois os valores de tensão esperados podem ser mais facilmente calculados e avaliados. Isso se deve a similaridade dos valores após normalizados, de tensão máxima de entrada estabelecida em 3,3 V, e o valor máximo no formato Q15 correspondente a 32767, iniciando este em 0.

A senóide declarada no *firmware* corresponde a um quadrante de uma senóide completa, com valores entre 0 e 32767, seguindo a normalização exposta. Os valores para os outros quadrantes são calculados pelo algoritmo a partir da detecção da passagem por zero.

Neste *firmware*, o conversor A/D junto às portas I/O é responsável pela aquisição da onda de referência da rede, já o PWM é responsável pelas ondas de saída, tanto a onda

senoidal em 60 Hz, quanto às ondas quadradas de sincronismo, as quais devem estar em fase e frequência com a onda de entrada.

Quanto às inicializações do PWM, os canais que foram utilizados para controlar as chaves de sincronismo devem estar configurados no formato complementar, assim enquanto uma chave estiver conduzindo a outra estará bloqueada, salvo momentos de inicialização e pausa do *firmware* onde ambas as chaves são bloqueadas. Além disso, um tempo morto de 20 ns é configurado para evitar a condução simultânea entre as chaves.

Outro fator importante a ser analisado é que num primeiro momento, enquanto o sistema estiver iniciando, o comando das chaves não pode atuar, pois nesse momento o sincronismo não é garantido. É na inicialização do PWM que as razões cíclicas iniciais são definidas a partir de frações do *clock*, e no caso deste *firmware*, o valor utilizado foi de 24 kHz.

Após as inicializações de variáveis e constantes, as interrupções começam a ser executadas, sendo estas acionadas a cada intervalo pré-definido escolhido arbitrariamente em 100 µs, chamado de *Timer* T2. As interrupções do conversor A/D, seguem a seguinte lógica, com sua implementação em escopo completo disponível no Apêndice B:

- O primeiro passo após a inicialização de todas as variáveis é a leitura da tensão atual CC a ser convertida, em formato Q15 e subtração do *offset* de 1,65 V o qual corresponde ao valor aproximado de 16250 no canal analógico AN5;

  tensaoRede = (ADCBUF5 << 5) - 16250; // Subtrai 1,65 V do Offset do AN5

- Leitura da tensão de saída do conversor, no canal analógico AN4 e normalização de valores no formato Q15;

  pvTensaoSaidaInversor = (ADCBUF4 << 5); // Leitura AN4

- O algoritmo passa a efetuar o cálculo de passagens por zero após as leituras de tensão e estado da rede. Assim, caso a *flag* de passagem por zero não esteja habilitada e a contagem de ciclos da senóide de entrada seja maior que 300, a execução do programa continua normalmente e o quadrante atual da senóide de entrada é calculado;

  if((passagemZeroRede == 0) && (contadorPeriodoRede > 300));

A verificação do quadrante é feita a partir da comparação entre a tensão atual da rede e a tensão medida no ciclo anterior. Caso o sinal da tensão tenha mudado pode-se identificar em qual quadrante a senóide se encontra. O valor de 300 é escolhido de

forma a assegurar que o chaveamento só entre em funcionamento após um número apropriado de aquisições.

```
if ((tensaoRede - tensaoRedeAnterior) > 30); //30 é um valor arbitrário no algoritmo
```

- Caso a passagem por zero tenha ocorrido, a *flag* de passagem por zero é habilitada, a *flag* do quadrante é habilitada (como por exemplo, a *flag* do primeiro quadrante). No caso de identificação do primeiro quadrante, o período da rede é salvo em uma variável auxiliar e o contador do período é zerado, o contador de passagem por zero é incrementado e o número de amostras adquiridas é calculado;

```
passagemZeroRede = 1;
flagDeteccaoPrimeiroQuadrante = 1;
flagPrimeiroQuadrante = 1;
contadorPassagemZero++;
if (contadorPassagemZero == 1)
numeroAmostras = 0;
```

- Após esta etapa, a tensão da rede é salva na variável que indica a tensão medida anteriormente, isto é feito para se ter uma base de comparação com os valores a serem obtidos. Assim que detectadas as passagens por zero da senóide, deve-se habilitar a *flag* de inicialização do sistema, a qual serve de controle para a continuação da execução do algoritmo.

```
redeTensaoAnterior = redeTensao;
if(passagemZeroRede == 1)
passagemZeroRede = 0;
if(flagInicializacaoSistema == 1)
contadorZCD ++;
```

- Deve-se sempre verificar o estado da aquisição do sinal da rede. Isto é feito a partir da comparação entre a frequência mínima e máxima definidas pela concessionária de energia e a frequência atual da rede. Se a rede estiver fora da faixa de frequência definida, a *flag* que indica o estado da rede é habilitada para “erro na rede” e as operações na saída do conversor são paradas, caso esteja dentro dos limites permitidos no algoritmo, este continua a execução de acordo com o trecho de código a seguir;

```
if ((periodoRede > periodoMinimoRede ) && (periodoRede < periodoMaximoRede))
if(flagErroFrequenciaRede == 1)
if((periodoRede > (periodoMinimoRede + 20)) && (periodoRede < (periodoMaximoRede - 20)))
estadoFrequenciaRede = FREQUENCIA_REDE_OK;
flagErroFrequenciaRede = 0;
```

- Caso nenhum erro tenha sido identificado, o ângulo da senóide é calculado, a variação do ângulo é calculada e o ângulo atual é salvo na variável que representa o ângulo anterior;

```
anguloFracionario = anguloFracionario + saidaFracionaria; // Cálculo ângulo
if(anguloFracionario > 32767) //32767 é o valor máximo da senóide declarada)
anguloFracionario = anguloFracionario - 32767; // Caso ultrapasse o limite
flagFracionaria = 1;
```

Após isso, utilizando o ângulo calculado, os PWMs de sincronismo PWM3H e PWM3L (que correspondem diretamente aos semiciclos positivo e negativo) são habilitados ou desabilitados conforme o quadrante atual identificado. Isto é feito de acordo com a detecção do ângulo e comparação com o valor atribuído na tabela da senóide, destacada por *Senoide512* no trecho do *firmware* a seguir. A identificação do valor de 90º é necessária, pois somente um quadrante da senóide é declarado no *firmware,* e a partir deste valor e com a utilização de *flags* de estado descobre-se qual é o quadrante real da aquisição;

```
if(flagDeteccao90graus == 0)
if (anguloSenoide < NOVENTA_GRAUS)
referenciaDinamicaCorrente = Senoide512[((anguloSenoide) >> 5)];
anguloSenoide = anguloSenoide + deltaAngulo2 + flagFracionaria;
else
anguloSenoide = NOVENTA_GRAUS;
flagDeteccao90graus = 1;
```

- O PWM do primário PWM1H (que corresponde a senóide de sincronismo) é habilitado conforme a razão cíclica necessária calculada no programa e os quadrantes de sincronismo são determinados a partir do estado em que se

encontra a senóide adquirida, estado este verificado com as *flags* de ângulo da senóide além da *flag* de detecção de passagem por zero;

```
switch(estadoPWM)
case QuadranteSuperior:
if (anguloGlobal > CENTODEZ_GRAUS) // Caso ultrapasse limite de chaveamento do quadrante, deve-se chavear o próximo quadrante
estadoPWM = QuadranteInferior;
case QuadranteInferior:
if (PORTBbits.RB5 == 1) // Checagem do pino Fault
contadorZC = 0; //Zera o detector de passagem por zero, pois está no último quadrante
break;
contadorZC++;
if((anguloGlobal > anguloAcionamentoPWM) && (anguloGlobal < CENTODEZ_GRAUS) && (contadorZC >= 3)) //Caso ponto de acionamento e passagem do Zero Cross Detect for superior ao limite
contadorZC = 0;
estadoPWM = QuadranteInferior; //Permite chaveamento do quadrante inferior
```

- A saída dos PWMs de sincronismo PWM3H para o quadrante superior e PWM3L para o quadrante inferior é habilitada de acordo com o estado da passagem da senóide, de forma a obter o sincronismo em fase e frequência*;*

```
if(estadoPWM = QuadranteSuperior)
IOCON3Bits.OVRDAT = 2;  //PWM 3 High Ativo
if(estadoPWM = QuadranteInferior)
IOCON3bits.OVRDAT = 1; // PWM 3 Low Ativo
else
IOCON3bits.OVRDAT = 0; // PWM 3 Desativado
```

- Por último a *flag* da interrupção é zerada e a *flag* que indica a leitura do conversor A/D é zerada;

Deve-se levar em consideração que para o cálculo das médias com confiabilidade, garantir que a rede esteja funcionando normalmente e que um sinal sincronizado possa ser gerado uma certa margem deve ser dada ao iniciar o programa, esta geralmente é em torno de 300 ciclos da rede. O fluxograma simplificado do algoritmo é apresentado na Figura 31.

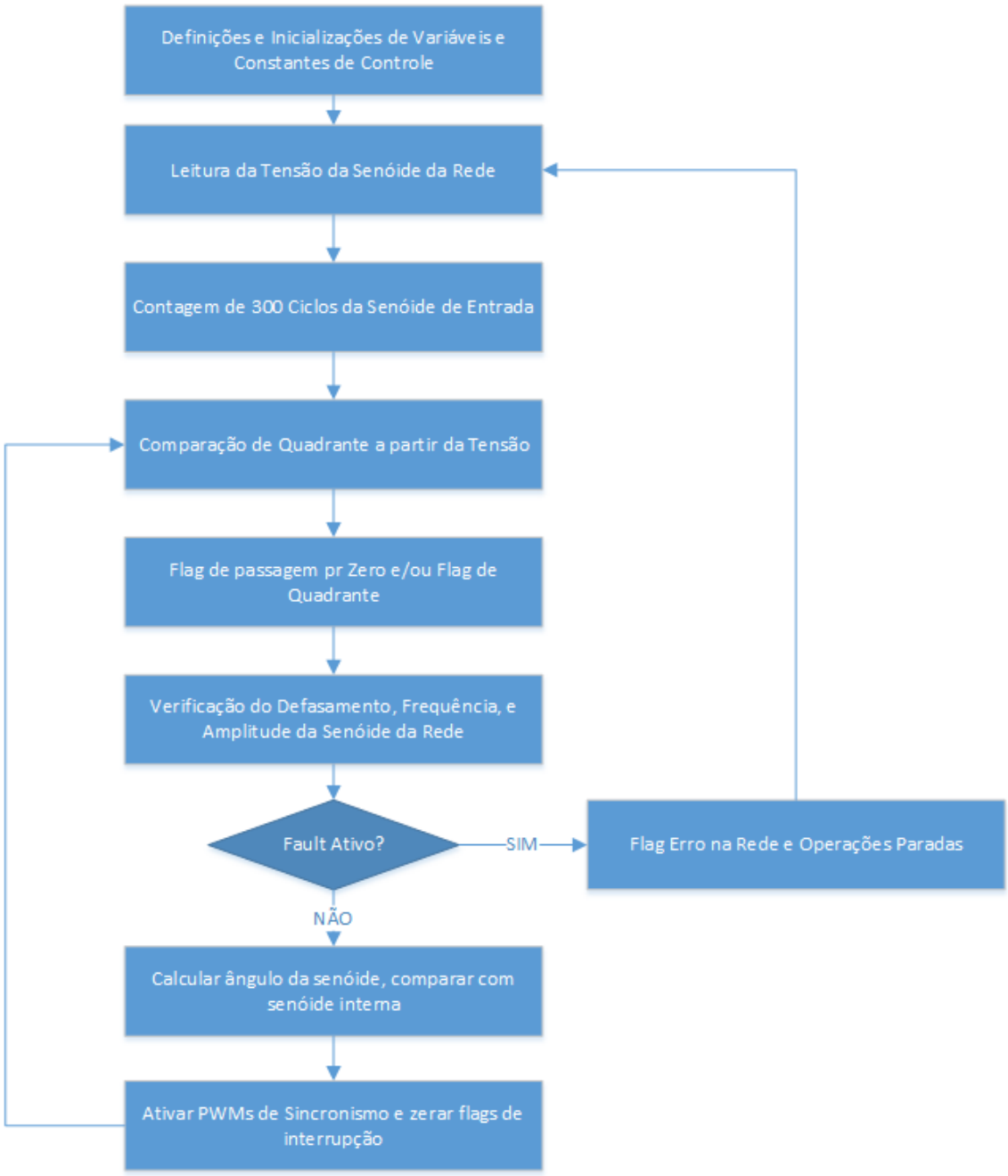


Figura 31 - Fluxograma básico do algoritmo implementado
Fonte: Autoria Própria.

