IMPLEMENTAÇÃO DO ALGORITMO DA MODULAÇÃO VETORIAL USANDO COORDENADAS MÓVEIS NÃO-ORTOGONAIS EM *FIELD-PROGRAMMABLE GATE ARRAY* PARA INVERSORES MULTINÍVEIS FONTE DE TENSÃO

EDVALDO FRANCISCO FREITAS LIMA

Campo Grande

2009

UNIVERSIDADE FEDERAL DO MATO GROSSO DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

IMPLEMENTAÇÃO DO ALGORITMO DA MODULAÇÃO VETORIAL USANDO COORDENADAS MÓVEIS NÃO-ORTOGONAIS EM *FIELD-PROGRAMMABLE GATE ARRAY* PARA INVERSORES MULTINÍVEIS FONTE DE TENSÃO

Dissertação submetida à Universidade Federal de Mato Grosso do Sul como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Elétrica.

EDVALDO FRANCISCO FREITAS LIMA

Campo Grande, Abril de 2009.

IMPLEMENTAÇÃO DO ALGORITMO DA MODULAÇÃO VETORIAL USANDO COORDENADAS MÓVEIS NÃO-ORTOGONAIS EM *FIELD-PROGRAMMABLE GATE ARRAY* PARA INVERSORES MULTINÍVEIS FONTE DE TENSÃO

Edvaldo Francisco Freitas Lima

"Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em *Eletrônica de Potência*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Mato Grosso do Sul**."**

_____

Nicolau Pereira Filho, Doutor
Orientador

_____

Luciana Cambraia Leite, Doutora
Coordenadora do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____

Benjamim Rodrigues de Menezes, Doutor.

_____

João Onofre Pereira Pinto, Doutor.

Dedicatória

Aos meus pais Antonio e Iris, por fazerem de tudo por mim e por sempre me incentivarem ao estudo.

Agradecimentos

A Deus;

Aos professores Nicolau e Luciana, pelo apoio nos momentos de dificuldades passados no inicio do mestrado;

Aos amigos Meliton, Hebert, Nelci, Susana, Edgard, Rafael, José e todos os outros colegas de laboratório;

Aos amigos Maicon, Daniela, Márcia, Thalles, Alexsandro, pela parceria nas horas de estudo.

Ao amigo André Muniz, pelas dicas em FPGA e VHDL.

Ao amigo André Ricardo, pela ajuda com o ModelSim®, com a qual muitas horas de verificação do projeto foram poupadas;

Ao meu orientador Nicolau, pela paciência e orientação, sem a qual não teria obtido todas as conquistas com o desenvolvimento deste trabalho;

À minha família Antonio, Iris e Lore por todo apoio e compreensão nos momentos mais difíceis;

À UFMS e ao CNPQ pelo fornecimento de estrutura e bolsa de estudos que contribuíram para a realização deste trabalho.

Resumo da Dissertação apresentada à UFMS como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

IMPLEMENTAÇÃO DO ALGORITMO DA MODULAÇÃO VETORIAL USANDO COORDENADAS MÓVEIS NÃO-ORTOGONAIS EM *FIELD-PROGRAMMABLE GATE ARRAY* PARA INVERSORES MULTINÍVEIS FONTE DE TENSÃO

Edvaldo Francisco Freitas Lima
Abril/2009

Orientador: Nicolau Pereira Filho, Doutor.
Área de Concentração: Energia, Planejamento, Operação e Controle de Sistemas Elétricos.
Palavras-chave: Inversores Multiníveis, Modulação Vetorial, FPGA e VHDL.
Número de Páginas: 142.

RESUMO:

Este trabalho apresenta a análise e implementação do algoritmo da Modulação Vetorial (MV) utilizando coordenadas móveis não-ortogonais para inversores multiníveis com diodo de grampeamento (DCI) em *Field Programmable Gate Array* (FPGA). Nesse algoritmo, a tensão de referência não-ortogonal é obtida de acordo com o setor onde o Vetor de Referência (V*) está localizado. A partir da identificação do triângulo dentro do hexágono, os Três Vetores mais Próximos (TVP) são determinados utilizando a informação do setor e do triângulo onde o V* está localizado. As razões cíclicas são calculadas por um conjunto de equações simples. O padrão de chaveamento é gerado por meio de coeficientes referenciados pelo número do triângulo em que o V* está localizado. Os *softwares*, QuartusII®, ModelSim® e MatLab® foram utilizados para descrição do algoritmo em linguagem de descrição de *hardware*, verificação, teste e simulação do trabalho. Os cálculos utilizaram padrão ponto-fixo de 16 bits do tipo *signed*. Um *clock* de 10 MHz é usado para obtenção do tempo de chaveamento, enquanto o modulador PWM trabalha com um *clock* de 50 MHz, de maneira a melhorar a precisão na geração do PWM. O sincronismo entre a obtenção do tempo de chaveamento e a geração dos sinais PWM foi feito por uma máquina de estados. O *kit* DK-CYCII-2C20N da Altera® com o FPGA EP2C20F484C7N da família *Cyclone II* foi utilizado para gerar os vetores de referência e desenvolver o algoritmo proposto. Os resultados obtidos com o inversor de três níveis DCI foram satisfatórios, validando a implementação do algoritmo no FPGA. Esse algoritmo pode ser estendido a topologias de inversores multiníveis DCI de ordem genérica, alterando muito pouco seu custo computacional.

Abstract of Dissertation presented to UFMS as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

DEVELOPMENT OF SPACE VECTOR PWM USIN NON-ORTHOGONAL REFERENCE FRAME FOR MULTILEVEL INVERTER VOLTAGE SOURCE IN FPGA

Edvaldo Francisco Freitas Lima

April/2009

Advisor: Nicolau Pereira Filho, Doctor.
Area of Concentration: Energy, Planning, Electric System, Operation and Control.
Keywords: Multi-Level Inverter, Vectorial Modulation, FPGA and VHDL.
Number of Pages: 142.

Abstract – This work presents the implementation and analysis of Space Vector PWM algorithm using non-orthogonal moving reference frame for diode clamped multilevel inverter in Field Programmable Gate Array (FPGA). In this algorithm, the non-orthogonal reference voltage is obtained according to the sector where the Reference Voltage (V*) lies. From the triangle identification inside hexagon, the Nearest Three Vector (NTV) are determined using the information of the sector and triangle where V* is located. The duty cycles are calculated by a set of simple equations. The switching pattern is generated through coefficients referred to by the triangle number where V* lies. The softwares Quartus II®, ModelSim® and MatLab® were used to describe the algorithm in hardware description language VHDL, to check, test and simulate work. Fixed point 16-bit signed pattern was used for calculus. A 10 MHz clock is used to obtain the switching time, whereas the PWM works with a 50 MHz clock, in order to improve the PWM generation accuracy. The synchronism between switching time calculation and the PWM signal generation was carried out by a state machine. Altera® Cyclone® II FPGA Starter Development Kit with EP2C20F484C7N FPGA, was used to generate the V* and develop the proposed algorithm. The results obtained with the DCI three-level inverter were satisfactory, validating the FPGA algorithm implementation. This algorithm can be extended to topologies of generic-ordered DCI multilevel inverters, very slightly altering its computational efforts.

SUMÁRIO

RESUMO:.....iv
NOMENCLATURA ix
1 INTRODUÇÃO 1
1.1 Justificativas 5
1.2 Objetivos 6
1.3 Organização do Trabalho 7
2 INVERSORES MULTINÍVEIS COM MODULAÇÃO VETORIAL 9
2.1 Inversores Multiníveis 9
2.2.1 Inversores Multiníveis com Diodo de Grampeamento 11
2.2 Algoritmo da Modulação Vetorial (SVPWM) 14
2.3 Algoritmo da Modulação Vetorial Via Coordenadas Móveis Não-Ortogonais 19
2.4 Conclusões 21
3 DESENVOLVIMENTO DO ALGORITMO DA MODULAÇÃO VETORIAL VIA COORDENADAS MÓVEIS NÃO ORTOGONAIS EM VHDL 22
3.1 Estratégia desenvolvida para cálculo na linguagem de descrição de *hardware* VHDL. 22
3.2 Fluxograma para desenvolvimento do algoritmo da Modulação Vetorial na linguagem VHDL 23
3.2.1 Geração do vetor de referência (Vd, Vq) 27
3.2.2 Identificação do sextante 29
3.2.3 Normalização do vetor de referência (Vg, Vh) 31
3.2.4 Identificação do triângulo e cálculo das razões cíclicas 33
3.2.5 Síntese do padrão de chaveamento para inversores multiníveis 37
3.2.6 Cálculos dos valores de comparação para geração dos sinais PWM 41
3.2.7 Geração do sinal de MLP com tempo morto 42
3.2.7.1 Processo de geração do sinal PWM 42
3.2.7.2 Processo de geração do tempo morto 43
3.2.7.3 Processo para Sincronismo entre o Cálculo do Vetor de Referência e a Geração do Sinal PWM 44

3.3 Conclusões 47
4 IMPLEMENTAÇÃO DO ALGORITMO DA MODULAÇÃO VETORIAL EM FPGA 48
4.1 Especificações do projeto desenvolvido em FPGA 48
4.2 Ferramentas Utilizadas para o Desenvolvimento do Algoritmo da Modulação Vetorial no FPGA 49
4.2.1 Ferramenta para determinação da frequência máxima de operação do circuito 49
4.2.2 Desenvolvimento do Algoritmo da Modulação Vetorial no MatLab® 51
4.2.3 Verificação dos Dados de Simulação do Algoritmo da Modulação Vetorial com o ModelSim® 51
4.2.4 Descrição e Simulação do Algoritmo da MV no Quartus II 52
4.3 Estrutura utilizada para síntese do algoritmo da Modulação Vetorial no FPGA 54
4.4 Características do FPGA com o Algoritmo de Modulação Vetorial 56
4.5 Visão Geral do *Hardware* Utilizado para Desenvolvimento do Projeto 58
4.6 Conclusões 59
5 RESULTADOS EXPERIMENTAIS 60
5.1 Resultados de simulação obtidos com os programas ModelSim® e MatLab®. 61
5.2.1 Simulação do vetor de referência 61
5.2.2 Simulação da normalização do vetor de referência nas coordenadas moveis não-ortogonais. 63
5.2.3 Simulação da identificação do triângulo e sextante 66
5.2.4 Simulação das razões cíclicas tg, th e tgh. 69
5.2.5 Simulação para o valor do comparador 73
5.2 Resultados experimentais com o algoritmo sintetizado no FPGA utilizando o *kit* de desenvolvimento DK-CYCII2C20N 75
5.2.1 Tempo Morto 77
5.2.2 Tensão entre fase para índice de modulação 0,30 77
5.2.3 Tensão entre fase para índice de modulação 0,48 79
5.2.4 Tensão entre fase para índice de modulação 0,55 80

5.2.5 Tensão entre fase para índice de modulação 0,63 81
5.2.6 Tensão entre fase para índice de modulação 0,70 82
5.2.7 Tensão entre fase para índice de modulação 0,90 83
5.3 Conclusões 84
6 CONCLUSÕES 85
REFERÊNCIAS 88
Anexo A – Geração do Vetor Referência no MatLab® 91
Anexo B – Desenvolvimento do Algoritomo da Modulação Vetorial no MatLab® 94
Anexo C – Verificação do Projeto Utilizando a Ferramenta ModelSim® 97
Anexo D – Desenvolvimento do Algoritmo da Modulação Vetorial em VHDL 109
Anexo E – Cálculos dos Tempos de Chaveamento em VHDL 116
Anexo F – Modulador PWM desenvolvido em VHDL 118
Anexo G – Cálculos em Ponto Fixo no FPGA 123