Deve-se sempre analisar o estado do sinal adquirido da rede, de forma a manter o sincronismo ou detectar algum erro. Desta forma, a interrupção que verifica o sinal de *fault* do sistema é chamada a cada intervalo de tempo T2, definido na inicialização do programa e correspondente a 100 µs, e segue os determinados passos:

- Primeiramente a tensão retificada é comparada com o a tensão máxima permitida, caso esta exceda a tensão máxima, ela é limitada a tensão máxima;
- São calculadas as médias de tensão e corrente CC a serem convertidas;
- Após isso o sinal de *fault* é lido, caso este esteja habilitado os sinais de saída são desabilitados e só voltam a funcionar quando o sinal de *fault* for desabilitado;

O fluxograma do algoritmo de controle *fault* é apresentado na Figura 32.

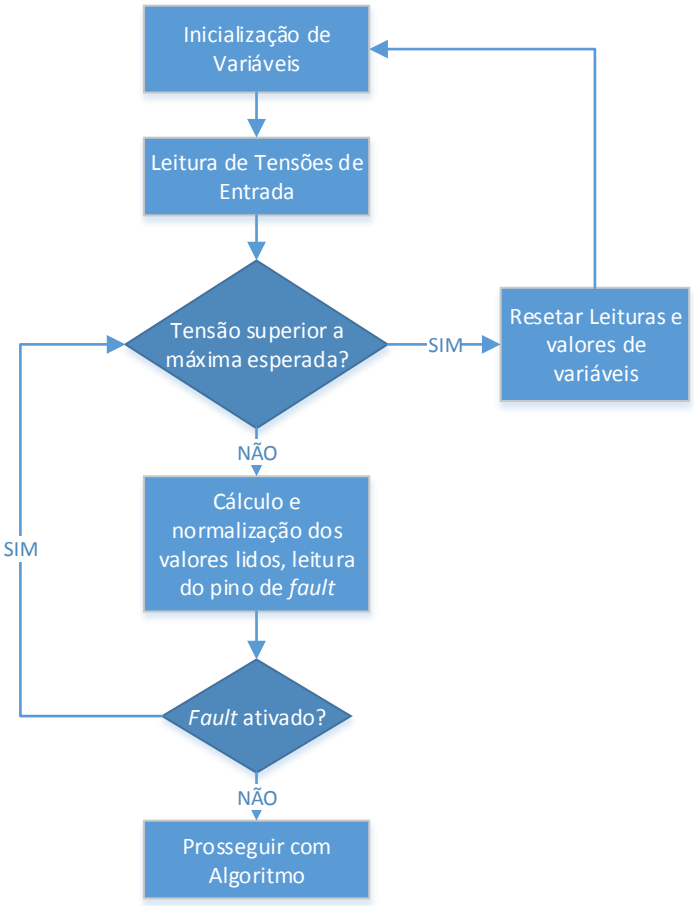


Figura 32 - Fluxograma básico do algoritmo implementado
Fonte: Autoria Própria.

A execução do *firmware* PLL, além do controle das chaves de sincronismo deve gerar uma senóide também em fase e frequência com a rede. Para gerar esta onda o *firmware* utiliza de um quadrante de senóide declarado internamente e calcula os demais quadrantes de acordo com os *flags* de interrupção junto à *flag* de detecção de passagem por zero. A senóide declarada neste algoritmo possui 512 pontos, variando de 0 até 32767, seguindo a normalização Q15 já apresentada, sendo esta chaveada pelo PWM1H em fase e frequência com os PWMs de sincronismo PWM3H e PWM3L, os quais correspondem aos quadrantes superiores e inferiores respectivamente.

As mesmas premissas de checagem do estado do sinal da rede, número de aquisições, leitura de sinal de *fault*, entre outros são adotadas neste caso. O fluxograma do controle da senóide de sincronismo a partir da situação da detecção de uma rede sem erros é apresentado na Figura 33.

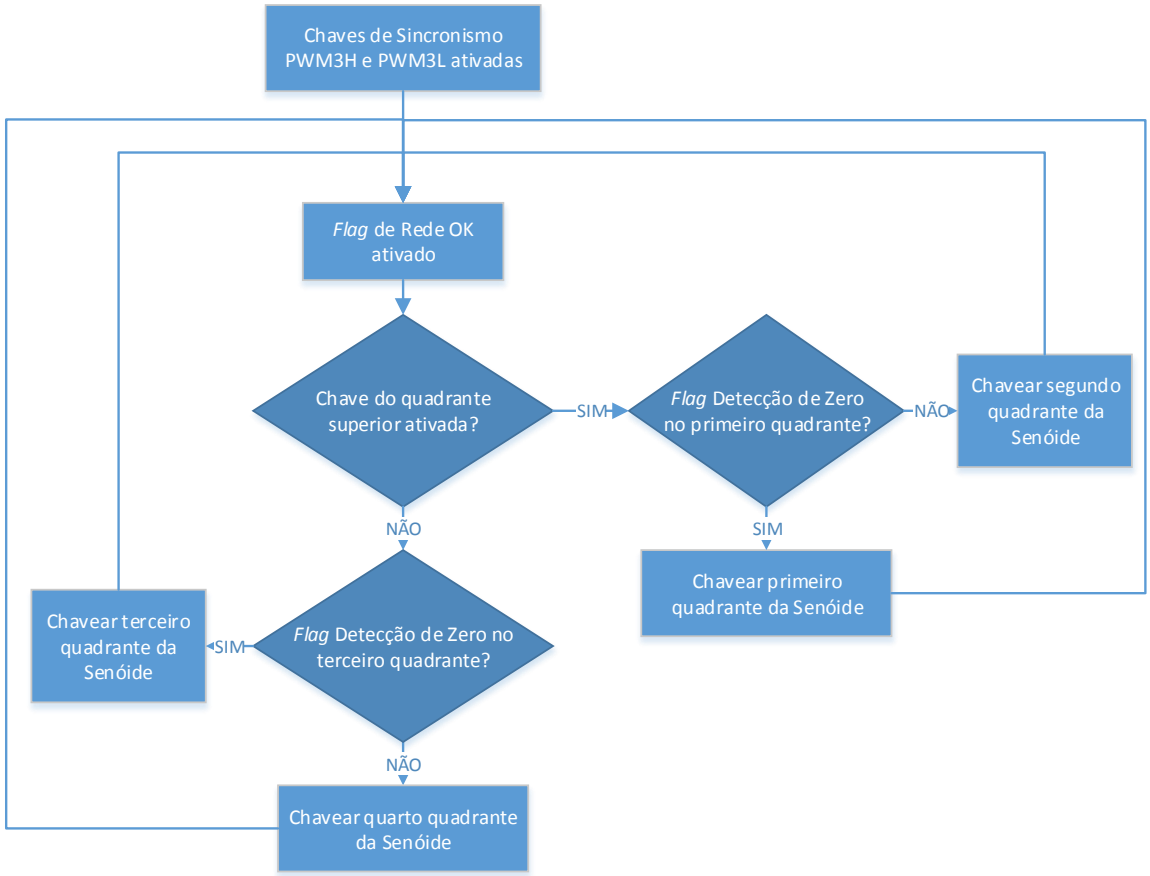


Figura 33 - Fluxograma básico do algoritmo implementado
Fonte: Autoria Própria.

3.6 RESULTADO DO PROTÓTIPO E ALGORITMO PLL

Os testes do protótipo foram efetuados no laboratório LPEE/B207, inicialmente com um gerador de frequências com uma onda senoidal e sua frequência variando entre 47 Hz e 67 Hz, e após os ajustes efetuados no *lock range* do algoritmo, este foi testado junto a rede elétrica em 60 Hz. Estes ajustes foram feitos com o intuito de otimizar o algoritmo na faixa de conexão especificada nas premissas iniciais, ou seja, de 57 Hz a 63 Hz, porém o mesmo ainda permanece apto a buscar as demais frequências especificadas acima, já que seu *capture range* não foi alterado.

O primeiro teste com o gerador de funções foi executado em 50 Hz, o que permitiu aferir o limite inferior do algoritmo implementado. Na Figura 34 podem ser visualizadas as saídas dos PWMs de sincronismo, e a senóide de entrada.

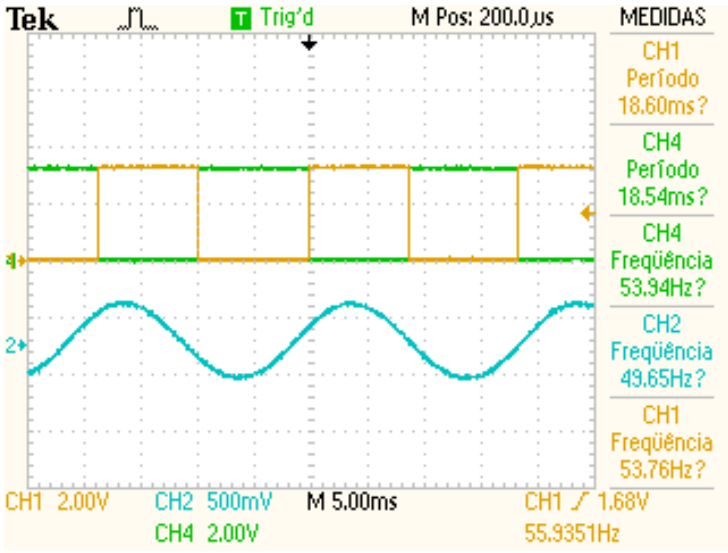


Figura 34 - Teste com gerador de funções em 50 Hz
Fonte: Autoria Própria.

Na continuação dos testes com o gerador de função, a frequência foi alterada para 55 Hz, onde se pode notar já um melhor sincronismo entre a onda de entrada e os PWMs de sincronismo. A Figura 35 mostra o teste em tal frequência.

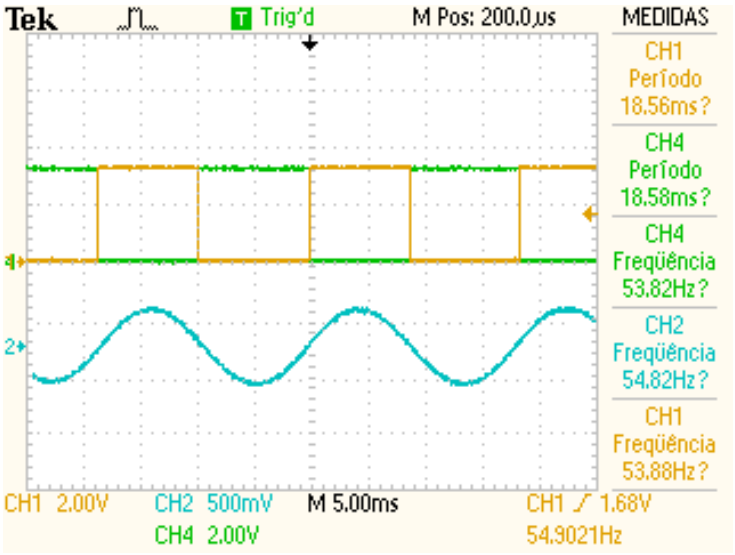


Figura 35 - Teste com gerador de funções em 55 Hz
Fonte: Autoria Própria.

Finalizando os testes práticos de sincronismo e aferição do PLL utilizando-se o gerador de funções, a Figura 36 apresenta a senóide de referência na frequência de 65 Hz, próxima ao limite superior do algoritmo implementado e as respectivas ondas de saída dos PWMs.

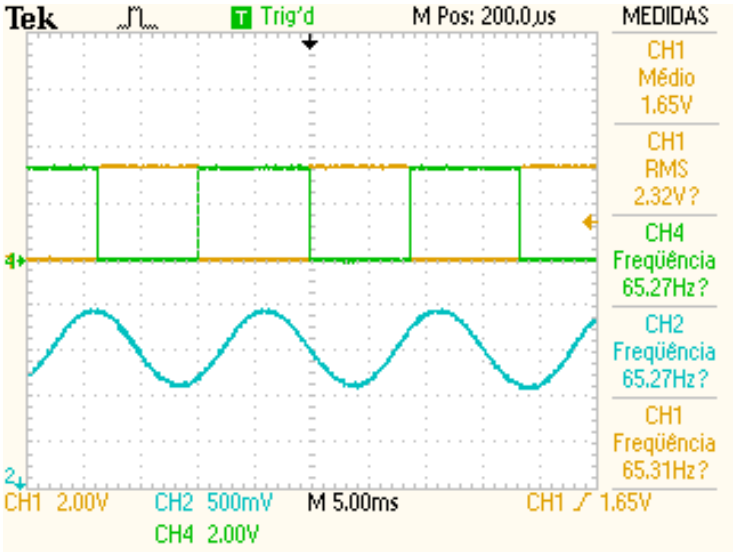


Figura 36 - Teste com gerador de funções em 65 Hz
Fonte: Autoria Própria.

Já com a frequência em 60 Hz, frequência em que o algoritmo esta otimizado para o sincronismo e com o sinal senoidal proveniente da rede elétrica ocorrem os formatos de onda para a senóide de referência e as saídas dos PWMs de sincronização apresentados na Figura 37.

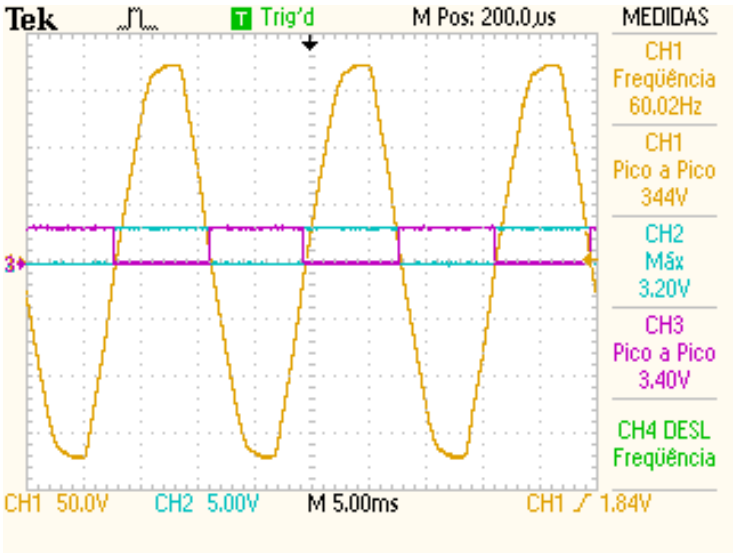


Figura 37 - PLL operando em 60 Hz com a rede elétrica
Fonte: Autoria Própria.

Em uma menor divisão de tempo no osciloscópio, pode-se analisar melhor o ponto onde a onda senoidal cruza o eixo horizontal e ambas as chaves de sincronismo, conforme a Figura 38.

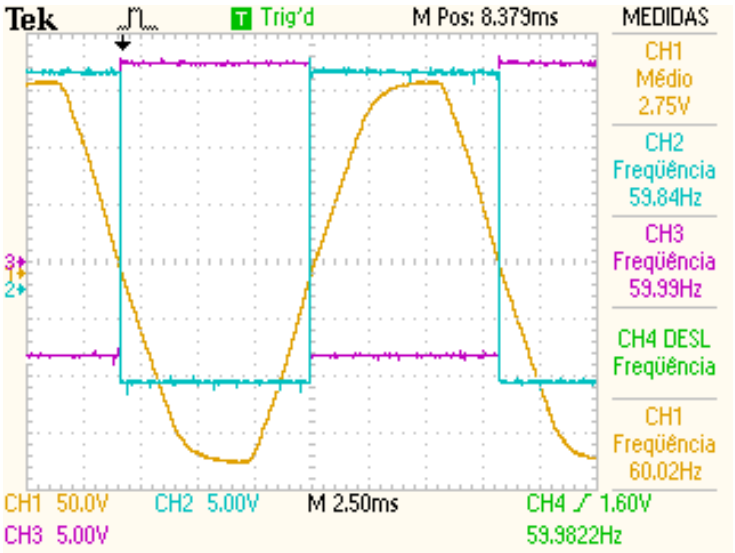


Figura 38 - PLL operando em 60 Hz com a rede elétrica em maior resolução
Fonte: Autoria Própria.

A Figura 39 mostra as ondas de sincronismo do PLL, com o PWM gerador da senóide de sincronismo em verde na frequência de 24 kHz, e as chaves de sincronismo em 60 Hz, onde a onda amarela senoidal corresponde ao sinal da rede de referência.

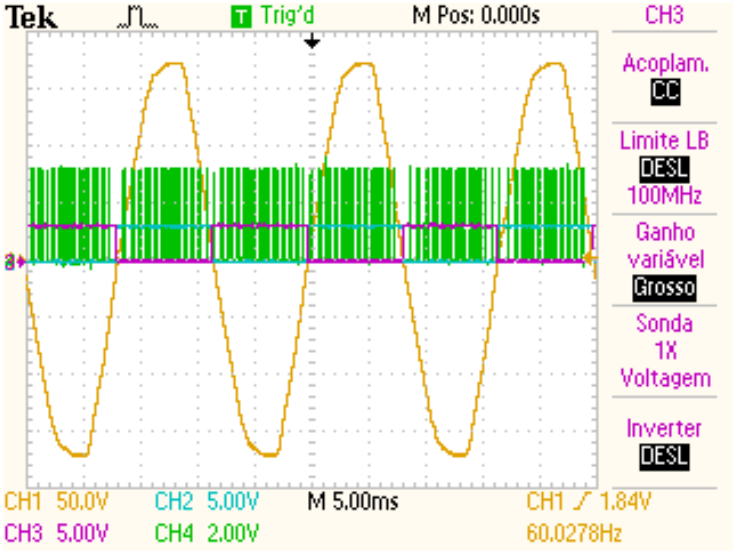


Figura 39 - PLL operando em 60 Hz com a rede elétrica e PWMs de sincronismo
Fonte: Autoria Própria.

Por fim o PLL operando em 60 Hz na rede elétrica com o conversor, onde as ondas de cor roxo e azul são as chaves de sincronismo, a onda verde corresponde ao sinal da rede, e a onda laranja a senóide na saída do conversor, esta sintetizada pelo algoritmo. Nota-se que neste caso o conversor não está conectado a rede elétrica, pois a amplitude da tensão na saída deste é insuficiente para tal conexão.

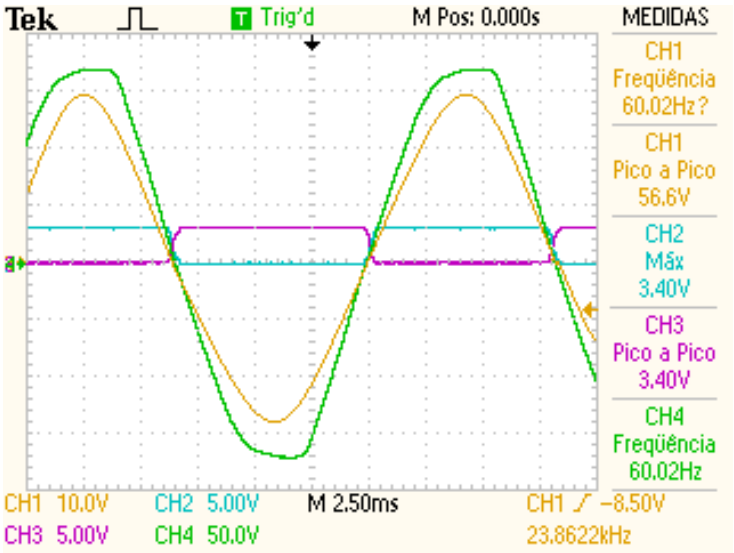


Figura 40 - PLL operando em 60 Hz com a rede elétrica utilizando o conversor implementado
Fonte: Autoria Própria.

4 CONCLUSÕES GERAIS

Foi apresentando neste trabalho de conclusão de curso a implementação digital de um algoritmo de sincronismo aplicado ao controle de um conversor estático. A revisão bibliográfica a respeito dos controles analógicos e digitais de sincronismo foi apresentada, com exemplos de equacionamento básico em anexo a este trabalho. Foram especificados os circuitos e o protótipo operacional montado neste trabalho, além do algoritmo implementado no *firmware* no DSP.

Os testes de sincronismo em fase e frequência apresentaram resultados satisfatórios em frequências próximas de 60 Hz, com os melhores resultados na questão da sincronia com a rede elétrica, fato este devido à própria otimização do algoritmo nesta frequência. O sincronismo com as frequências próximas a 60 Hz pode ser plenamente averiguado e visualizado através do uso de osciloscópios, e no caso das ondas previstas com o uso do conversor, estas foram alcançadas com pleno sucesso, tanto em fase quanto em frequência com a rede elétrica.

O desenvolvimento deste trabalho de conclusão de curso proporcionou grande aprendizado na área de sincronismo e controle de conversores estáticos, podendo servir de auxílio a outros estudantes na UTFPR na implementação de estruturas de sincronismo utilizando o PLL em seus projetos. Por fim, entre as sugestões de trabalhos futuros está a implementação deste algoritmo em um conversor plenamente funcional conectado à rede elétrica, o desenvolvimento de uma malha de corrente para um algoritmo MPPT junto a este PLL e a otimização deste algoritmo apresentado para a detecção e sincronismo em frequências diferentes, além da detecção dinâmica do *offset*.

REFERÊNCIAS

ALBUQUERQUE, M; NUNES, R.A. **Introdução a processadores de sinais digitais - DSP**. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008. Disponível em < http://www.cbpf.br/~rastuto/>. Acesso em: Novembro de 2012.

ALLEN, P. E. **Lecture 170 – Applications of PLLs and Frequency Dividers**, 2003. Disponível em < http://users.ece.gatech.edu/pallen/>. Acesso em: Novembro de 2012.