NOMENCLATURA

ASIC Application-Specific Integrated Circuit

CC Corrente Contínua

DSP Processador Digital de Sinais

FPGA Field-Programmable Gate Array

L_H Triângulo onde está localizado o vetor de referência no hexágono

L_S Triângulo onde está localizado o vetor de referência no sextante

Md Modo de operação do inversor

MLP Modulação por Largura de Pulso

MV Modulação Vetorial

n Número de Níveis do Inversor

PNG Ponto Neutro Grampeado

S Sextante onde está localizado o vetor de referência

SVM Space Vector Modulation

SVPWM Space Vector PMW

Tg Razão cíclica no eixo g

Tgh Complemento das razões cíclicas dos eixos g e h

th Razão cíclica no eixo h

THD Taxa de distorção harmônica

Triangle_type Tipo do triângulo (0 ou 1)

TVP Três Vetores Mais Próximos

Vd Tensão de referência no eixo direto

Vg Componente da tensão de referência no eixo g

V_{GF} Componente fracionária da tensão de referência no eixo g

V_{GU} Componente inteira da tensão de referência no eixo g

Vh Componente da tensão de referência no eixo h

VHDL VHSIC Hardware Description Language

V_{HF} Componente fracionária da tensão de referência no eixo h

V_{HU} Componente inteira da tensão de referência no eixo h

Vq Tensão de referência no eixo de quadratura

LISTA DE FIGURAS

Figura 2.1 - Braço de uma fase do inversor multiníveis com (a) dois níveis (b) três níveis, e (c) *n* níveis. 10
Figura 2.2 - Braço de uma fase do inversor multiníveis com diodo de grampeamento, (a) três níveis e (b) cinco níveis..... 13
Figura 2.3 - Inversor de dois níveis 15
Figura 2.4– Diagrama de estados de chaveamento para inversores de dois níveis 16
Figura 2.5 – Sequência de chaveamento da chave 1 da fases A, B e C para o sextante 1..... 17
Figura 2.6- Diagrama vetor de espaço para inversor de três níveis. 18
Figura 3.1– Fluxograma para desenvolvimento do algoritmo da modulação vetorial em VHDL..... 24
Figura 3.2 – Estrutura do algoritmo da modulação vetorial desenvolvida em FPGA. 27
Figura 3.3 – Componentes Vd e Vq para índice de modulação 0,70. 28
Figura 3.4 - Gráfico de Vd por Vq com índice de modulação 0,70. 29
Figura 3.5 – Código em VHDL para encontrar o valor da variável Ns. 30
Figura 3.6 – Localização do vetor de referência no sextante para índice de modulação 0,70. 31
Figura 3.7 – Componente Vg normalizada para índice de modulação 0,70..... 32
Figura 3.8 – Componente Vh normalizada para índice de modulação 0,70..... 32
Figura 3.9 - Modo de operação do inversor (M_d) e parte inteira e fracionária de Vg e Vh. 33
Figura 3.10 – Código em VHDL para determinar a parte inteira e fracionária de Vg e Vh. 34
Figura 3.11 – Localização do vetor de referência dentro do hexágono para índice de modulação 0,70..... 35
Figura 3.12 – Código em VHDL para encontrar os valores das variáveis Ls e Lh. 36
Figura 3.13 – Código em VHDL para cálculo das razões cíclicas. 37
Figura 3.14 – Triângulo 1 e seus vetores de chaveamento..... 38

Figura 3.15 – Sequência de chaveamento das chaves 1 e 2 da fase A para o triângulo 1 (Sextante ímpar). 38
Figura 3.16 – Triângulo 5 e seus vetores de chaveamento 39
Figura 3.17 – Sequência de chaveamento das chaves 1 e 2 da fase A para o triângulo 5 (Sextante par). 39
Figura 3.18 – Ilustração do modulador Ativo_Alto e Ativo_Baixo 41
Figura 3.19 – Modulação da largura de pulso do sinal de controle das chaves do inversor 42
Figura 3.20 – Ilustração da inserção de tempo morto no sinal de PWM. 43
Figura 3.21 – Diagrama em blocos do algoritmo para inserção do tempo morto nas chaves complementares. 44
Figura 3.22 – Máquina de estado para sincronismo entre a geração do vetor de referência e a geração do sinal MLP. 45
Figura 4.1 – Ligação dos *clocks* na parte serial e paralela do algoritmo. 50
Figura 4.2 – Gráfico com todos os valores das chaves do inversor para índice de modulação igual a 0,70. 52
Figura 4.3 – Período para geração de um padrão de chaveamento. 53
Figura 4.4 – Período da geração da forma de onda da tensão do inversor. 53
Figura 4.5 – Figura de ilustração da placa de desenvolvimento 55
Figura 4.6 – Diagrama em blocos do algoritmo da MV para inversores de três níveis desenvolvido em FPGA. 56
Figura 4.7 – Resumo da compilação do projeto completo para um inversor de três níveis. 57
Figura 4.8 – Resumo da compilação do bloco de desenvolvimento do algoritmo da MV simplificado. 57
Figura 4.9 – Resumo da compilação do bloco modulador PWM da chave 1 da fase A e sua chave complementar. 58
Figura 4.10 – *Hardware* utilizado para desenvolvimento do projeto. 59
Figura 5.1 – Cálculo do vetor de referência utilizando o MatLab®. 61
Figura 5.2 – Cálculo do vetor de referência em VHDL para índice de modulação 0,70. 62
Figura 5.3 – Diferença entre o cálculo do vetor de referência no MatLab e em VHDL para índice de modulação 0,70. 62

Figura 5.4 – Cálculo de Vg no MatLab com índice de modulação 0,70. 63
Figura 5.5 – Cálculo de Vg em VHDL com índice de modulação 0,70. 63
Figura 5.6 – Diferença da coordenada Vg calculada pelo MatLab e em VHDL com índice de modulação 0,70. 64
Figura 5.7 – Coordenada Vh calculada no MatLab para índice de modulação 0,70. 64
Figura 5.8 – Coordenada Vh calculada em VHDL para índice de modulação 0,70. 65
Figura 5.9 – Erro entre Vh calculado em VHDL e em MatLab® com índice de modulação 0,70. 65
Figura 5.10 – Cálculo do sextante em MatLab®. 66
Figura 5.11 – Cálculo do sextante em VHDL. 66
Figura 5.12 – Diferença entre os cálculos do valor do sextante em VHDL e em MatLab® para índice de modulação 0,70. 67
Figura 5.13 – Valor de Lh calculado em MatLab® para índice de modulação 0,70 para índice de modulação 0,70. 67
Figura 5.14 – Variável Lh calculada em VHDL para índice de modulação 0,70... 68
Figura 5.15 – Erro entre o cálculo de Lh feito em VHDL e MatLab® para índice de modulação 0,70. 68
Figura 5.16 – Valor de tg calculado em MatLab® para índice de modulação 0,70. 69
Figura 5.17 – Valor de tg calculado em VHDL para índice de modulação 0,70. 69
Figura 5.18 – Erro de cálculo entre a variável tg calculada em MatLab e em VHDL. 70
Figura 5.19 – Valor de th calculado em MatLab® para índice de modulação 0,70. 70
Figura 5.20 – Valor de th calculado em VHDL para índice de modulação 0,70. .. 71
Figura 5.21 – Erro de cálculo entre a variável th calculada em MatLab® e em VHDL. 71
Figura 5.22 – Valor de tgh calculado em MatLab® para índice de modulação 0,70. 72

Figura 5.23 – Valor de tgh calculado em VHDL para índice de modulação 0,70. 72
Figura 5.24 – Erro de cálculo entre a variável tgh calculada em MatLab e em VHDL. 73
Figura 5.25 – Valor de comparação da chave 1 da fase A calculado em MatLab®. 74
Figura 5.26 – Valor de comparação da chave 1 da fase A calculado em VHDL. . 74
Figura 5.27 – Erro entre os valores do comparador da chave 1 da fase A calculados em MatLab® e em VHDL. 75
Figura 5.28 – Visão geral do protótipo implementado. 76
Figura 5.29 – FPGA, Acoplamento Óptico e Inversor de Três Níveis. 76
Figura 5.30 - Tempo morto entre as chaves complementares. 77
Figura 5.31 – Tensão de linha para índice de modulação 0,30. 78
Figura 5.32 – Espectro harmônico para índice de modulação 0,30. 78
Figura 5.33 – Tensão de linha para índice de modulação 0,48. 79
Figura 5.34 – Espectro harmônico para índice de modulação 0,48. 80
Figura 5.35 – Tensão de linha para índice de modulação 0,55. 80
Figura 5.36 – Espectro harmônico para índice de modulação 0,55. 81
Figura 5.37 – Tensão de linha para índice de modulação 0,63. 81
Figura 5.38 – Espectro harmônico para índice de modulação 0,63. 82
Figura 5.39 – Tensão de linha para índice de modulação 0,70. 82
Figura 5.40 – Espectro harmônico para índice de modulação 0,70. 83
Figura 5.41 – Tensão de linha para índice de modulação 0,90. 83
Figura 5.42 – Espectro harmônico para índice de modulação 0,90. 84

LISTA DE TABELAS

Tabela 2.1 – Relação dos estados de chaveamento e estado das chaves. 13
Tabela 3.1 – Relação entre o sextante e N_S 30
Tabela 4.1– Tabela com as características do FPGA EP2C20F484C7N. 55

1 INTRODUÇÃO

Atualmente, a indústria tem iniciado uma alta demanda por equipamentos de alta potência que chegam ao nível de *megawatts* [1]. Com isso, o campo dos conversores de alta potência tem sido uma das mais atrativas áreas em pesquisa e desenvolvimento da eletrônica de potência nos últimos anos [2]. Hoje em dia, é complicado conectar um simples semicondutor de potência diretamente na rede de média tensão (2,3; 3,3; 4,16 ou 6,9 kV). Por esta razão, os inversores multiníveis têm surgido como uma solução para se trabalhar com tais níveis de tensão [3]. A sua crescente demanda no uso de aplicações que exigem alta potência se deve a seu rendimento superior em comparação aos inversores de dois níveis, entre eles, baixa taxa de distorção harmônica, baixa *dv/dt* e as chaves semicondutoras que são submetidas a uma tensão reduzida [3].

As pesquisas atuais sobre estratégias de controle e modulação de inversores buscam aproveitar ao máximo as vantagens das topologias multiníveis. As otimizações dessas estratégias buscam diminuir o número de chaveamento durante o período de chaveamento e assegurar uma baixa taxa de distorção harmônica da tensão de saída do inversor [4].

Atualmente, a topologia de inversores multiníveis tem despertado grande interesse no campo de aplicações em alta tensão e alta corrente. Nesses conversores, a qualidade da tensão de saída do inversor melhora, aumentando-se o número de níveis disponíveis, o que reduz a taxa de distorção harmônica da tensão de saída do inversor. [5]

Os inversores multiníveis são baseados na síntese de uma forma de onda a partir dos níveis de tensão disponíveis pela divisão da tensão do elo CC do inversor. Com o aumento dos níveis do inversor, a tensão de saída apresenta mais degraus fazendo com que o inversor produza uma tensão de saída cada vez mais semelhante à tensão de referência. As maiores vantagens do desenvolvimento de um inversor multiníveis são: [6]

a) alta tensão de saída, porém os semicondutores de potência são submetidos a uma tensão limitada;

b) baixa taxa de distorção harmônica;

c) redução das perdas de chaveamento;

d) grande eficiência;

e) boa compatibilidade eletromagnética.