ANALOG DEVICES. **Phase-Locked Loops for High-Frequency Receivers and Transmitters**, 1999. Disponível em : <http://www.analog.com/static/imported-files/tech_articles/206357438phase_locked.pdf> . Acesso em: Abril de 2013.

AVAGO TECHNOLOGIES. **HCPL-316J Datasheet**, 2011. Disponível em : < http://www.avagotech.com/docs/AV02-0717> . Acesso em: Fevereiro de 2013.

BANERJEE, D. **PLL Performance, Simulation and Design Handbook**, 4 Edition, 2006. Disponível em < http://www.ti.com/tool/pll_book>. Acesso em Dezembro de 2012.

BDTi, Berkeley Design Technology Inc. **Comp.DSP FAQ: Part 3,** 2002**.** Disponível em: < http://www.bdti.com/Resources/Comp.DSP.FAQ/Part3 > Acesso em: Abril de 2013.

BEST, R. E, **Phase-Locked Loops: Design, Simulations and Applications**, 6. ed. Edição. McGraw-Hill, 2007.

BUSO, S; MATTAVELLI, P. **Digital Control in Power Electronics**. Lectures on Power Electronics #2, 2006, v.1.

COPEL Distribuição. **Requisitos Técnicos Para A Conexão De Geração Em Paralelo Com O Sistema Elétrico Da Copel**. Curitiba, 2002. Disponível em: < www.copel.com/hpcopel/normas/>. Acesso em Novembro de 2012.

FISCHETTE, D. **Practical Tips for Phase - Locked Loop Design,** 2009. Disponível em < http://ece.wpi.edu/analog/resources/PLLTutorialISSCC2004.pdf>. Acesso em: Novembro de 2012

FREESCALE. **Phase-Locked Loop Design Fundamentals**, 2006. Disponível em < http://web.itu.edu.tr/~pazarci/pll/MOT_an535rev0a.pdf>. Acesso em: Janeiro de 2013

KASA, N; IIDA, T. ***Flyback* type inverter for small photovoltaic power system**. IEEE IECON, v.2. p.1089-1094, 2002.

MICROCHIP. **16-bit PIC24 MCUs and dsPIC DSCs**, 2011. Disponível em: < http://www.microchip.com/pagehandler/en-us/family/16bit/architecture/dspic33f.html > Acesso em 20 de Novembro de 2012.

PELLENZ, M. E. **Processamento Digital de Sinais (PDS).** Notas de Aula, PPGIA-Pontifícia Universidade Católica do Paraná, 2005. Disponível em < http://www.ppgia.pucpr.br/~marcelo/pds/apostila%20pds.pdf >. Acesso em Novembro de 2012

PICCIONI, C. A; OLIVEIRA, R. S. **Introdução ao DSP**. DAS-UFSC, Florianópolis, 2002.

SCHAFER, R, W; OPPENHEIM, A, V. **Discrete-Time Signal Processing.** 3rd Edition. Prentice Hall Signal Processing, 2008.

STOLZE, T. K; FENGLER, W, **Benchmarking: Classic DSPs vs. Microcontrollers,** Department of Automation and Computer Science, Harz University, 2008. Disponível em < http://www.iiis.org/cds2010/cd2010imc/imcic_2010/paperspdf/za086io.pdf > Acesso em: Abril de 2013.

TI. **TMS320F2x user's guide,** 1992. Disponível em < http://www.ti.com/lsds/ti/dsp/overview.page>. Acesso em Novembro de 2012

TI, **CD54HC297, CD74HC297, CD74HCT297 Datasheet**. Disponível em: <http://www.ti.com/lit/ds/symlink/cd74hc297.pdf/>. Acesso em: Março de 2013.

TI, **Introduction to phase-locked loop system modeling**, 2000. Disponível em: <http://www.ti.com/lit/an/slyt169/slyt169.pdf>. Acesso em: Abril de 2013

TORRES, R. **Processadores Digitais de Sinais e suas Aplicações**. COPPE – UFRJ, Rio de Janeiro, 2006. Disponível em < http://www.cbpf.br/cat/pdsi/downloads/>. Acesso em: Novembro de 2012.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ. **Normas para elaboração de trabalhos acadêmicos**, Comissão de Normalização de Trabalhos Acadêmicos. Curitiba, UTFPR, 2008.

WANG, M; SUN, Y. "**A pratical, precise method for frequency tracking on phasor estimation**", IEEE Trans. on Power Delivery, v. 19, n. 4, pp. 1547-1552, 2004.

WICKERT, M. **Phase-Locked Loops with Applications** - ECE 5675/4675 Lecture Notes, 2011. Disponível em < http://www.eas.uccs.edu/wickert/>. Acesso em Novembro de 2012

ANEXO A – EXEMPLO DE MODELAGEM BÁSICA DE UM PLL ANALÓGICO

O PLL Analógico pode ser modelado a partir das Transformadas de Laplace, as quais permitem a representação de uma resposta no tempo de um sistema no domínio complexo (FREESCALE, 2006). A Figura 41 mostra um modelo do PLL analógico no domínio da frequência.

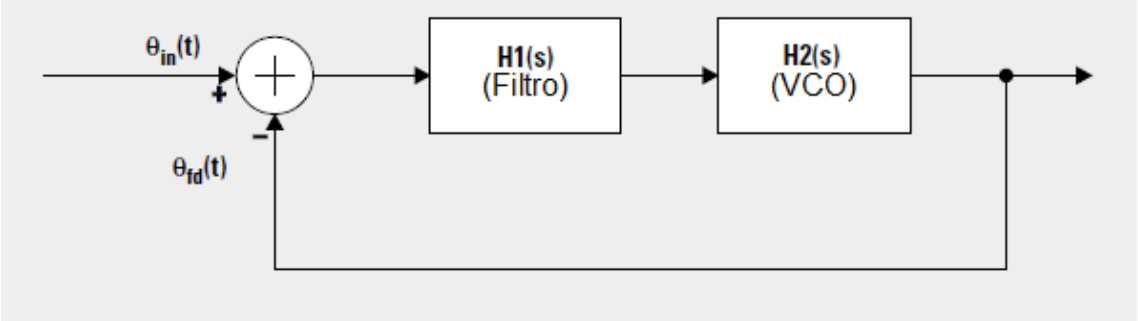


Figura 41 - Modelo de uma malha PLL analógica utilizando transformadas de Laplace
Fonte: Adaptado de TI (2000).

O equacionamento proposto por TI (2000) a seguir é baseado na premissa que o erro de fase é pequeno, com $sen(\theta) \approx (\theta)$, e dessa forma PLL pode ser dePWMito como um modelo linear. Na figura 41, $\theta_{in}(t)$ é a fase do sinal de entrada e $\theta_{fd}(t)$ é a fase do sinal de retorno. Assim as funções de transferências para cada componente podem ser dadas pela transformada de Laplace.

Função de transferência do filtro de primeira ordem:

$$H_1(s) = \frac{G_{lp}}{G_{lp}+S} \quad (1)$$

Função de transferência do VCO:

$$H_2(s) = \frac{G_{VCO}}{S} \quad (2)$$

A partir destas equações e análise da malha pode ser obtida a função de transferência do PLL linear:

$$H_{PLL}(s) = \frac{G_{lp}.G_{VCO}}{S^2+G_{lp}.S+G_{lp}.G_{VCO}} \quad (3)$$

Basendo-se na função de transferência do PLL dada em (3), pode-se notar que a mesma é um sistema de segunda ordem. Na teoria de sistemas de controle, a função de transferência de um sistema de segunda ordem pode ser dada por:

$$H_2(s) = \frac{{\omega_n}^2}{S^2 + 2\xi\omega_n S + {\omega_n}^2} \tag{4}$$

Onde ω_n é definido como a frequência natural e ξ como a taxa de amortecimento. A partir disso, baseando-se na função de transferência do protótipo de um sistema de segunda ordem, a equação característica do sistema pode ser definida por:

$$\Delta(s) = S^2 + 2\xi\omega_n S + {\omega_n}^2 \tag{5}$$

Resolvendo as raízes da equação característica apresentada em (5), dois pólos do sistema, S_0 e S_1 são encontrados:

$$S_0 = -\zeta\omega_n + j\omega_n\sqrt{1-\xi^2} = -\alpha + j\omega \tag{6}$$

$$S_1 = -\zeta\omega_n - j\omega_n\sqrt{1-\xi^2} = -\alpha - j\omega \tag{7}$$

Onde α é definido como o fator de amortecimento e ω definido como frequência amortecida. A partir das equações (6) e (7) e assim que dados ω_n e ξ no sistema, os pólos do protótipo do sistema de segunda ordem podem ser determinados. Estes dois parâmetros são geralmente utilizados na especificação dos parâmetros de desempenho do sistema. De fato, a maioria das resposta de desempenho do sistema podem ser determinadas baseando-se nestes dois parâmetros, tais como:

Fator de Amortecimento α:

$$\alpha = \xi\omega_n \tag{8}$$

Frequência amortecida ω:

$$\omega = \omega_n \sqrt{1 - \xi^2} \tag{9}$$

Tempo de estabelecimento:

$$t_{est} = \frac{4}{\zeta \omega_n} \tag{10}$$

Tempo máximo de *overshoot*:

$$t_{max} = \frac{\pi}{\omega_n \sqrt{1-\xi^2}} \tag{11}$$

Overshoot máximo:

$$M = 1 + e^{-\pi\xi / \sqrt{1-\xi^2}} \tag{12}$$

Overshoot máximo em porcentagem:

$$M = 100.e^{-\pi\xi / \sqrt{1-\xi^2}} \tag{13}$$

Dessa forma é definido o PLL linear como um sistema de segunda ordem no dominio da frequência com este seguindo os critérios de desempenho de acordo com os valores especificados por ω_n e ξ (TI, 2000). Já a estabilidade desse sistema pode ser determinada a partir da técnica do Lugar das Raízes, na qual são dadas as posição dos pólos e zeros do sistema no plano S de forma a visualizar graficamente a estabilidade do sistema. O desenho ilustra como as raízes características da equação variam com o ganho da malha. Para a estabilidade, todos os pólos devem estar na metade esquerda do plano S, o que pode ser assegurado desde que o valor da taxa de amortecimento ξ >0. Já a relação dos pólos e zeros do sistema determina o grau de estabilidade (FREESCALE, 2006).

Para um sistema do tipo 1, ou seja, possuindo apenas um pólo na função de transferência com este localizado na origem, e de segunda ordem com uma entrada *step*, tem-se o seguinte gráfico de estabilidade e *overshoot* mostrado na Figura 42.