As aplicações industriais dos inversores multiníveis envolvem acionamento de máquinas, retificadores ativos, interface para fonte de energia renováveis e compensadores síncronos estáticos [6].

A tecnologia de conversores multiníveis fonte de tensão foi primeiramente apresentada por Baker [7] em 1979, com a introdução da topologia do conversor com ponto neutro grampeado de três níveis.

O termo multiníveis teve início com o inversor de três níveis apresentado pelo trabalho de Nabae *et al.* [8] no desenvolvimento de um inversor com diodo de grampeamento de três níveis com modulação PWM. Os inversores multiníveis apresentam basicamente três topologias: com diodo de grampeamento, com capacitor flutuante e com ponte-H em cascata.

A topologia com diodo de grampeamento faz com que todos os semicondutores de potência operem com uma tensão menor que a tensão do elo CC do inversor. Essa topologia de inversores é uma solução simples para aumentar a tensão e potência dos inversores em relação aos inversores de dois níveis, os quais são severamente limitados devido à tensão de bloqueio dos semicondutores de potência.

A topologia com capacitor flutuante foi proposta há mais de quinze anos de acordo com o trabalho de Rodriguez et al. [2]. Nessa topologia de inversor, cada capacitor é carregado em diferentes níveis de tensão dependendo dos estados de chaveamento. O capacitor e a fonte CC são conectados de diferentes maneiras e produzem várias tensões de saída.

A topologia com ponte-H iniciou-se em 1988 de acordo com Marchesoni et al. [9] e obteve seu amadurecimento nos anos de 1990 de acordo com [2]. A topologia de inversores multiníveis com ponte-H é composta pela conexão em série de células de potência em ponte-H. Por essa razão, a topologia ponte-H é conhecida como inversor multicélulas. Cada célula inclui um inversor monofásico

em ponte-H, um elo CC capacitivo, um retificador e uma fonte de tensão isolada fornecida por uma bateria ou um transformador.

Na literatura são apresentadas várias técnicas de modulação para inversores multiníveis que podem ser classificadas de acordo com a freqüência de chaveamento [3]. Técnicas com várias comutações dos semicondutores de potência no período fundamental da tensão de saída são classificadas como métodos de modulação de alta freqüência. Uma técnica de modulação muito popular dessa categoria é a modulação Senoidal PWM. Um outro exemplo dessa categoria é a modulação vetorial (MV) "*Space Vector Modulation*".

Técnicas de modulação que apresentam baixa frequência de chaveamento, uma ou duas comutações dos semicondutores de potência durante um ciclo da tensão de saída do inversor, têm com exemplo a técnica Eliminação Seletiva de Harmônicas [3].

Dentre todas as técnicas de modulação utilizadas em inversores multiníveis, a técnica de Modulação Vetorial (MV) *Space Vector Pulsewidth Modulation (SVPWM)* é a que mais se destaca entre elas devido às seguintes vantagens:

a) implementação digital;

b) Baixa Taxa de Distorção Harmônica (THD) do sinal de saída;

c) otimização da seqüência de chaveamento;

d) operação na faixa linear é estendida.

As implementações do algoritmo da MV baseiam-se principalmente em DSP's ou Microprocessadores e essas implementações requerem uma dedicação muito grande dos seus recursos computacionais para a execução do algoritmo e a geração dos sinais PWM, limitando o processador para o uso em outras tarefas.

A implementação do algoritmo da Modulação Vetorial (MV) *Space Vector Pulse With Modulation*, em FPGA, já vem sendo explorada desde a década de 1990, com implementações para inversores de dois níveis [10]. Atualmente, com FPGA's mais rápidos, com mais disponibilidade de células lógicas e também, devido às simplificações propostas para o algoritmo da MV, implementações do

algoritmo para inversores de três níveis vêm sendo desenvolvidas [5] [11]. A implementação do algoritmo da MV em FPGA é um desafio, uma vez que o algoritmo deve ser simplificado e de fácil implementação digital.

O desenvolvimento do algoritmo da MV em FPGA apresenta a vantagem de aliviar o DSP da carga de processamento exigida pelo algoritmo, liberando o processamento para outras tarefas, tais como controle aprimorado do motor, predição de erros e proteção do sistema desenvolvido. Além disso, o FPGA apresenta grande flexibilidade, tais como: a) síntese de funções lógicas, b) síntese de circuitos em paralelo, c) cálculos em ponto fixo, d) cálculos em ponto flutuante, d) utilização de funções parametrizadas, dentre outras.

A estratégia apresentada no trabalho [4] para simplificação do algoritmo da MV utiliza coordenadas móveis não-ortogonais. Nesse algoritmo, a tensão de referência não-ortogonal é obtida de acordo com o setor em que o Vetor de Referência (V*) está localizado. A partir da identificação do triângulo dentro do hexágono, os Três Vetores mais Próximos (TVP) são determinados utilizando a informação do setor e do triângulo em que o V* está localizado. As razões cíclicas são calculadas por um conjunto de equações simples. O padrão de chaveamento é gerado por coeficientes referenciados pelo número do triângulo em que o V* está localizado.

Esse algoritmo simplificado apresenta as seguintes etapas para seu desenvolvimento:

a) identificação do sextante;

b) conversão do V* para coordenadas não-ortogonais;

c) identificação do triângulo onde V* está localizado;

d) cálculo das razões cíclicas;

e) geração dos sinais PWM.

Assim, neste trabalho, é desenvolvido em FPGA o algoritmo da MV utilizando coordenadas móveis não-ortogonais para inversores PNG de três níveis. A implementação em FPGA compreende o desenvolvimento do algoritmo

da MV, geração dos sinais de chaveamento com inserção do tempo morto e também a geração dos vetores de referência (Vd,Vq).

1.1 **Justificativas**

Neste trabalho, utilizou-se a topologia com diodo de grampeado. Essa topologia apresenta as seguintes vantagens [4]:

a) as chaves são submetidas à baixa tensão;

b) o conteúdo harmônico da tensão de saída do inversor é centrado em duas vezes a freqüência de chaveamento;

c) esta topologia pode ser generalizada para qualquer número de níveis.

Os inversores multiníveis surgiram com o objetivo de diminuir a tensão submetida aos dispositivos semicondutores de potência que compõem o inversor. Por consequência houve uma melhoria do conteúdo harmônico da tensão da corrente de saída do inversor em relação aos inversores de dois níveis. A tensão de saída dos inversores multiníveis apresenta diferentes níveis e seu conteúdo harmônico melhora quanto maior for a ordem do inversor.

A busca pela melhoria na qualidade da tensão de saída dos inversores impulsiona cada vez mais o incremento da sua quantidade de níveis. Porém, isso causa o aumento na complexidade do inversor e, consequentemente, na complexidade do seu algoritmo de modulação.

O algoritmo da MV é o que mais desperta interesse dos pesquisadores por causa das seguintes características: a) implementação digital, b) baixa Taxa de Distorção Harmônica do sinal de saída c) poder de otimizar a sequência de chaveamento e d) operação na faixa linear estendida.

A topologia de inversores multiníveis com diodo de grampeamento vem ganhando cada vez mais espaço devido: a) às chaves serem submetidas a baixas tensões, b) ao espectro harmônico da tensão de saída ser centrado em duas vezes a frequência de chaveamento e c) a essa topologia que pode ser generalizada para inversores de n-níveis. Essa topologia ainda possibilita a

conexão do inversor à rede de média e alta tensão. No entanto, apresenta algumas desvantagens, tais como a necessidade de diodos de alta velocidade e o desbalanço dos capacitores do elo *CC* para inversores maiores que três níveis, ainda é um assunto em pesquisa.

Tendo em mente que um dos objetivos do desenvolvimento em FPGA é a baixa utilização de seus recursos lógicos, o algoritmo da Modulação Vetorial, em que utiliza coordenadas móveis não-ortogonais proposto em [3], foi escolhido para o desenvolvimento deste trabalho por apresentar fácil implementação digital, apresentar simplificações que facilitam seu desenvolvimento em VHDL e ainda por ser de ordem genérica.

Desde a década de 1990, a implementação do algoritmo da MV em FPGA já vem sendo explorada, com implementações para inversores de dois níveis [10]. Atualmente, com FPGA's mais rápidos, com mais disponibilidade de células lógicas e também devido às simplificações propostas para o algoritmo da MV, a implementação do algoritmo para inversores de três níveis vem sendo desenvolvida [5][11].

A implementação do algoritmo da MV em FPGA é um desafio pois o algoritmo deve ser simplificado, ou seja de baixa complexidade computacional. O FPGA apresenta grande flexibilidade de desenvolvimento com o poder de síntese de circuitos paralelos e dispõe também de uma grande quantidade de pinos de saída, suportando uma implementação do algoritmo SVPWM para inversores de três níveis, bem como para inversores de ordens superiores.

O desenvolvimento do algoritmo da MV em FPGA surge como uma alternativa às implementações convencionais com DSP's, uma vez que, atualmente, não se tem conhecimento da disponibilidade no mercado, DSP's com módulos PWM que suporte a implementação desse algoritmo para inversores multiníveis de ordem elevada [11].

1.2 **Objetivos**

Este trabalho tem os objetivos: a) desenvolver em FPGA o algoritmo da MV utilizando coordenadas móveis não-ortogonais para inversores multiníveis de

ordem genérica; b) fazer a codificação do algoritmo da MV em linguagem de descrição de *hardware* (VHDL) explorando a capacidade de síntese de circuitos em paralelo do FPGA para diminuir o tempo de processamento do algoritmo; c) avaliar o desempenho do algoritmo por meio de um inversor de três níveis, analisando sua tensão de saída com relação ao conteúdo harmônico e frequência fundamental.

E após a síntese do projeto, analisar a freqüência de operação do circuito desenvolvido e fazer as adequações necessárias para que ele trabalhe na frequência de operação desejada (Cálculos a 10 MHz e modulador PWM a 50 MHz).

Adequar o algoritmo à freqüência de chaveamento do inversor ajustando o valor máximo do contador. Definir a freqüência fundamental da tensão de saída do inversor. Determinar a taxa de amostragem, ou seja, quantos pontos da tensão de referência serão sintetizados pelo inversor. Fazer a sincronização das etapas do algoritmo e, por fim, inserir o tempo morto entre os sinais de controle das chaves complementares do inversor.

Verificar o projeto desenvolvido em FPGA por meio da comparação entre os resultados obtidos com o algoritmo desenvolvido em MatLab® e o algoritmo descrito em VHDL. Com o uso das ferramentas ModelSim®, projetar o *testbench* e MatLab® para gerar os dados de comparação e fazer a verificação do projeto desenvolvido em FPGA.

Validar o projeto, por meio de um inversor de três níveis com diodo de grampeamento utilizando o algoritmo da MV sintetizado no FPGA EP2C20F484C7N da Altera® e analisar as formas de onda obtidas com a simulação do algoritmo para diferentes índices de modulação.

1.3 **Organização do Trabalho**

O presente trabalho está dividido em 6 capítulos. Neste capítulo introdutório foi feita uma introdução sobre inversores multiníveis, algoritmos de modulação e algumas características da implementação do algoritmo da MV

implementado no FPGA. Também são apresentados a justificativa e os objetivos do trabalho proposto para a dissertação.

No capítulo 2, são abordados os inversores multiníveis com modulação vetorial, com uma breve abordagem sobre inversores multiníveis e o algoritmo de modulação utilizado no desenvolvimento do projeto.

No capítulo 3, são apresentadas as etapas para desenvolvimento do algoritmo e as estratégias utilizadas em seu desenvolvimento, utilizando-se da linguagem de descrição de *hardware* VHDL.

No capítulo 4, são apresentadas as ferramentas utilizadas no desenvolvimento do projeto, Quartus II®, ModelSim® e MatLab®, a estrutura interna do FPGA para desenvolvimento do algoritmo, as análises e a adequação do projeto para sua síntese no FPGA EP2C20F484C7N .

O capítulo 5 é destinado à apresentação dos resultados obtidos com o desenvolvimento do projeto e com a sua simulação, utilizando os programas QuartusII® em conjunto com o ModelSim®, e também os resultados das formas de onda obtidas com a simulação do projeto em um inversor de três níveis.

No capitulo 6 apresentam-se as conclusões do trabalho e propostas para estudos futuros.

2 INVERSORES MULTINÍVEIS COM MODULAÇÃO VETORIAL

Neste capítulo, são apresentadas as abordagens sobre o algoritmo da modulação vetorial e a topologia de inversores multiníveis com diodo de grampeamento, utilizados no desenvolvimento deste projeto.

2.1 **Inversores Multiníveis**

Atualmente, indústrias têm iniciado uma grande demanda por equipamentos de alta potência que chegam a níveis de *megawatts*, sendo necessário o desenvolvimento de controladores para esse tipo de equipamentos. No entanto, a conexão de dispositivos semicondutores diretamente na rede de média tensão é muito difícil, já que esses dispositivos não suportam tal tensão elevada. Os inversores multiníveis surgem como uma alternativa para contornar esse problema, uma vez que, em sua estrutura, a tensão é dividida em diferentes níveis e distribuída entre as chaves do inversor.

No ano de 1981, uma nova topologia de inversores multiníveis foi proposta por Nabae *et. al* em [8], a topologia do conversor de três níveis fonte de tensão com diodo de grampeamento. Nessa topologia, todos os semicondutores operam com uma tensão que corresponde à metade da tensão do elo *CC* do inversor. Essa topologia oferece uma alternativa para aumentar as faixas de potência dos conversores em relação aos inversores de dois níveis, pois esses conversores são severamente limitados devido à tensão de bloqueio dos dispositivos semicondutores. O inversor de três níveis com diodo de grampeamento pode ser estendido para gerar uma tensão de saída com vários níveis [2], ou seja, ser estendido a uma topologia multiníveis.

Inversores multiníveis podem ser compostos por um arranjo de semicondutores de potência, capacitores e fontes de tensão. Sua saída apresenta uma tensão com forma de onda que se assemelha a uma escada, ou seja, uma forma de onda com vários níveis de tensão. A comutação das chaves permite a adição de tensões de capacitores que podem atingir uma tensão elevada na

saída, enquanto os semicondutores de potência devem suportar apenas as tensões reduzidas [3].

Na Figura 2.1, é mostrado o diagrama esquemático do braço de uma fase de um inversor de dois, três e com vários níveis.

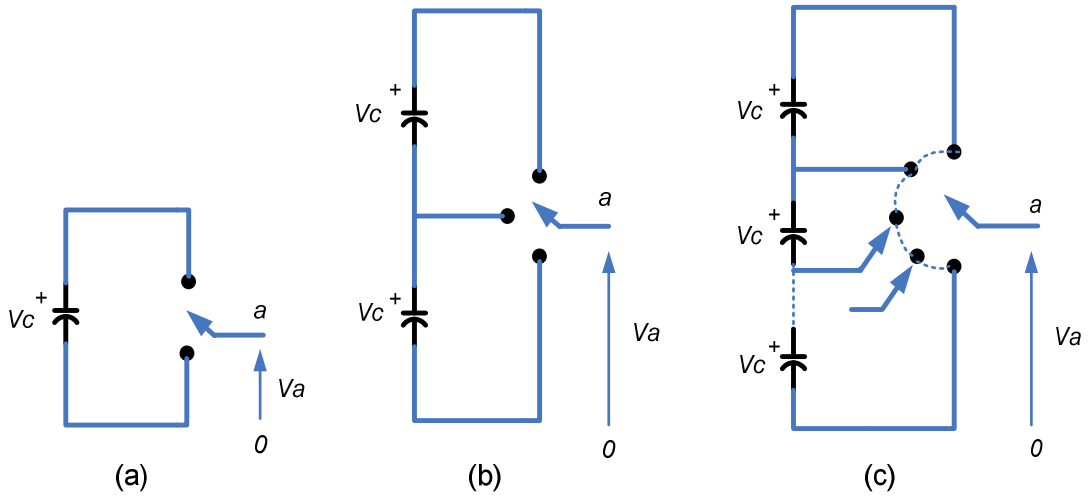


Figura 2.1 - Braço de uma fase do inversor multiníveis com (a) dois níveis (b) três níveis, e (c) *n* níveis.

Um inversor com dois níveis disponibiliza uma tensão de saída com dois níveis de tensão em relação ao terminal de referência, um inversor com três níveis disponibiliza três níveis de tensão, e assim por diante.

De acordo com Nabae *et. al* [8] um inversor é considerado multiníveis quando apresenta, em sua saída, três ou mais níveis de tensão. O incremento do número de níveis de tensão do inversor faz com que a tensão de saída apresente o formato de uma escada, isso diminui a distorção harmônica da tensão de saída do inversor. No entanto, o incremento da quantidade de níveis do inversor faz com que seu controle se torne cada vez mais complexo, exigindo cada vez mais esforço computacional para a execução do algoritmo.

Esse aumento do esforço computacional se deve pelo aumento da quantidade de chaves do inversor e, consequentemente, ao aumento na quantidade de vetores de chaveamento que devem ser identificados pelo algoritmo.

As características mais atrativas para o desenvolvimento dos inversores multiníveis destacadas por [3] são:

a) podem gerar tensões de saída com distorção harmônica extremamente baixa e com menor dv/dt;
b) drenam a corrente de entrada com distorção harmônica extremamente baixa;
c) geram tensão de modo comum inferior, reduzindo assim o estresse dos rolamentos do motor. Com a utilização de métodos sofisticados de modulação, as tensões de modo comum podem ser eliminadas;
d) podem operar com uma frequência de chaveamento baixa.

Existem três topologias básicas de inversores multiníveis propostas na literatura, são elas: inversores multiníveis com diodos de grampeamento *(diode-clamped multilevel inverter)*, inversores multiníveis com capacitores flutuantes *(flying-capacitors multilevel inverter)* e inversores multiníveis com módulos ponte-H em cascata *(H-bridge multilevel inverter)*.

A topologia com diodo de grampeamento apresenta como uma das suas principais caracteristicas a divisão da tensão do elo CC fazendo com que as chaves do inversor sejam submetidas a baixas tensões. Essa é uma das características que levaram a utilização desta topologia para o desenvolvimento desse trabalho. Esta topologia de inversor multiníveis será discutida a seguir.

2.2.1 Inversores Multiníveis com Diodo de Grampeamento

A topologia de inversores multiníveis com diodo de grampeamento surgiu com o trabalho proposto por Nabae *et. al.* em [8] na implementação de um inversor de três níveis. O trabalho propôs uma topologia com melhores vantagens em relação à topologia de inversores de dois níveis.

Na topologia de inversores com três níveis, além de a tensão de saída apresentar um menor conteúdo harmônico, ela possibilita a utilização de dispositivos semicondutores com a metade do limite de tensão dos dispositivos utilizados nas topologias de dois níveis.

As vantagens que um inversor multiníveis com diodo de grampeamento apresenta são as seguintes:

a) as chaves são submetidas a uma fração da tensão do elo *CC*;

b) a primeira ordem de harmônicas é centrada em torno de duas vezes a freqüência de chaveamento;

c) esta topologia pode ser generalizada e estendida a topologias de *n*-níveis.

Os diodos de grampeamento de um inversor multiniveis devem, entretanto, suportar diferentes níveis de tensão reversa assumindo que cada diodo de grampeamento possui a mesma especificação de tensão dos interruptores o número de diodos de grampeamento cresce rapidamente, e é igual a 3(n–1)(n–2). Por exemplo, em um inversor de três níveis, o número de diodos de grampeamento é 3(3-1)(3-2) o que corresponde a seis diodos de grampeamento; já numa topologia de cinco níveis, o número de diodos de grampeamento é 3(5-1)(5-2), ou seja, são trinta e seis diodos de grampeamento.

A topologia de inversores multiníveis com diodo de grampeamento também apresenta algumas desvantagens por exemplo:

a) essa topologia requer diodos grampeadores de alta velocidade que sejam capazes de drenar a corrente de plena carga e são submetidos à estresse de recombinação reversa.

b) para topologias com mais de três níveis, os diodos grampeadores são sujeitos ao aumento do estresse de tensão para Vcc (n-1)/n. Portanto, as conexões de diodos em série podem ser requeridas, o que torna o projeto mais complexo e se aumentam as preocupações com segurança e custos.

c) o problema de manter o balanço de carga dos capacitores permanece um tema aberto para a solução em topologias PNG com mais de três níveis.

Na Figura 2.2 é apresentado um braço de uma fase de um inversor de três níveis (a) e cinco níveis (b) com a topologia de inversores multiníveis diodo de grampeamento.

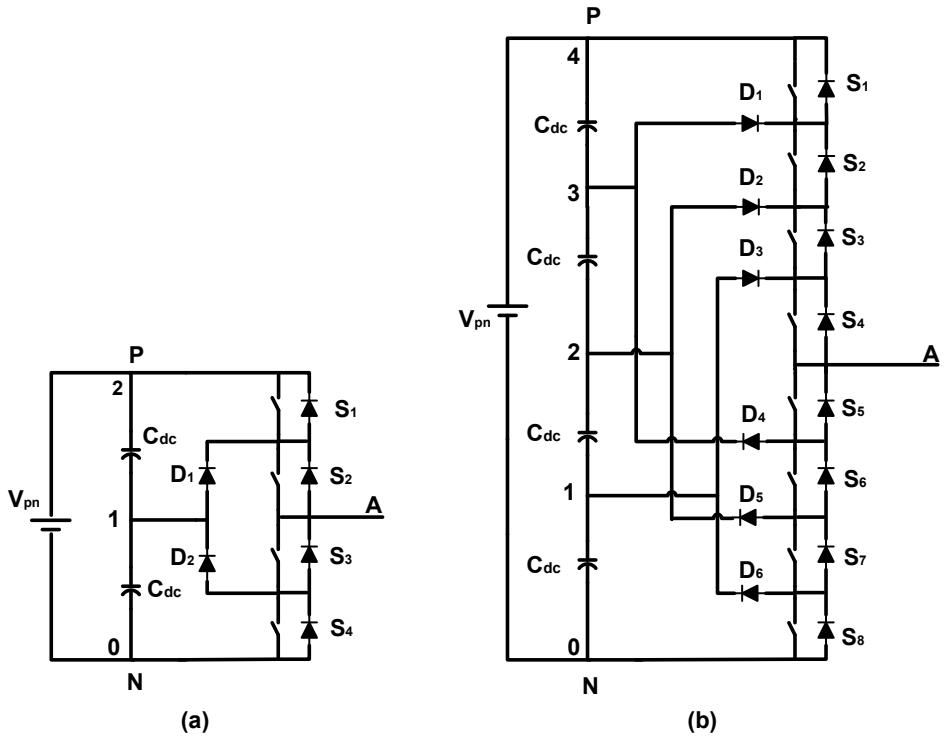


Figura 2.2 - Braço de uma fase do inversor multiníveis com diodo de grampeamento, (a) três níveis e (b) cinco níveis.