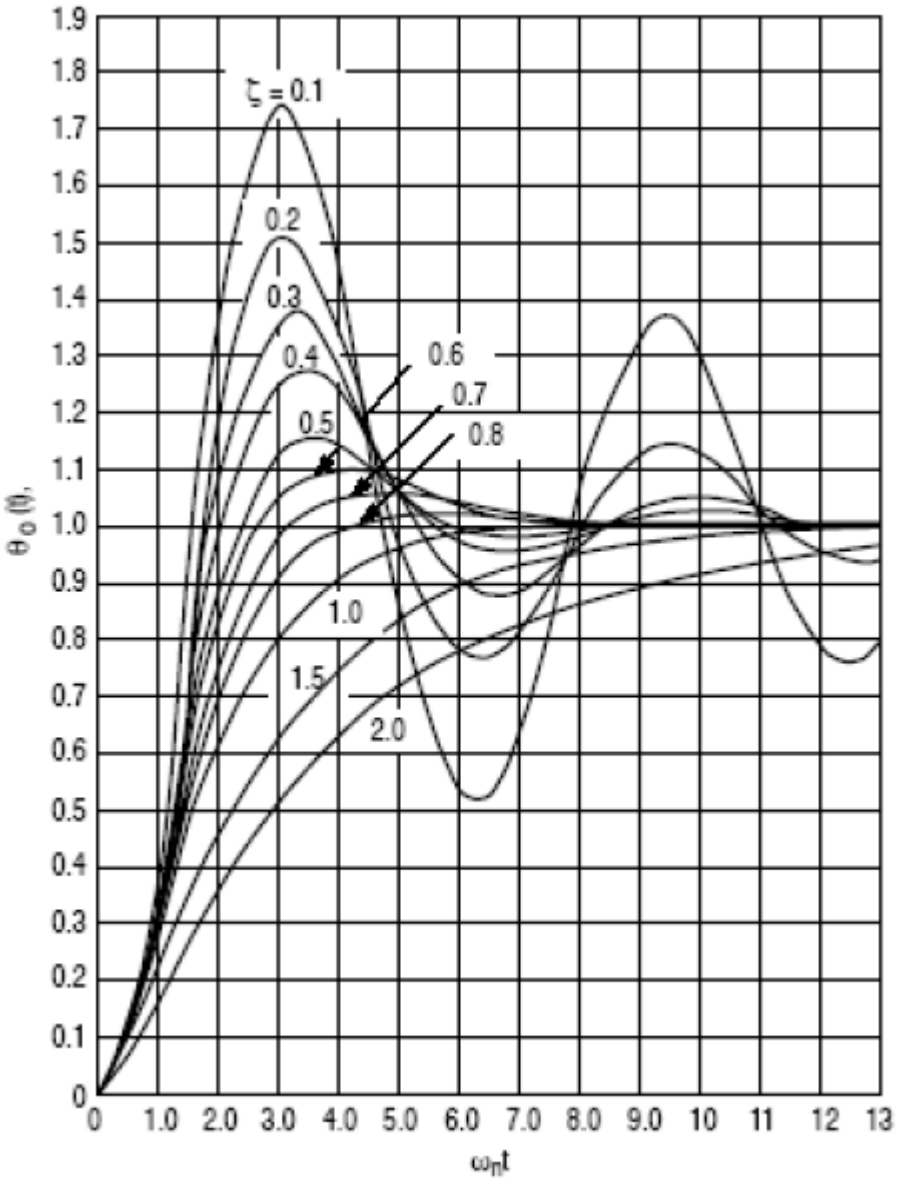


Figura 42 - Resposta de um sistema de segunda ordem, tipo 1
Fonte: Freescale (2006).

A Figura 42 mostra a resposta típica de um sistema de segunda ordem, tipo 1, a uma entrada step, em que o *overshoot* e a estabilidade respondem em função do valor da taxa de amortecimento ξ. Assim a resposta do sistema pode ser otimizada em um determinado tempo com o ajuste dos ganho do detector de fase e do VCO, bem como o acerto dos coeficientes do filtro passa-baixas.

ANEXO B – EXEMPLO DE MODELAGEM BÁSICA DE UM PLL DIGITAL

O PLL digital necessita da discretização da malha PLL básica, a qual no espaço Z pode ser dado pela estrutura presente na Figura 43.

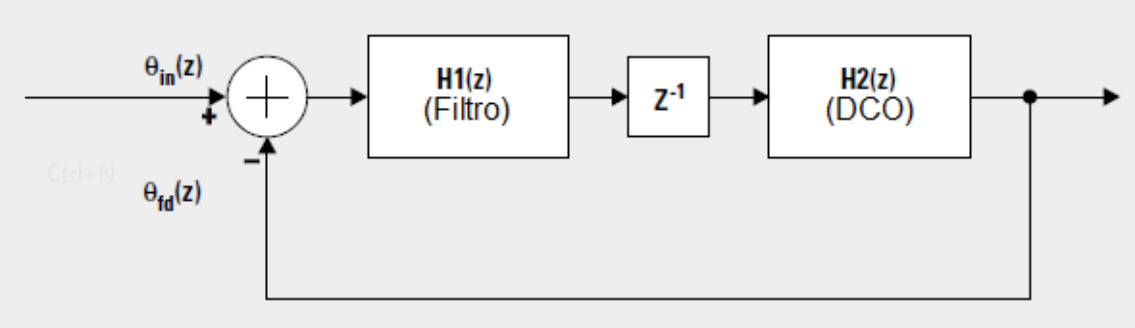


Figura 43 - Modelo discretizado da malha PLL básica
Fonte: Adaptado de TI (2000).

O diagrama de blocos da figura 43 mostra a malha discretizada, onde DCO corresponde ao oscilador em forma digital e as funções de transferência de cada componente do PLL são dadas por:

Função de transferência do filtro:

$$H1(Z) = \frac{aZ-1}{Z-1} \tag{14}$$

Frequência amortecida ω:

$$H2(z) = \frac{cZ}{Z-1} \tag{15}$$

E z^{-1} corresponde ao atraso entre a conversão A/D e a saída do PLL.

A partir do diagrama de blocos e as funções de transferência dos componentes, um modelo em tempo linear invariante pode ser desenvolvido para representar o PLL (TI, 2000). Assim a função de transferência da malha fechada do PLL digital pode ser calculada como:

$$H(z) = \frac{acZ-c}{Z^2+(ac-2).Z+(1-C)} \tag{16}$$

Mapeando os pólos de um sistema de segunda ordem do domínio da frequência para o domínio do tempo, pode-se reePWMever tal equação:

$$H(z) = \frac{N(Z)}{(Z-Z_0).(Z-Z_1)} \tag{17}$$

Onde Z_0 e Z_1 são os dois pólos do sistema no domínio do tempo (Z). De forma análoga a análise do domínio da frequência (S), a equação característica no sistema em tempo discreto é definida por:

$$\Delta(z) = (Z - Z_1).(Z - Z_0) = Z^2 - (Z_1 + Z_0).Z + Z_1.Z_0 \quad (18)$$

C_0 e C_1 são definidos como coeficientes da equação característica:

$$C_1 = -(Z_1 + Z_0) \quad (19)$$

$$C_0 = Z_1.Z_0 \quad (20)$$

Assim a equação característica pode ser ePWMita do modo simplificado a seguir:

$$\Delta(z) = Z^2 + C_1 Z + C_0 \quad (21)$$

Pela definição da transformação em tempo discreto por Schafer e Oppenheim (2008), os dois pólos desse sistema no domínio do tempo (Z) podem ser mapeados a partir dos pólos no domínio da frequência (S), de forma que:

$$z_0 = e^{S_0.T_s} = e^{(-\zeta\omega_n.T_s + j\omega_n.T_s\sqrt{1-\xi^2})} \quad (22)$$

$$z_1 = e^{S_1.T_s} = e^{(-\zeta\omega_n.T_s - j\omega_n.T_s\sqrt{1-\xi^2})} \quad (23)$$

Onde T_s é o período de amostragem no sistema discreto.

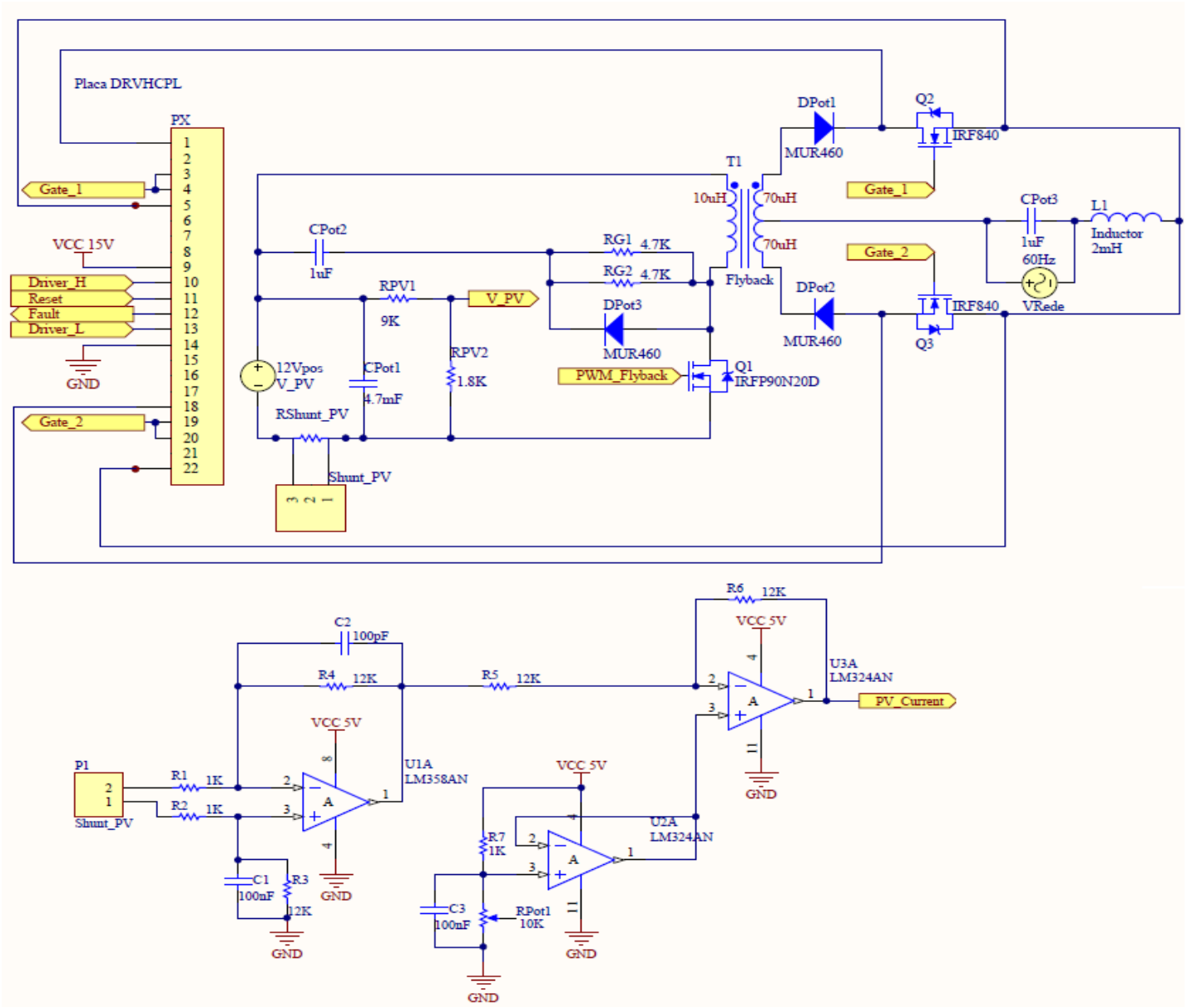
Com os pólos mapeados no domínio do tempo (Z) e junto a equação (19) e (20) os coeficientes C_0 e C_1 da equação característica (21) podem ser obtidos em um formato que é dePWMito pelos parâmetros ω_n e ξ (já citados anteriormente):

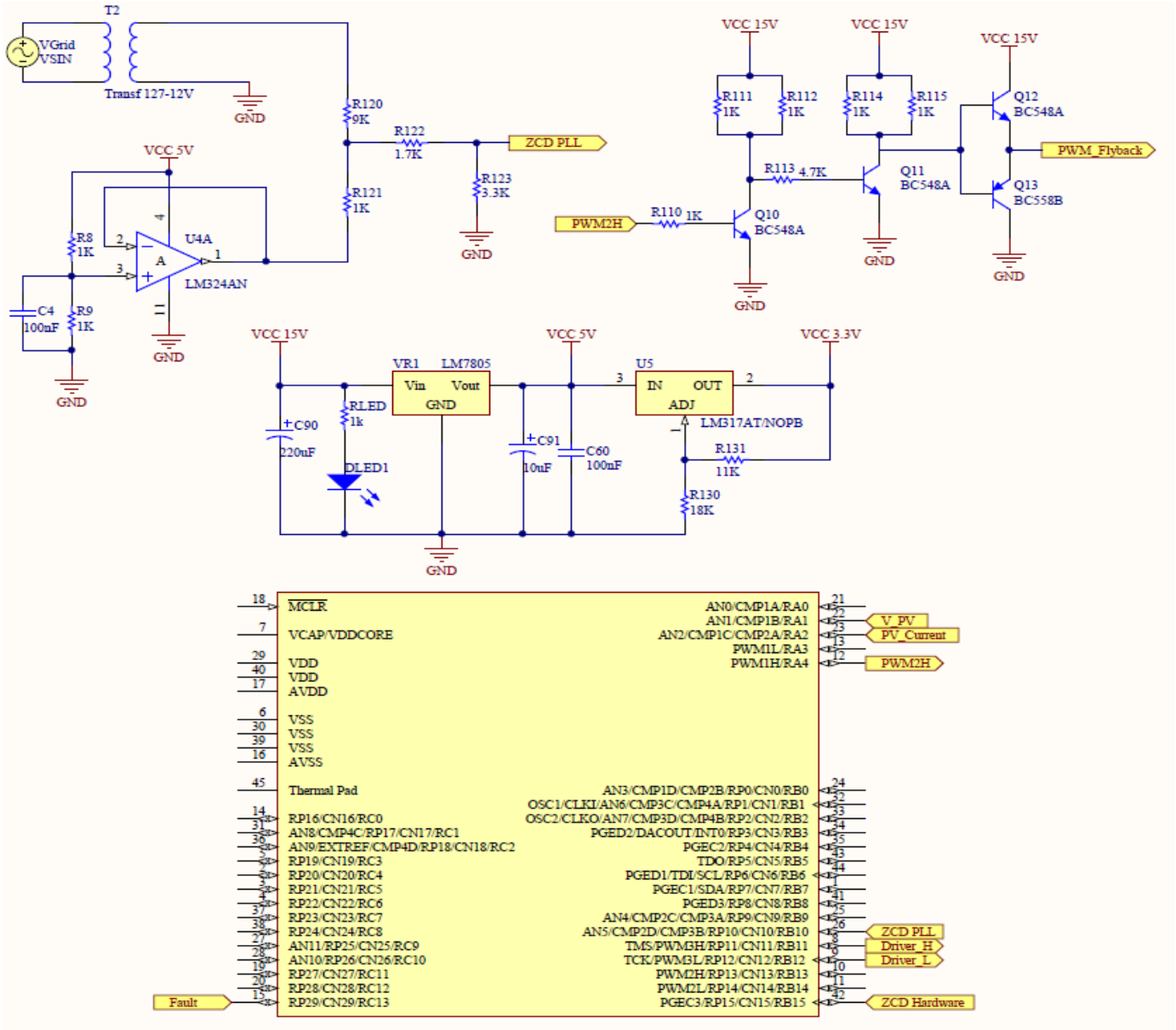
$$C_0 = e^{-2\xi.\omega_n.T_s} \quad (24)$$

$$C_1 = -2e^{-\xi.\omega_n.T_s}\cos(\omega_n.T_s\sqrt{1-\xi^2}) \quad (25)$$

Assim a equação característica pode ser definida com o levantamento dos pólos no domínio do tempo contínuo. Sabendo que a função característica irá afetar as respostas transitórias do sistema, as equações (17) e (21) podem determinar a função de transferência do PLL digital. O numerador da equaçao (17) pode ser um fator de escala constante, ou zeros podem ser introduzidos para melhorar o desempenho do sistema. No caso de um PLL digital adotar o aspecto da equação (16), a sua função de transferência será determinada assim que os pólos forem mapeados (TI, 2000).

APÊNDICE A – CIRCUITOS IMPLEMENTADOS NESTE TRABALHO


Placa DRVHCPL
PX
Gate_1
VCC_15V
Driver_H
Reset
Fault
Driver_L
GND
Gate_2
CPot2
1uF
RPV1
9K
V_PV
12Vpos
V_PV
CPot1
4.7mF
RPV2
1.8K
RShunt_PV
Shunt_PV
RG1 4.7K
RG2 4.7K
DPot3
MUR460
PWM_Flyback
Q1
IRFP90N20D
T1
10uH
70uH
70uH
Flyback
DPot1
MUR460
DPot2
MUR460
Q2
IRF840
Gate_1
Gate_2
Q3
IRF840
CPot3
1uF
60Hz
VRede
L1
Inductor
2mH
C2
100pF
R4 12K
R5 12K
R6 12K
VCC_5V
U1A
LM358AN
U2A
LM324AN
U3A
LM324AN
P1
R1 1K
R2 1K
C1
100nF
R3
12K
R7
1K
C3
100nF
RPot1
10K
PV_Current


T2
VGrid
VSIN
Transf 127-12V
GND
R120
9K
R122
1.7K
ZCD PLL
R123
3.3K
R121
1K
VCC 5V
R8
1K
U4A
LM324AN
C4
100nF
R9
1K
VCC 15V
R111
1K
R112
1K
R114
1K
R115
1K
Q12
BC548A
PWM_Flyback
R113
4.7K
Q11
BC548A
Q13
BC558B
PWM2H
R110
1K
Q10
BC548A
VCC 3.3V
VR1
LM7805
Vin
Vout
U5
IN
OUT
ADJ
LM317AT/NOPB
RLED
1k
C90
220uF
DLED1
C91
10uF
C60
100nF
R131
11K
R130
18K
MCLR
VCAP/VDDCORE
VDD
AVDD
VSS
AVSS
Thermal Pad
RP16/CN16/RC0
AN8/CMP4C/RP17/CN17/RC1
AN9/EXTREF/CMP4D/RP18/CN18/RC2
RP19/CN19/RC3
RP20/CN20/RC4
RP21/CN21/RC5
RP22/CN22/RC6
RP23/CN23/RC7
RP24/CN24/RC8
AN11/RP25/CN25/RC9
AN10/RP26/CN26/RC10
RP27/CN27/RC11
RP28/CN28/RC12
RP29/CN29/RC13
Fault
AN0/CMP1A/RA0
AN1/CMP1B/RA1
AN2/CMP1C/CMP2A/RA2
PWM1L/RA3
PWM1H/RA4
V_PV
PV_Current
PWM2H
AN3/CMP1D/CMP2B/RP0/CN0/RB0
OSC1/CLKI/AN6/CMP3C/CMP4A/RP1/CN1/RB1
OSC2/CLKO/AN7/CMP3D/CMP4B/RP2/CN2/RB2
PGED2/DACOUT/INT0/RP3/CN3/RB3
PGEC2/RP4/CN4/RB4
TDO/RP5/CN5/RB5
PGED1/TDI/SCL/RP6/CN6/RB6
PGEC1/SDA/RP7/CN7/RB7
PGED3/RP8/CN8/RB8
AN4/CMP2C/CMP3A/RP9/CN9/RB9
AN5/CMP2D/CMP3B/RP10/CN10/RB10
TMS/PWM3H/RP11/CN11/RB11
TCK/PWM3L/RP12/CN12/RB12
PWM2H/RP13/CN13/RB13
PWM2L/RP14/CN14/RB14
PGEC3/RP15/CN15/RB15
ZCD PLL
Driver_H
Driver_L
ZCD Hardware

APÊNDICE B – Programa dsPIC – Arquivo C: “PLL_principal”

```
#include <p33FJ16GS.h>

// bits de configuração
_FOSCSEL(FNOSC_FRC)
_FOSC(FCKSM_CSECMD & OSCIOFNC_ON & IOL1WAY_OFF)
_FWDT(FWDTEN_OFF)
_FPOR(FPWRT_PWR128)
_FICD(ICS_PGD2 & JTAGEN_OFF)

int main(void)
{
        initClock();                // inicialização do clock primário e auxiliar
        initADC();                  // inicialização do conversor ADC

        ADCONbits.ADON = 1;         // habilita o módulo ADC

        initPWM();                  // inicializa o módulo PWM
        initIOports();              // inicializa as portas I/O
        initEstadoTimer();          // inicializa o Timer2

        T2CONbits.TON = 1;          // habilita o Timer2
        PTCONbits.PTEN = 1;         // habilita o módulo PWM

        while(1)
        {
                Nop();
                Nop();
                Nop();
        }
}
```