A Tabela 2.1 apresenta os estados de chaveamento 0, 1 e 2 de um inversor de três níveis, com relação ao ponto **N** e os respectivos estados das chaves do inversor.

Tabela 2.1 – Relação dos estados de chaveamento e estado das chaves.

Nível de Tensão	Tensão de saída Van	S_1	S_2	S_3	S_4
2	V_{DC}	Ligado	Ligado	Desligado	Desligado
1	$V_{DC}/2$	Desligado	Ligado	Ligado	Desligado
0	0	Desligado	Desligado	Ligado	Ligado

Nos inversores de três níveis, a tensão de saída Van pode assumir um dos três estados: 0, $V_{DC}/2$ e V_{DC}, corresponde aos níveis de tensão 0, 1 e 2 respectivamente. Para que seja obtido na saída Van o nível de tensão V_{DC}, é preciso que as chaves S_1 e S_2 estejam ligadas e as chaves S_3 e S_4, desligadas. Para obter o nível $V_{DC}/2$, é preciso que as chaves S_1 e S_4 estejam desligadas e as

chaves S_2 e S_3,ligadas. O nível 0 é obtido com as chaves S_1 e S_2 desligadas e as chaves S_3 e S_4, ligadas.

Para que um inversor de tensão disponibilize na sua saída uma forma de onda da tensão desejada é preciso definir uma estratégia de modulação que definirá o período de ativação de cada uma das chaves do inversor. A literatura apresenta diversos algoritmos para controle de inversores multiníveis, além de oferecer estratégias de modulação para a tensão de saída do inversor.

O algoritmo utilizado neste trabalho para o controle do inversor é o algoritmo da Modulação Vetorial por Largura de Pulso (*Space Vector Pulse With Modulation "SVPWM"*). Esse algoritmo será utilizado pela possibilidade de implementação digital, por apresentar vantagens em relação aos outros algoritmos de modulação, por exemplo, sua qualidade harmônica superior e faixa de operação linear estendida, e por apresentar na literatura uma simplificação proposta no trabalho [4] que possibilita uma facilidade na descrição do algoritmo em linguagem de descrição de *hardware* e sua implementação em um FPGA.

A seguir, o algoritmo da Modulação Vetorial será discutido mostrando as suas características e vantagens de implementação.

2.2 **Algoritmo da Modulação Vetorial (SVPWM)**

Existem diferentes esquemas de modulação para gerar a desejada tensão convertida. Os métodos de modulação mais utilizados em inversores multiníveis são: a modulação senoidal e a modulação vetorial (*Space Vector Modulation)*. Outros tipos de modulação como *Selective Harmonic Elimination* (SHE) também têm sido adaptados para formas de ondas multiníveis [2].

Dentre todas as técnicas de modulação para inversores multiníveis, a Modulação Vetorial *Space Vector PWM (SVPWM)* é a mais atrativa por causa da sua alta qualidade harmônica, e estendida faixa de operação linear.

Na Figura 2.3 é apresentado um inversor fonte de tensão de dois níveis, constituído por seis chaves de maneira que as chaves SW1, SW2 e SW3 são complementares às chaves SW1', SW2' e SW3', respectivamente.

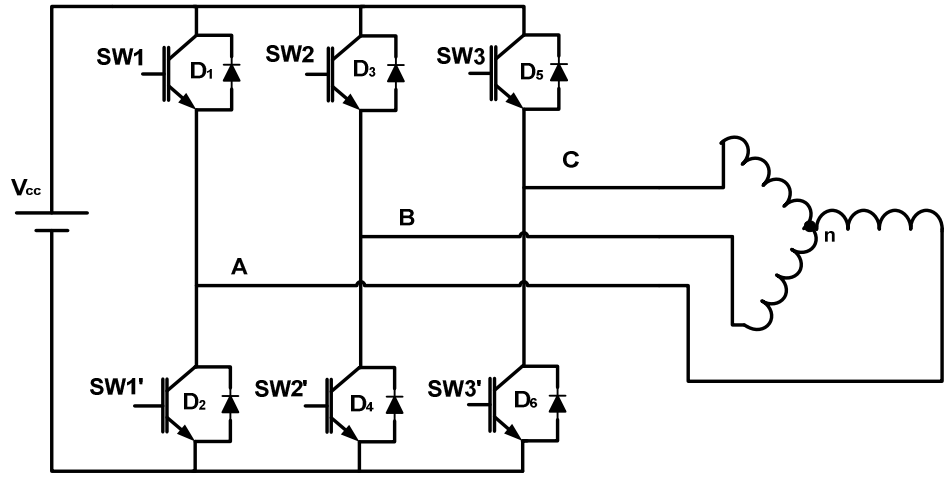


Figura 2.3 - Inversor de dois níveis

Em um algoritmo de modulação vetorial, o objetivo é aproximar a tensão de referência à tensão Van (Figura 2.2), instantaneamente, por meio da combinação dos estados de chaveamento correspondentes aos vetores de espaço [12].

Para sintetizar a forma de onda da tensão desejada na saída do inversor de dois níveis é preciso fazer a correta combinação entre todas as chaves do inversor. Essa sequência de combinação é denominada estado de chaveamento. Esses estados são dispostos em vetores nos vértices do hexágono apresentado na Figura 2.4.

Na Figura 2.4 observam-se o hexágono e os sextantes (S1 – S6) com seus respectivos vetores de chaveamento (V_1 – V_6) em seus vértices para um inversor de dois níveis.

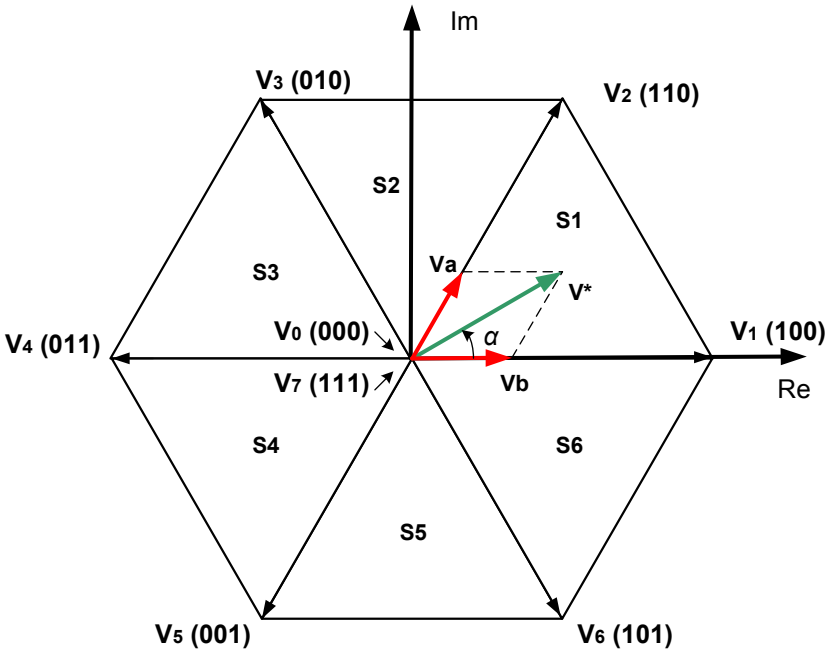


Figura 2.4– Diagrama de estados de chaveamento para inversores de dois níveis

A síntese da tensão de saída de um inversor pelo algoritmo da MV consiste em determinar em cada período de chaveamento os estados de cada uma das chaves e o tempo de ativação de cada uma delas.

Os estados das chaves em cada período de chaveamento são determinados a partir da localização do vetor de referência em um dos sextantes do diagrama de estados de chaveamento apresentado na Figura 2.4. Após a determinação do sextante, os vetores de chaveamento são sequenciados. Para que apenas uma chave seja alterada a cada troca de vetor, os três vetores adjacentes do sextante são escolhidos. No sequenciamento dos vetores, ainda, é utilizado o padrão de síntese completo, no qual todos os vetores que compõem os vértices do sextante são utilizados.

Supondo que o vetor de referência esteja localizado no sextante de número 1(um), têm-se os seguintes vetores de chaveamento para formar a tensão desejada na saída do inversor 111/000, 100 e 110. A Figura 2.5 apresenta a sequência dos vetores e o estado das chaves do inversor.

	Sequência Direta				Sequência Reversa			
A	0	1	1	1	1	1	1	0
B	0	0	1	1	1	1	0	0
C	0	0	0	1	1	0	0	0
Duty-cycle	**t0/2**	**tb**	**ta**	**t0/2**	**t0/2**	**ta**	**tb**	**t0/2**
Chave 1A	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*off*
Chave 1B	*off*	*off*	*on*	*on*	*on*	*on*	*off*	*off*
Chave 1C	*off*	*off*	*off*	*on*	*on*	*off*	*off*	*off*

Figura 2.5 – Sequência de chaveamento da chave 1 da fases A, B e C para o sextante 1.

As razões cíclicas na região linear de cada chave do inversor são obtidas pelas seguintes expressões:

$$ta = 2\frac{\sqrt{3}Ts}{4Vcc}V^{*}sen\left(\frac{\pi}{3}-\alpha\right)$$

$$tb = 2\frac{\sqrt{3}Ts}{4Vcc}V^{*}sen\left(\frac{\pi}{3}\right)$$

$$t0 = \frac{Ts}{2}-(ta+tb)$$

Onde *Ts* é o tempo de amostragem, *V** é o vetor de referência, *Vcc* é o valor da tensão do elo *CC* do inversor e α é o ângulo do vetor tensão de referência.

O diagrama de vetor de espaço de qualquer inversor trifásico de *n*-níveis é composto de seis setores. Cada setor consiste de $(n-1)^2$ triângulos, onde "*n*" corresponde à quantidade de níveis do inversor. Em um inversor de dois níveis, o setor contém $(2-1)^2$, ou seja, um triângulo por sextante, já, em um inversor de três níveis, têm-se $(3-1)^2$, portanto, cada sextante apresenta quatro triângulos.

O vetor de referência pode estar localizado em qualquer um dos triângulos do hexágono. Cada vértice do triângulo representa um vetor de chaveamento. Um vetor de chaveamento representa um ou mais estados de chaveamento dependendo da sua localização. Existem n^3 estados de chaveamento no diagrama de estados de chaveamento de um inversor n-níveis, por exemplo, para um inversor de dois níveis são 2^3, ou seja, são oito estados de chaveamento. Em um inversor de três níveis, são 3^3, logo há vinte e sete estados de chaveamento.

A Modulação Vetorial é caracterizada por selecionar e executar o estado de chaveamento de um dado triângulo para os respectivos ciclos de chaveamento. Isso também é conhecido como aproximação dos Três Vetores mais Próximos (TVP). O comportamento do inversor depende significantemente da seleção desses vetores de chaveamento [14].

A figura 2.6 mostra o diagrama do vetor espaço de estados de um inversor de três níveis. Existem seis setores (S1 – S6), quatro triângulos em cada setor e um total de 27 estados de chaveamento neste diagrama.