***Firmware* dsPIC – Arquivo C: "PLL_inicializa"**

```
#include <p33FJ16GS.h>
#include "PLL_Auxiliar.h"

void initClock(void)
{
// Configuração do oscilador
// Fosc = Fin*M/(N1*N2), Fcy = Fosc/2
// Fosc = 7.37*(43)/(2*2) = 80MHz for Fosc, Fcy = 40MHz
// Configuração do PLL prescaler, PLL postscaler, divisor PLL

PLLFBD = 41;                        // M = PLLFBD + 2
CLKDIVbits.PLLPOST = 0;             // N1 = 2
CLKDIVbits.PLLPRE = 0;              // N2 = 2

// mudança do oscilador para FRC + PLL
__builtin_write_OSCCONH(0x01);      // novo oscilador FRC com PLL
__builtin_write_OSCCONL(0x01);      // habilita chaveamento do clock

while(OSCCONbits.COSC != 0b001);    // aguarda o oscilador se tornar FRC com
PLL
while(OSCCONbits.LOCK != 1);        // aguarda o travamento do PLL

// Configura o ADC e o PWM clock para 120MHz
// ((FRC * 16) / APSTSCLR ) = (7.37 * 16) / 1 = ~ 120MHz

ACLKCONbits.FRCSEL = 1;             // FRC é responsável pela entradas do PLL
auxiliar(x16)
ACLKCONbits.SELACLK = 1;            // oscilador auxiliar, responsável pelo
clock do PWM e ADC
ACLKCONbits.APSTSCLR = 7;           // divisão do clock auxiliar por 1
ACLKCONbits.ENAPLL = 1;             // habilita PLL auxiliar
while(ACLKCONbits.APLLCK != 1);     // aguarda o travamento do PLL auxiliar

}
void initPWM(void)
{

// configuração do PWM para um inversor Flyback

PTPER = FLYBACK_VALOR_PERIODO;      // valor do período do Flyback
PTCON2bits.PCLKDIV = 0;             // resolução máxima de tempo (1.04nsec)
```

```
PWMCON1bits.ITB = 0;                // PTPER: registrador responsável por gerar o tempo para PWM1, PWM2
PWMCON1bits.IUE = 1;                // habilita atualizações imediatas
PWMCON1bits.DTC = 0;                // dead-time positivo habilitado
PWMCON1bits.DTC = 0;
IOCON1bits.PENH = 0;                // GPIO controla o pino do PWM na inicialização
IOCON1bits.PENL = 0;
IOCON1bits.PMOD = 0;                // PWM1H e PWM1L no modo complementar
 IOCON1bits.OVRDAT = 0;             // o inversor dos PWMs serão desligados em caso de sobrecarga
IOCON1bits.OVRENH = 1;              // chaveamento dos PWMs serão desligado durante a INICIALIZACAO_SISTEMA
IOCON1bits.OVRENL = 1;              // chaveamento dos PWMs serão desligados durante a INICIALIZACAO_SISTEMA
PDC1 = 100;                         // inicialização do duty-cicle do flyback
PDC2 = 100;
TRGCON2bits.TRGDIV = 1;             // ativado a cada 2 períodos do PWM ~17us
TRGCON2bits.TRGSTRT = 0;            // aguarda 0 ciclos do PWM antes de gerar o primeiro ciclo
TRIG1 = 1000;                       // PWM1 usado para ativar o ADCP0

// inicialização do PWM3 para controle de sincronismo

PWMCON3bits.ITB = 0;                // base de tempo para o PWM3
PWMCON3bits.IUE = 1;                // habilita atualizações imediatas
IOCON3bits.PENH = 0;                // GPIO controla o o pino do PWM na inicialização
IOCON3bits.PENL = 0;
IOCON3bits.PMOD = 3;                // modo de saída independente
IOCON3bits.OVRDAT = 0;              // PWM será desativado inicialmente
IOCON3bits.OVRENH = 1;              // habilita Override durante a inicialização
IOCON3bits.OVRENL = 1;              // habilita Override durante a inicialização

PDC3 = 0;                           // inicialização do duty-cicle do PWM3
SDC3 = 0;
}
```

```
void initADC(void)
{
ADCONbits.SLOWCLK = 1;          // requerido pela errata do ADC
ADCONbits.FORM = 0;             // saída no formato inteiro
ADCONbits.EIE = 0;              // desabilita interrupções durante a
inicialização
ADCONbits.ORDER = 0;            // ordem dos canais
ADCONbits.SEQSAMP = 0;          // amostras simultâneas habilitado
ADCONbits.ASYNCSAMP = 1;        // amostra desincronizadas habilitada
ADCONbits.ADCS = 5;             // divisor do clock configurado para
Fadc/6, TAD = 50ns

ADSTAT = 0;                     // limpa o registrador ADSTAT
ADPCFG = 0xFFC0;                // AN0 - AN5 são valores analógicos
ADPCFGbits.PCFG5 = 0;           // AN5 aquisição da tensão da rede
ADCPC0bits.TRGSRC0 = 5;         // AN0 e AN1 disparados pelo PWM2
ADCPC0bits.TRGSRC1 = 4;         // AN2 e AN3 disparados pelo PWM1
ADCPC1bits.TRGSRC2 = 4;         // AN4 e AN5 disparados pelo PWM1
IPC27bits.ADCP0IP = 5;          // ADC prioridade de interrupções
IFS6bits.ADCP0IF = 0;           // limpa a flag de interrupção do ADC
IEC6bits.ADCP0IE = 1;           // habilita a interrupção do ADC
}

void initIOports(void)
{
TRISBbits.TRISB7 = 0;           // porta I/O para o LED
TRISAbits.TRISA4 = 0;           // GPIO configuração de ultrapassagem do
PWM1
TRISAbits.TRISA3 = 0;
LATAbits.LATA4 = 0;
LATAbits.LATA3 = 0;
TRISBbits.TRISB13 = 0;          // GPIO configuração de ultrapassagem do
PWM2
TRISBbits.TRISB14 = 0;
LATBbits.LATB13 = 0;
LATBbits.LATB14 = 0;
TRISBbits.TRISB11 = 0;          // GPIO configuração de ultrapassagem do
PWM3
TRISBbits.TRISB12 = 0;
LATBbits.LATB11 = 0;
LATBbits.LATB12 = 0;
}
```

```
void initEstadoTimer(void)
{
T2CONbits.TCKPS = 0;                    // Prescaler de 1:1
PR2 = 4000;                             // (100us /25ns) = 4000
IPC1bits.T2IP = 4;                      // configuração da interrupção do Timer2
IEC0bits.T2IE = 1;                      // habilita a interrupção do Timer2
}
```

***Firmware* dsPIC – Arquivo C: "PLL_isr"**

```
#include <p33FJ16GS.h>
#include "PLL_Auxiliar.h"
#include "Senoide512.h" // Senóide de 512 pontos, de 0 a 32767

void __attribute__((interrupt, no_auto_psv)) _ADCP0Interrupt()
{
// leitura da tensão e corrente no formato Q15

pvTensaoModulo  = (ADCBUF1 << 5);       // leitura da tensão
pvCorrenteModulo = (ADCBUF2 << 5);      // leitura da corrente do flyback

// é necessário subtrair o Offset (1.65V), 522*32(Q15) ~ 16383 para utilizar apenas o nível AC

pvTensaoSaidaInversor = (ADCBUF4 << 5);          // leitura da tensão de saída
tensaoRede = (ADCBUF5 << 5) - 16250;    // leitura da tensão da rede

// leitura da corrente de saída
if(pvCorrenteSaidaInversor >= 0)
{
        retificadaIacQ15 = pvCorrenteSaidaInversor;
}
else
{
        retificadaIacQ15 = (-pvCorrenteSaidaInversor);
}

if((passagemZeroRede == 0) && (contadorPeriodoRede > 300))
{
        // detecta se houve passagem por zero no 1º quadrante
        if ((tensaoRedeAnterior < 0) && (tensaoRede >= 0))
        {
                if ((tensaoRede - tensaoRedeAnterior) > 30)
                {
                        passagemZeroRede = 1;
                        flagDeteccaoPrimeiroQuadrante = 1;
                        flagPrimeiroQuadrante = 1;
// contador do período da rede, para cálculo da frequência
                        periodoRede = contadorPeriodoRede;
                        contadorPeriodoRede = 0;
```

```
// guarda a soma da tensão e a quantia de passagens por zero para cálculo da média (média calculada no T2ISR)
// a média é calculada a cada 3 passagens por zero
                        contadorPassagemZero++;

                        if (contadorPassagemZero == 1)
                        {
                                numeroAmostras = 0;
                        }

// periodoRede é de apenas meio ciclo, então é adicionado periodoRedeAnterior
                        numeroAmostras = numeroAmostras + periodoRede + periodoRedeAnterior;
                        if(contadorPassagemZero >= 3)
                        {
                                contadorPassagemZero = 0;

                                finalInputVoltageSum = somaTensaoEntrada;
                                finalInputCurrentSum = somaCorrenteEntrada;
                                avgInputDataReadyFlag = 1;

                                somaTensaoEntrada = 0;
                                somaCorrenteEntrada = 0;
                        }
                }
        }

// detecta se houve passagem por zero no 3º quadrante
        else if ((tensaoRedeAnterior >= 0) && (tensaoRede < 0))
        {
                if ((tensaoRede - tensaoRedeAnterior) < -30)
                {
// devemos começar com a passagem por zero no 1º quadrante
                        if (flagDeteccaoPrimeiroQuadrante == 1)
                        {
                                passagemZeroRede = 1;
                                flagPrimeiroQuadrante = 0;

// carrega o valor do período da rede para calcular a frequência
                                periodoRede = contadorPeriodoRede;
                                contadorPeriodoRede = 0;
                        }
                }
```

```
        }
}

// contador para verificar a frequência da rede (numero de interrupções do ADC dividido pelo número de meios ciclos)
// variável é resetada assim que o evento de passagem por zero é verificado
if(flagDeteccaoPrimeiroQuadrante == 1)
{
        contadorPeriodoRede++;
// quando é identificada uma passagem por zero os valores de tensão e corrente são acumulados para calcular a média
        somaCorrenteEntrada = somaCorrenteEntrada + pvCorrenteModulo;
        somaTensaoEntrada = somaTensaoEntrada + pvTensaoModulo;
        }

// salva o valor da tensão da rede para calcular com o próximo valor medido
tensaoRedeAnterior = tensaoRede;

if(passagemZeroRede == 1)
{
        passagemZeroRede = 0;

        if(flagInicializacaoSistema == 1)
        {
                contadorZCD ++;
        }

// após 100 passagens por zero, remove-se o PWM Override
        if((contadorZCD > 100) && (flagInicializacaoSistema == 1) && (flagPrimeiroQuadrante == 1))
        {
                FLYBACKDUTY = 0;
                IOCON1bits.OVRENH = 0;  // Remove-se o Override para ligar a saída do PWM

                IOCON1bits.PENH = 1;       // PWM que controla o pino do PWM1H
                IOCON3bits.PENL = 1;       // PWM que controla o pino do PWM3L
                IOCON3bits.PENH = 1;       // PWM que controla  pino do PWM3H
                estadoPWM = PWM_INATIVO_4_QUADRANTE;
                flagInicializacaoSistema = 0;
                flagInicioMppt = 1;
                Isaida = 0;
                contadorZCD = 0;
```

```
        }
//verifica-se se a frequencia da rede está OK
        if ((periodoRede > periodoMinimoRede ) && (periodoRede < periodoMaximoRede))
        {
                if(flagErroFrequenciaRede == 1)
                {
// Histerese da frequência
                        if((periodoRede > (periodoMinimoRede + 20)) && (periodoRede < (periodoMaximoRede - 20)))
                        {
                                estadoFrequenciaRede = FREQUENCIA_REDE_OK;
                                flagErroFrequenciaRede = 0;
                        }
                }
                else
                {
                        estadoFrequenciaRede = FREQUENCIA_REDE_OK;
                }
                if (periodoRedeAnterior <= periodoRede)
                {
                        periodoRedeAtual = periodoRedeAnterior;
                }
                else
                {
                        deltaPeriodoRede = (periodoRedeAnterior - periodoRede);
                        periodoRedeAtual = (periodoRede + (deltaPeriodoRede >> 1));
                }
                periodoRedeAnterior = periodoRede;
                anguloSenoide = 0;
                anguloGlobal = 0;
                flagDeteccao90graus = 0;
        }
        else
        {
                estadoFrequenciaRede = FREQUENCIA_REDE_NAO_OK;
                flagErroFrequenciaRede = 1;
                anguloSenoide = 0;
                anguloGlobal = 0;
                flagDeteccao90graus = 0;
        }

        deltaAnguloAnterior = deltaAngulo;
```

```
        deltaAngulo = __builtin_divsd((long)32767,periodoRedeAtual);
        deltaAngulo2 =((deltaAngulo + deltaAnguloAnterior) >> 1);
        saidaFormatoInt = __builtin_muluu( deltaAngulo,periodoRedeAtual);
        manter = 32767 - saidaFormatoInt;
        saidaFracionaria = __builtin_divsd(((long)manter << 15),periodoRedeAtual);
        anguloAcionamentoSRC = 12 * deltaAngulo2;
// carrega o valor de pico (feito toda vez que uma passagem por zero é identificada)
        picoVacQ15 = maxVacQ15;
        maxVacQ15 = 0;
        picoRedeVac = maxRedeVacQ15;
        maxRedeVacQ15 = 0;
}

anguloFracionario = anguloFracionario + saidaFracionaria;

if(anguloFracionario > 32767)
{
        anguloFracionario = anguloFracionario - 32767;
        flagFracionaria = 1;
}
else
{
        flagFracionaria = 0;
}

if(flagDeteccao90graus == 0)
{
        if (anguloSenoide < NOVENTA_GRAUS)
        {
                referenciaDinamicaCorrente = Senoide512[((anguloSenoide) >> 5)];
                anguloSenoide = anguloSenoide + deltaAngulo2 + flagFracionaria;
        }
        else
        {
                anguloSenoide = NOVENTA_GRAUS;
                flagDeteccao90graus = 1;
        }
}
else
{
        referenciaDinamicaCorrente = Senoide512[((anguloSenoide) >> 5)];
        anguloSenoide = anguloSenoide - deltaAngulo2 - flagFracionaria;
```

```
        if (anguloSenoide < ZERO_GRAUS)
        {
                anguloSenoide = ZERO_GRAUS;
        }
}

anguloGlobal = anguloGlobal + deltaAngulo2 + flagFracionaria;

if(flagInicializacaoSistema == 0)
{
// é preciso determinar qual angulo da senóide atual para habilitar e desabilitar o PWM3
        switch(estadoPWM)
        {
                case QUADRANTESUPERIOR:
                {
                        if (anguloGlobal > CENTODEZ_GRAUS)
                        {
                                estadoPWM = QUADRANTEINFERIOR;
                        }
                }
                break;

                case QUADRANTEINFERIOR:
                {
                        if (PORTBbits.RB5 == 1) // Checa pino Fault
                        {
                                contadorZC = 0;
                                break;
                        }

                        contadorZC++;

                        if((anguloGlobal > anguloAcionamentoPWM) && (anguloGlobal < CENTODEZ_GRAUS) && (contadorZC >= 3))
                        {
                                contadorZC = 0;
                                estadoPWM = QUADRANTEINFERIOR;
                        }
                }
                break;

                case QUADRANTEINFERIOR:
```

```
                {
                        if (anguloGlobal > CENTODEZ_GRAUS)
                        {
                                estadoPWM = PWM_INATIVO_4_QUADRANTE;
                        }
                }
                break;

                case PWM_INATIVO_4_QUADRANTE:
                {
                        if(PORTBbits.RB5 == 0) // Checa pino Fault
                        {
                                contadorZC = 0;
                                break;
                        }

                        contadorZC++;

                        if((anguloGlobal > anguloAcionamentoSRC) && (anguloGlobal
< CENTODEZ_GRAUS) && (contadorZC >= 3))
                        {
                                contadorZC = 0;
                                estadoPWM = QUADRANTESUPERIOR;
                        }
                }
                break;
        }

// agora é possível modificar a saída dos PWM's

        if(estadoPWM == QUADRANTESUPERIOR)
        {
                IOCON3bits.OVRDAT = 2;                  // PWM3H ativo
        }
        else if (estadoPWM == QUADRANTEINFERIOR)
        {
                IOCON3bits.OVRDAT = 1;                  // PWM3L ativo
        }
        else
        {
                IOCON3bits.OVRDAT = 0;
        }
}
```

```
IoReferencia = (__builtin_mulss((int)referenciaDinamicaCorrente,(int)mpptFator) >>
15);                                        //IoReferencia = 32767 => pico 5A

erroCorrente = 0;

Psaida  =  (__builtin_mulss(erroCorrente,Ra) >> 15);
IRdrop  = (__builtin_mulss(IoReferencia,Rp) >> 15);

if((flagSaturacao == 0) || (erroCorrente < 0))
{
        Isaida  =  Isaida + (long)((__builtin_mulss((int)erroCorrente,(int)Rsa)) >> 15);
}

if(Isaida > 32767)
{
        Isaida = 32767;
}
else if(Isaida < -32767)
{
        Isaida = -32767;
}

saidaTotal = (long)Psaida + Isaida + (long)IRdrop;

if(saidaTotal > 32767)
{
        saidaTotal = 32767;
}
else if(saidaTotal < -32767)
{
        saidaTotal = -32767;
}

VinnominalAcimaVinfactor      =      __builtin_divsd(((long)VINNOMINAL      <<
15),(int)pvTensaoModulo );
saidaTotal =  (__builtin_mulss( saidaTotal , VinnominalAcimaVinfactor) >> 15);

Duty = saidaTotal;

flybackDuty  =  (__builtin_mulss((int)Duty,(int)FLYBACK_VALOR_PERIODO)  >>
15);
```

```
finalFlybackDutyCycle = flybackDuty + deltaDuty ;
if(finalFlybackDutyCycle < 0)
{
        finalFlybackDutyCycle = 0;
}

FLYBACKDUTY = finalFlybackDutyCycle;
if(FLYBACKDUTY < 300)
{
        TRIG1 = (FLYBACKDUTY >> 1);
}
else
{
        TRIG1 = (FLYBACKDUTY - 200);
}

if((estadoTensaoRede == TENSAO_REDE_NAO_OK)||(estadoFrequenciaRede ==
FREQUENCIA_REDE_NAO_OK))
{
        estadoRede = REDE_NAO_OK;
}
else
{*/
        estadoRede = REDE_OK;
}

ADSTATbits.P0RDY = 0;           /* limpa o ADC ready bit */
IFS6bits.ADCP0IF = 0;           /* limpa a flag de interrupção do ADC */
}
```