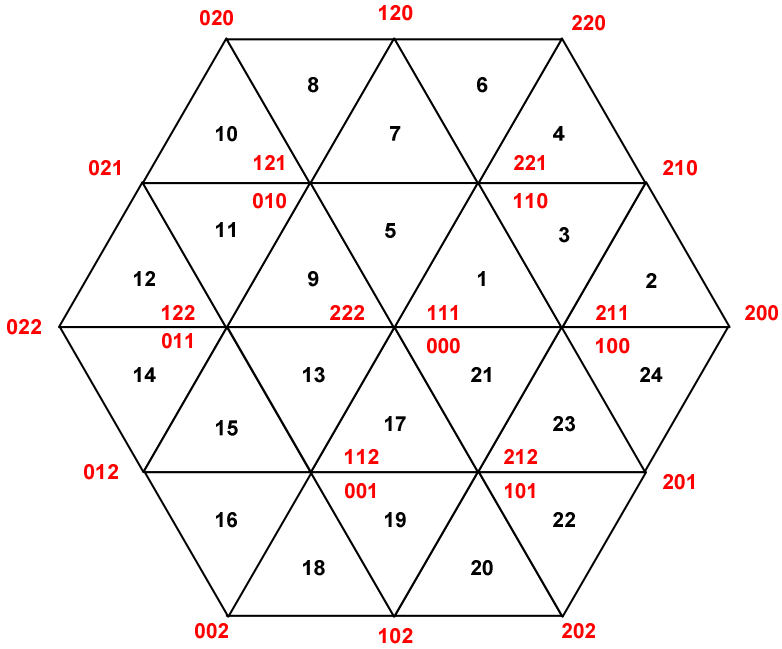


Figura 2.6- Diagrama vetor de espaço para inversor de três níveis.

Para a síntese da tensão de saída do inversor, o algoritmo da MV tem disponível uma grande quantidade de vetores de chaveamento, porém apenas os Três Vetores mais Próximos (TVP) são utilizados para a síntese. Os TVP são os vetores que se encontram nos vértices do triângulo onde o vetor de referência está localizado. Supondo que o vetor de referência esteja localizado no primeiro sextante e no triângulo de número um, têm-se nos vértices do triângulo os seguintes conjuntos de vetores 222/111/333/000, 332/221/443/110 e 322/211/433/100, respectivamente.

Basicamente o algoritmo da MV faz com que a tensão média de saída do inversor seja igual ao vetor de referência desejado. Por causa da natureza dessa estratégia de modulação, a MV exige um alto esforço computacional para

execução do algoritmo que, geralmente, limita a operação de chaveamento a alguns Khz [13].

Dentre as abordagens para simplificação do algoritmo, destaca-se o uso de coordenadas não-ortogonais iniciada por [14]. E devido às simplificações presentes no algoritmo desenvolvido por [15], ele foi escolhido para o desenvolvimento deste trabalho.

2.3 Algoritmo da Modulação Vetorial Via Coordenadas Móveis Não-Ortogonais

O algoritmo de modulação vetorial tem a característica de buscar o melhoramento das estratégias de modulação para um inversor multiníveis, com os objetivos de diminuir o número de chaveamentos e melhorar a qualidade harmônica da tensão de saída do inversor.

Esse algoritmo busca uma nova estratégia para a seleção dos três vetores mais próximos e a localização do vetor de referência, utilizando coordenadas móveis não-ortogonais.

Na estratégia desenvolvida por esse algoritmo, a referência não-ortogonal varia de acordo com o setor onde o vetor de referência está localizado. Os TVP são determinados pela identificação do triângulo dentro do hexágono, utilizando a informação do setor e o triângulo onde o vetor de referência está localizado. As razões cíclicas são determinadas por meio de um simples conjunto de equações e informação de onde o vetor de referência está localizado [4].

O desenvolvimento do algoritmo da Modulação Vetorial começa determinando o valor do índice de modulação da tensão de referência a ser sintetizada pelo inversor. Esse procedimento define o tamanho do vetor de referência. Rotacionando esse vetor dentro do circulo trigonométrico, são obtidas as coordenadas Vd e Vq para início dos cálculos de cada padrão de chaveamento do inversor.

Para cada iteração do algoritmo na síntese de um ponto da tensão de saída do inversor, o algoritmo segue os seguintes passos:

a) geração das coordenadas *Vd* e *Vq*.
b) identificação do sextante onde o vetor de referência se encontra.
c) normalização do vetor representados nas coordenadas *Vd* e *Vq* nas coordenadas móveis não ortogonais *Vg* e *Vh*.
d) localização do triângulo dentro do hexágono.
e) cálculo dos valores das razões cíclicas.
f) busca das constantes pelo número do hexágono (TVP).
g) cálculo dos valores de comparação com a onda triangular simétrica, para gerar o sinal de *PWM*.
h) chaveamento do Inversor.

A cada iteração, o algoritmo gera os vetores de referência nas coordenadas Vd e Vq; faz a identificação do sextante em que o vetor de referência está localizado; normaliza o vetor de referência nas coordenadas móveis não-ortogonais Vg e Vh; localiza o triângulo, dentro do hexágono, em que o vetor de referência está localizado; calcula os valores das razões cíclicas tg, th e tgh.

A partir do número do triângulo, o algoritmo faz as buscas das constantes de cada razão cíclicas denominadas Kg, Kh e Kgh e faz os cálculos para encontrar o valor de comparação com a onda triangular, e por fim envia os valores de comparação para o modulador PWM para gerar os sinais de controle das chaves e determinar o tempo de ativação de cada uma delas.

As etapas e o fluxograma do desenvolvimento do algoritmo são apresentados no Capítulo 3, no qual, juntamente com o algoritmo, são mostradas as estratégias utilizadas para a descrição do algoritmo da Modulação Vetorial na linguagem VHDL.

2.4 **Conclusões**

Neste capitulo foram apresentadas as principais características da topologia de inversores multiníveis e do algoritmo da modulação vetorial utilizando coordenadas móveis não-ortogonais, utilizados para o desenvolvimento deste projeto.

A topologia de inversor multiníveis com diodo de grampeamento apresenta como uma de suas principais características a divisão da tensão do elo CC por meio dos seus diodos de grampeamento, e o conteúdo harmônico da tensão de saída do inversor é centrado em duas vezes a freqüência de chaveamento do inversor.

A técnica de modulação vetorial possibilita a otimização na seqüência de chaveamento do inversor, desta forma, o conteúdo harmônico da tensão de saída do inversor é melhorado. A complexidade é uma das características deste algoritmo e a simplificação do algoritmo utilizando coordenadas móveis não ortogonais possibilita a redução da sua complexidade.

3 DESENVOLVIMENTO DO ALGORITMO DA MODULAÇÃO VETORIAL VIA COORDENADAS MÓVEIS NÃO ORTOGONAIS EM VHDL

Neste capítulo são abordadas as etapas para desenvolvimento do algoritmo da Modulação Vetorial para inversores de três níveis e toda a metodologia e as estratégias utilizadas para o desenvolvimento desse algoritmo na linguagem de descrição de *hardware* VHDL.

O algoritmo da Modulação Vetorial também foi desenvolvido no programa MatLab® com a finalidade de fornecer dados de simulação para serem comparados com os valores da simulação do algoritmo descrito em VHDL.

3.1 Estratégia desenvolvida para cálculo na linguagem de descrição de *hardware* VHDL.

O algoritmo da MV envolve cálculos numéricos e lógicos. A representação desses números para cálculo em notação ponto-fixo é mais simples e exige menos esforço computacional em comparação com a notação ponto flutuante. Assim, a notação ponto fixo foi escolhida para o desenvolvimento dos cálculos neste projeto.

A metodologia desenvolvida para fazer multiplicação, soma e subtração dos números em ponto fixo é muito simples. Fazendo uma analogia com as operações matemáticas com números decimais em ponto fixo na base decimal é possível aplicá-las em operação com números decimais na base binária.

Para o desenvolvimento desses cálculos, os números foram representados como vetores de *bits* com sinal, declarando-os como vetores do tipo *signed*. Definindo a quantidade de *bits* para representar a parte decimal e a parte inteira do número, o valor de resposta é facilmente identificado dentro do vetor de saída.

Na operação de multiplicação entre números em ponto fixo, o principio básico utilizado para determinar o valor da resposta consiste em identificar onde está localizado o ponto que separa a parte inteira da parte fracionária do número após a multiplicação das parcelas. Isso é feito somando a quantidade de *bits* que representam a parte fracionária de cada número, em cada parcela da

multiplicação e contando da direita para a esquerda no vetor resposta, a quantidade de *bits* que irão representar a parte fracionária, o restante dos *bits*, representa a parte inteira do valor de resposta.

Com essa metodologia foi possível fazer os cálculos em ponto fixo na formatação adequada a cada tipo de operação, obtendo uma melhor precisão nos cálculos do algoritmo da Modulação Vetorial.

As operações de soma e subtração entre números em ponto fixo na linguagem VHDL utilizando vetores de *bit*, devem ser feitas da mesma maneira que se faz a soma e a subtração de números decimais na base decimal, portanto, é necessário que as parcelas da soma ou da subtração estejam na mesma formatação, ou seja, ambas as parcelas devem conter o mesmo número de *bits* para representação da parte inteira e também a mesma quantidade de *bits* para representar a parte fracionária dos números.

Caso as parcelas da operação estejam em diferentes formatações é preciso fazer um deslocamento em uma das parcelas até que as duas fiquem com a mesma formatação.

No Anexo G encontra-se uma breve descrição de como foi feita a multiplicação de dois números utilizando a metodologia descrita nos parágrafos acima.

3.2 **Fluxograma para desenvolvimento do algoritmo da Modulação Vetorial na linguagem VHDL**

A Figura 3.1 apresenta o fluxograma para o desenvolvimento do algoritmo da Modulação Vetorial na linguagem de descrição de *hardware* VHDL. Ele é composto pelas seguintes etapas: 1) definição do índice de modulação; 2) geração do vetor de referência (Vd,Vq); 3) identificação do sextante onde o vetor de referência está localizado; 4) normalização do vetor de referência nas coordenadas móveis não-ortogonais (Vg e Vh); 5) identificação do triângulo em que o vetor de referência se encontra; 6) cálculo das razões cíclicas (tg,th e tgh); 7) cálculo dos valores de comparação com a onda triangular para cada par de chaves complementares do inversor; 8) geração dos sinais PWM para todas as chaves do inversor.

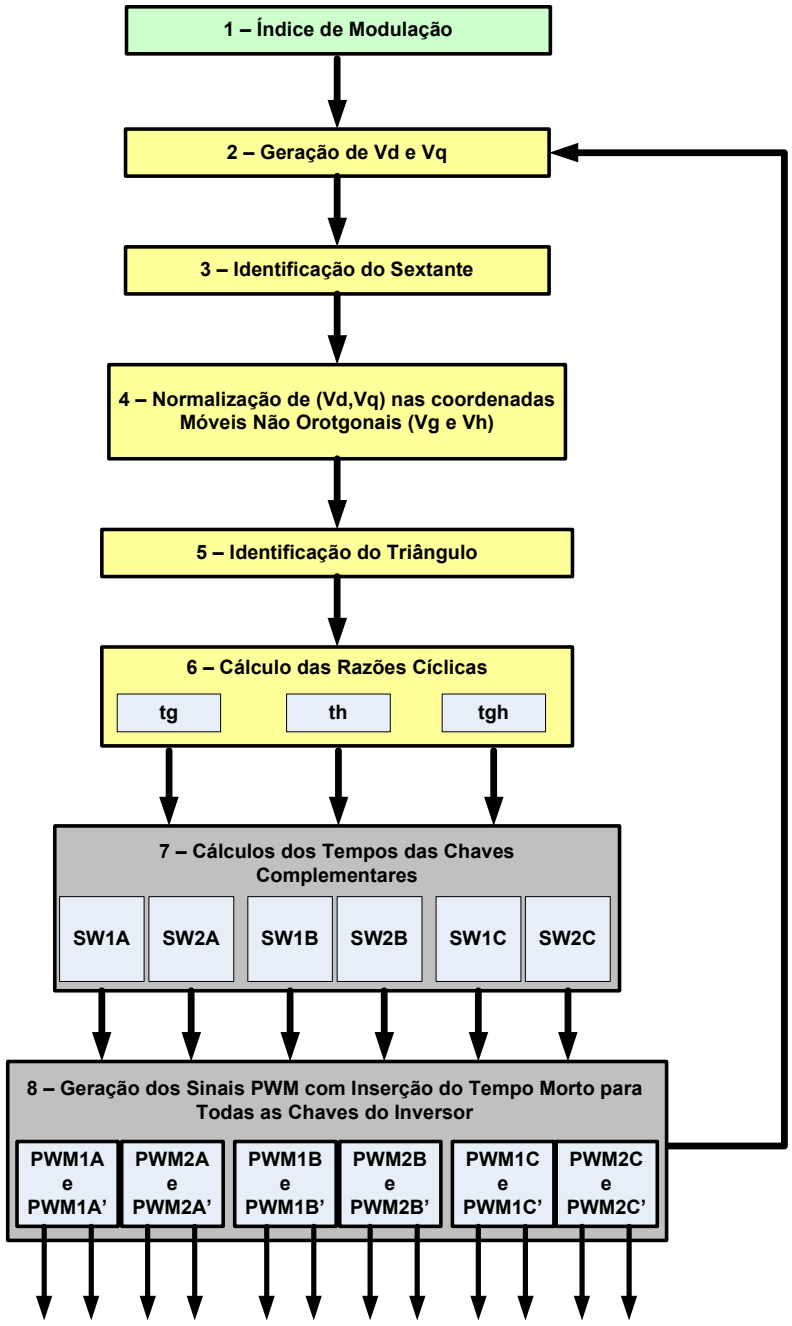


Figura 3.1– Fluxograma para desenvolvimento do algoritmo da modulação vetorial em VHDL.

O primeiro passo para o desenvolvimento do algoritmo da Modulação Vetorial é definir o índice de modulação (m), ele é definido pela razão entre a magnitude do vetor de referência (V*) e o valor de pico da componente fundamental da tensão de onda quadrada ($V_{1SW} = \left(\frac{2V_{CC}}{\pi}\right)$), correspondente ao valor máximo do elo *CC*, dado pela expressão (3.1).

$$m = \frac{V^*}{V_{1SW}} \tag{3.1}$$

As regiões de operação linear e de saturação do inversor são determinadas pelo valor do índice de modulação: com m entre (0<m<0.907), o inversor trabalha na região linear e, com m entre (0.907<m<1), o inversor trabalha na região de saturação. Relacionando o índice de modulação ao vetor de referência e utilizando o sistema por unidade (pu), tem-se que, para m=1, o valor de V* é aproximadamente 0,636 e para m=0 o valor de V* é 0. Neste trabalho é abordado apenas o desenvolvimento para a região linear de operação do inversor.

Definido o índice de modulação, o algoritmo gera um vetor de referência (Vd,Vq) correspondente a cada ponto de amostragem da tensão de referência. O algoritmo segue para o próximo passo identificando o sextante, em que se encontra o vetor de referência. Com a identificação do sextante o algoritmo da MV faz a transformação do vetor de referência das coordenadas Vd e Vq para as coordenadas móveis não-ortogonais Vg e Vh.

Após a normalização, o próximo passo do algoritmo é fazer a identificação do triângulo dentro do hexágono (Lh) onde o vetor de referência está localizado. As razões cíclicas tg, th e tgh são determinadas utilizando o vetor de referência normalizado nas coordenadas móveis não-ortogonais.

Com a determinação das razões cíclicas, o próximo passo do algoritmo é determinar o valor do tempo de chaveamento das chaves do inversor e enviar esses valores para o modulador, onde os sinais PWM de cada uma das chaves do inversor são gerados. O número do triângulo indexa as constantes denominadas pelo algoritmo de *Kg*, *Kh* e *Kgh,* pré-calculadas e armazenadas em vetores que, em conjunto com as razões cíclicas, são utilizados para determinar o tempo de chaveamento de cada uma das chaves do inversor.

Em um inversor de três níveis, os doze sinais de controle das chaves do inversor são gerados aos pares, uma vez que os pares de chaves (S1x, S1x') e (S2x, S2x') são complementares, onde x corresponde às fases A, B e C do inversor.

No fluxograma apresentado na Figura 3.1, as etapas de 2 a 6, correspondentes aos cálculos do algoritmo da MV, devem ser executadas sequencialmente; a etapa 7 é responsável pelo cálculo do tempo de chaveamento de cada par de chaves complementares do inversor, e pode ser desenvolvida em paralelo ou sequencialmente. No desenvolvimento dessa etapa do projeto, optou-se pelo desenvolvimento em paralelo, aproveitando a característica de síntese de circuitos em paralelo do FPGA. A etapa 8, onde são gerados todos os sinais PWM para controle das chaves do inversor, deve ser implementada em paralelo.

O desenvolvimento de etapas em paralelo do algoritmo auxilia na redução do seu tempo de cálculo, uma vez que, em implementações para inversores multiníveis de ordem maior que três níveis, o número de chaves aumenta consideravelmente de doze para vinte e quatro chaves em uma implementação para inversores de cinco níveis, por exemplo.

A estrutura desenvolvida no FPGA para os cálculos do algoritmo e para a modulação é apresentada na Figura 3.2. Nota-se que, a partir da etapa 6, o processamento do algoritmo apresenta diferentes blocos que são indepentes, que podem ser sintetizados paralelo.

A seguir serão descritas todas as etapas apresentadas no fluxograma da Figura 3.1 para o desenvolvimento do algoritmo da modulação vetorial na linguagem de descrição de hardware VHDL.

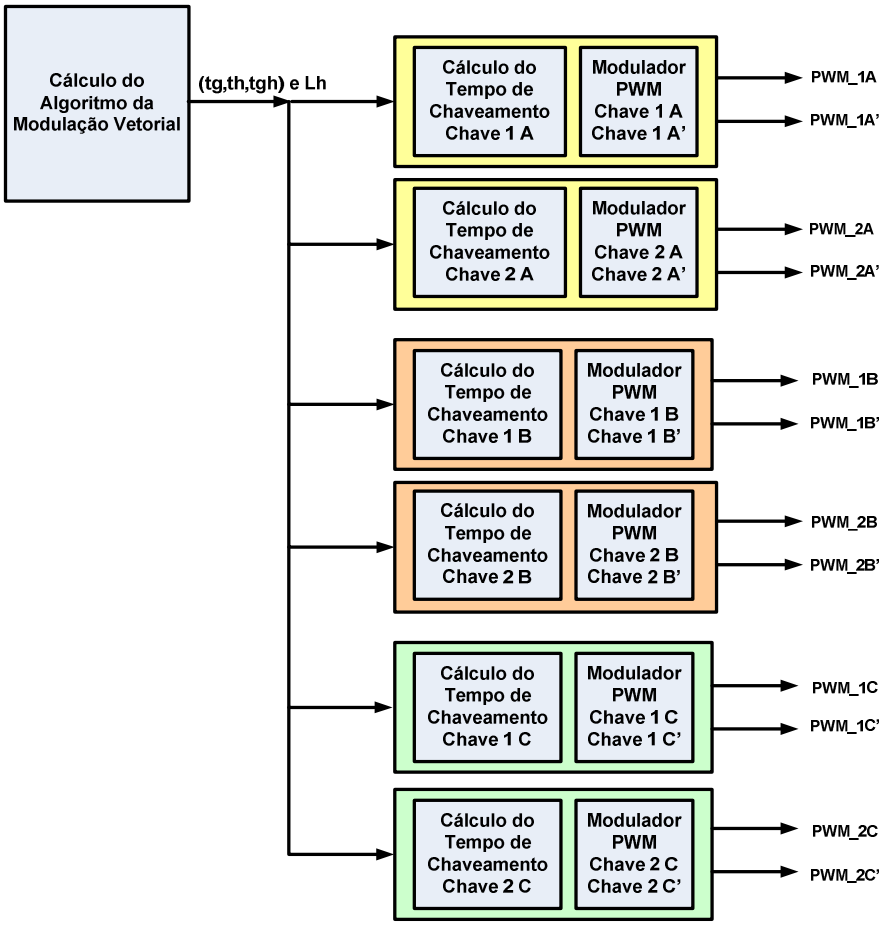


Figura 3.2 – Estrutura do algoritmo da modulação vetorial desenvolvida em FPGA.

3.2.1 Geração do vetor de referência (Vd, Vq)

Para o desenvolvimento do algoritmo da MV existe a necessidade da geração de uma tensão senoidal de referência, a qual será sintetizada pelo inversor. Essa tensão de referência deve ser representada no plano d-q. Para a composição senoidal do vetor de referência deve-se gerar uma amostra de ângulos a partir de um vetor de módulo fixo girante no plano d-q. A referência senoidal é composta calculando o seno e o co-seno dessas amostras.

A decomposição do vetor de referência nas coordenadas Vd e Vq é feita pelas seguintes expressões:

$$V_d = V^* \cos(\theta)$$
$$V_q = V^* \sin(\theta) \tag{3.2}$$

Para o desenvolvimento das expressões apresentadas em (3.2) foi preciso buscar uma alternativa para o cálculo do seno e co-seno de (θ), uma vez que, na linguagem VHDL, não existe uma instrução que faça esse cálculo.

A partir de um $\Delta\theta$ variando de 0° a 90°, calcula-se uma tabela de consulta com o seno para cada variação do ângulo. Pela identificação do quadrante, o seno e o co-seno são calculados para a variação do ângulo de 0° a 360°. A tabela de consulta é composta por 451 pontos e uma estratégia de interpolação é utilizada para aumentar a precisão dos cálculos. O algoritmo utilizado encontra-se no Anexo I.

Na Figura 3.3 são apresentados os valores do vetor de referência Vd e Vq obtidos para um índice de modulação 0,70.

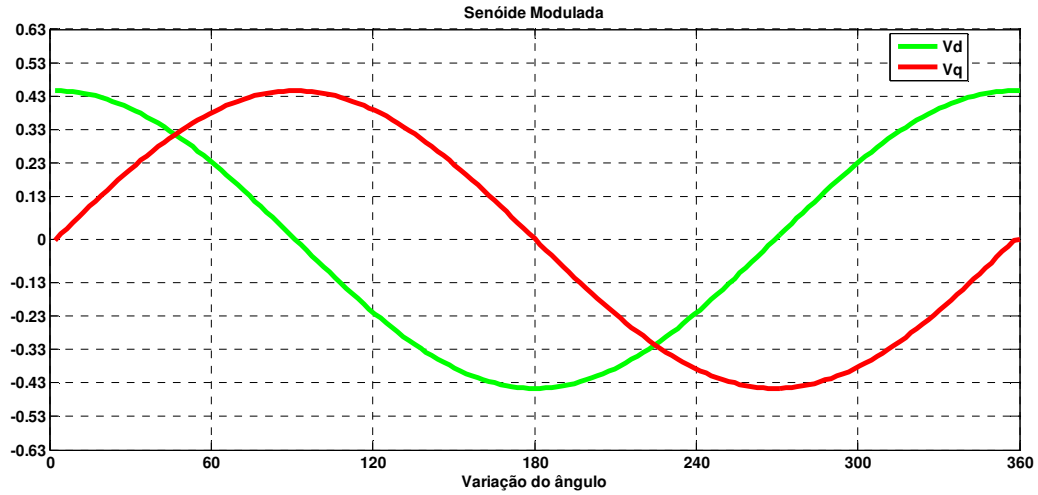


Figura 3.3 – Componentes Vd e Vq para índice de modulação 0,70.