***Firmware* dsPIC – Arquivo C : "PLL_Estado"**

```
#include <p33FJ16GS.h>
#include "PLL_Auxiliar.h"

void MPPTRotina(void);

// interrupção do Timer2, verificação do fault
void __attribute__((__interrupt__, no_auto_psv)) _T2Interrupt()
{
// tensão retificada da rede
if(tensaoRede >= 0)
{
        tensaoRetificadaRede = tensaoRede;
}
else
{
        tensaoRetificadaRede = (-tensaoRede);
}

// tensão de pico da rede
if(tensaoRetificadaRede > maxRedeVacQ15)
{
        maxRedeVacQ15 = tensaoRetificadaRede;
}

// tensão do inversor retificada
if(pvTensaoSaidaInversor >= 0)
{
        tempVacQ15 = pvTensaoSaidaInversor;
}
else
{
        tempVacQ15 = (-pvTensaoSaidaInversor);
}

// tensão de pico do inversor
if(tempVacQ15 > maxVacQ15)
{
        maxVacQ15 = tempVacQ15;
}

switch(estadoSistema)
```

```
{
        case INICIALIZACAO_SISTEMA:
        {
                if (estadoRede == REDE_OK)
                {
                        PLLStartupCount++;
                        if ((PLLStartupCount > 1000))
                        {
                                if ((periodoRede > 440 ) && (periodoRede <= 520))
                                {
                                        periodoMinimoRede = 440;
                                        periodoMaximoRede = 520;
                                }
                                else
                                {
                                        periodoMinimoRede = 521;
                                        periodoMaximoRede = 650;
                                }

                                Isaida = 0;
                                saidaTotal = 0;
                                Duty = 0;
                                deltaDuty = 0;
                                contadorZCD = 0;
                                startupCheckCount = 0;
                                mpptFator = FATOR_MINIMO_MPPT;
                                potenciaAnteriorEntrada = 0;
                                potenciaEntrada = 0;
                                mediaAnteriorTensaoEntrada = 0;
                                mediaTensaoEntrada = 0;
                                mediaCorrenteEntrada = 0;
                                estadoSistema = FUNCIONAMENTO_NORMAL;
                                estadoErro = SEM_FAULT;
                                PLLStartupCount = 0;
                                flagInicializacaoSistema = 1;
                        }
                }
                else
                PLLStartupCount++;
                        if ((PLLStartupCount > 1000))
                        {
                                if ((periodoRede > 440 ) && (periodoRede <= 520))
                                {
```

```
                                        periodoMinimoRede = 440;
                                        periodoMaximoRede = 520;
                                }
                                else
                                {
                                        periodoMinimoRede = 521;
                                        periodoMaximoRede = 650;
                                }
                                Isaida = 0;
                                saidaTotal = 0;
                                Duty = 0;
                                deltaDuty = 0;
                                contadorZCD = 0;
                                startupCheckCount = 0;
                                mpptFator = FATOR_MINIMO_MPPT;
                                potenciaAnteriorEntrada = 0;
                                potenciaEntrada = 0;
                                mediaAnteriorTensaoEntrada = 0;
                                mediaTensaoEntrada = 0;
                                mediaCorrenteEntrada = 0;
                                estadoSistema = FUNCIONAMENTO_NORMAL;
                                estadoErro = SEM_FAULT;
                                PLLStartupCount = 0;
                                flagInicializacaoSistema = 1;
                        }
                }
                break;
                case FUNCIONAMENTO_NORMAL:
                {
        // calculo da média da tensão e corrente de entrada
                if(avgInputDataReadyFlag == 1)
                {
                        avgInputDataReadyFlag = 0;
                        mediaTensaoEntrada =
__builtin_divsd((long)finalInputVoltageSum ,(int)(numeroAmostras));
                        mediaCorrenteEntrada =
__builtin_divsd((long)finalInputCurrentSum ,(int)(numeroAmostras));
                        numeroAmostras = 0;
                        inputAverageCompletedFlag = 1;
                }

                pvUnderVoltageCounter = 0;
        }
```

```
break;
TMR2 = 0;
IFS0bits.T2IF = 0;
}
```

***Firmware* dsPIC – Arquivo H : "PLL_Auxiliar"**

```
// variáveis do conversor ADC
int pvTensaoSaidaInversor = 0;
unsigned int pvTensaoModulo = 0;
int pvCorrenteModulo = 0;
int tensaoRede = 0;
int pvCorrenteSaidaInversor = 0;

// variáveis para o calculo do angulo e da frequência
unsigned int anguloAcionamentoSRC = 0;
long saidaFormatoInt = 0;
long saidaFracionaria = 0;
int flagFracionaria = 0;
long manter = 0;
unsigned int anguloFracionario = 0;
int deltaAngulo = 57;
int deltaAnguloAnterior = 0;
int periodoMinimoRede = 440;                    // no começo a frequência é desconhecido, visto que o programa opera de 47 Hz até 63 Hz
int periodoMaximoRede = 650;                    // o mínimo e máximo devem ser o mais exato possível
unsigned int deltaAngulo2 = 0, anguloSenoide = 0;
unsigned char passagemZeroRede = 0;
unsigned int periodoRede = 520;
unsigned int periodoRedeAtual = 520;
unsigned int periodoRedeAnterior = 500;
unsigned int deltaPeriodoRede = 0;
unsigned int contadorPeriodoRede = 520;
unsigned int anguloGlobal = 0;
int flagDeteccaoPrimeiroQuadrante = 0;
int tensaoRedeAnterior = 0;
char contadorPassagemZero = 0;
unsigned int contadorZC = 0;

// variáveis de controle
int flybackDuty = 0;
int finalFlybackDutyCycle = 0;
int Psaida = 0;
long Isaida = 0;
long saidaTotal = 0;
long Duty = 0;
int IRdrop = 0;
```

```
int IoReferencia = 0;
int erroCorrente = 0;
unsigned int referenciaDinamicaCorrente = 0;
int retificadaIacQ15 = 0;

// variáveis para a detecção do pico
int tempVacQ15 = 0,picoVacQ15 = 0,maxVacQ15 = 0;
int picoRedeVac = 0,maxRedeVacQ15 = 0;
int tensaoRetificadaRede = 0;

// variáveis de sistema
unsigned char estadoSistema = INICIALIZACAO_SISTEMA;
unsigned char estadoErro = SEM_FAULT;
unsigned char estadoFrequenciaRede = FREQUENCIA_REDE_OK;
unsigned char estadoTensaoRede = TENSAO_REDE_OK;
unsigned char estadoRede = REDE_NAO_OK;
unsigned char estadoPWM = PWM_INATIVO_4_QUADRANTE;

// variáveis de contagem
unsigned char contadorZCD = 0;

// variáveis flag
unsigned char flagInicioMppt = 0;
unsigned char flagSaturacao = 0;
unsigned char flagDeteccao90graus;
unsigned char flagInicializacaoSistema = 2;
unsigned char flagErroFrequenciaRede = 0;
unsigned char flagPrimeiroQuadrante = 0;

int deltaDuty = 0;      // esta variável é a diferença de duty cicle necessária a ser adicionada/subtraída
```