A Figura 3.4 a seguir mostra o gráfico de Vd por Vq para um índice de modulação 0,70.

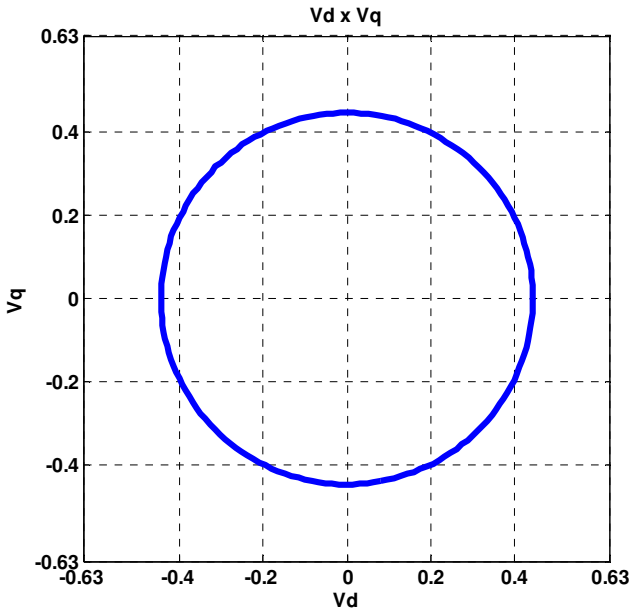


Figura 3.4 - Gráfico de Vd por Vq com índice de modulação 0,70.

3.2.2 Identificação do sextante

A identificação do sextante onde V* se encontra é utilizada para definir os coeficientes da conversão não-ortogonal e também para a síntese do padrão de chaveamento. A identificação do sextante é feita usando o conceito de semiplanos positivos e negativos [13] onde são utilizadas três equações de reta (3.3). A equação (3.4) calcula o número do sextante N_s e, a partir da Tabela 3.1, o valor do sextante N_s é organizado em forma crescente.

$$A = Sign(V_q)$$

$$B = Sign(\sqrt{3}V_d - V_q) \qquad (3.3)$$

$$C = Sign(-\sqrt{3}V_d - V_q)$$

Nas expressões (3.3) a função *Sign* retorna o sinal do valor obtido com a realização da expressão.

Em VHDL, a função *Sign* é implementada identificando o *bit* de sinal, ou seja, o *bit* mais significativo do resultado de cada uma das expressões apresentadas em 3.2.

A variável que fará a identificação do sextante é chamada de N_s, onde, a partir dos valores de A, B e C, é calculada a expressão (3.4) a seguir:

$$N_S = A + 2B + 4C \tag{3.4}$$

A tabela 3.1 relaciona o sextante com a variável N_s. Essa relação é utilizada para colocar o valor do sextante na forma sequencial de 1 até 6.

Tabela 3.1 – Relação entre o sextante e N_S

Ns	3	1	5	4	6	2
Sextante	1	2	3	4	5	6

A Figura 3.5 apresenta o trecho de código em VHDL para encontrar o valor da variável Ns e como foi implementada a relação dessa variável com o valor do sextante.

```
------ A ------
Ns:= “0000”
sigA := Vq_16(15) xor '1';
Ns(0) := sigA; -- Corresponde ao valor de A

------ B ------
temp_calc  :=  Vq_sft - prod_q3_12;
sigB := temp_calc(15);
Ns(1) := sigB; -- Corresponde ao valor de 2B

------ C -------
temp_calc  :=  Vq_sft + prod_q3_12;
sigC := temp_calc(15);
Ns(2) := sigC; -- Corresponde ao valor de 4C

------ Ns -------
Ns_int := CONV_INTEGER(Ns);
Sext := vetsext(Ns_int);
Aux_Sext := CONV_STD_LOGIC_VECTOR(Sext,3);
```

Figura 3.5 – Código em VHDL para encontrar o valor da variável Ns.

A Figura 3.6 apresenta o sextante onde o vetor de referência se encontra para um índice de modulação 0,70 para um ciclo da frequência fundamental.

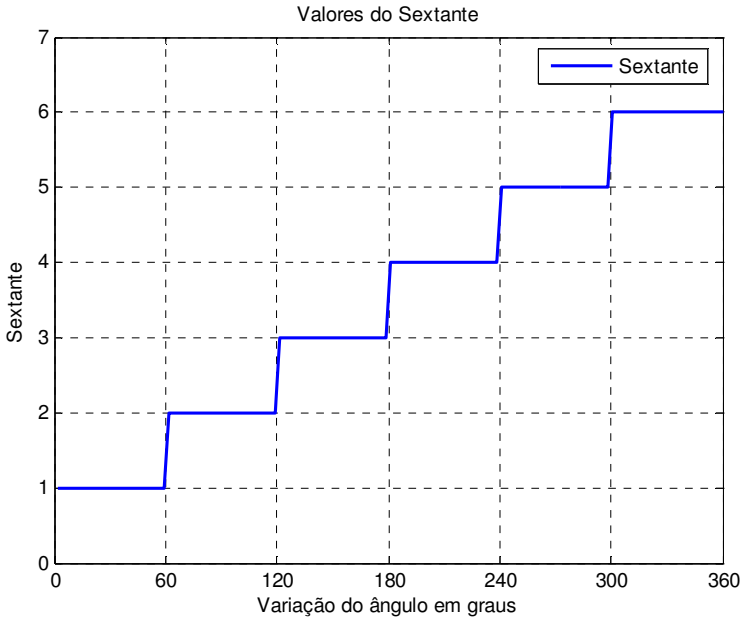


Figura 3.6 – Localização do vetor de referência no sextante para índice de modulação 0,70.

3.2.3 Normalização do vetor de referência (Vg, Vh)

A simplificação proposta por esse algoritmo consiste na utilização de coordenadas móveis não-ortogonais normalizadas (V_g e V_h) de acordo com o passo de tensão do inversor e de acordo com o sextante em que o vetor de referência está localizado. O processo de normalização é bastante simples.

Primeiro, identifica-se em qual sextante está localizado o vetor de referência, em seguida, aplica-se uma relação de transformação de coordenadas de acordo com o sextante, conforme a equação (3.5).

$$\begin{bmatrix} V_g \\ V_h \end{bmatrix} = \frac{1}{V_{dc}} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} V_d \\ V_q \end{bmatrix} \tag{3.5}$$

Como o plano *dq* é dividido em seis sextantes, a tabela Aij possui seis elementos que são indexadas de acordo com o número do sextante. A tabela Aij, com todos os elementos de normalização para um inversor de três níveis, é apresentada em (3.6).

$$
\begin{aligned}
A_{11} &= [\quad 3 \quad\quad 3 \quad\quad 0 \quad\quad -3 \quad\quad -3 \quad\quad 0] \\
A_{12} &= [\ -\sqrt{3} \quad\quad \sqrt{3} \quad\quad 2\sqrt{3} \quad\quad \sqrt{3} \quad\quad -\sqrt{3} \quad\quad -2\sqrt{3}] \\
A_{21} &= [\quad 0 \quad\quad -3 \quad\quad -3 \quad\quad 0 \quad\quad 3 \quad\quad 3\ ] \\
A_{22} &= [\ 2\sqrt{3} \quad\quad \sqrt{3} \quad\quad -\sqrt{3} \quad\quad -2\sqrt{3} \quad\quad -\sqrt{3} \quad\quad \sqrt{3}\ ]
\end{aligned}
\tag{3.6}
$$

Onde A_{11} é a matriz de coeficientes a_{11}, A_{12} é a matriz de coeficientes a_{12}, A_{21} é a matriz de coeficientes a_{21} e A_{22} é a matriz de coeficientes a_{22}.

A Figura 3.7 apresenta os valores da coordenada Vg normalizada.

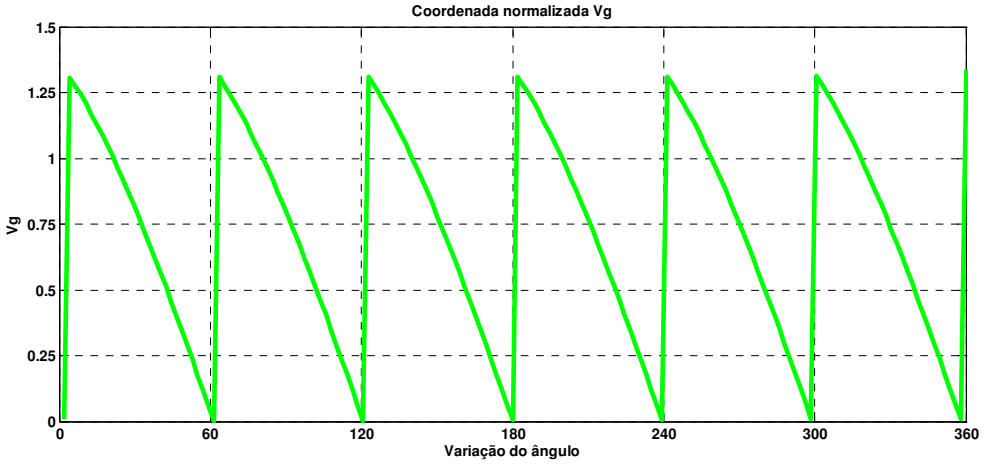


Figura 3.7 – Componente Vg normalizada para índice de modulação 0,70

A figura 3.8 apresenta os valores da coordenada Vh normalizada.

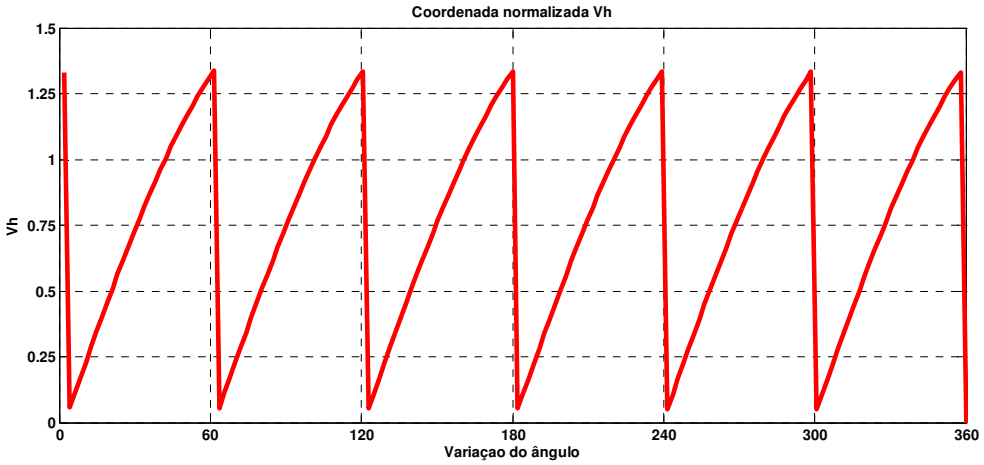


Figura 3.8 – Componente Vh normalizada para índice de modulação 0,70

O código completo, em VHDL, para a geração do vetor de referência e sua normalização encontra no Anexo C.

3.2.4 Identificação do triângulo e cálculo das razões cíclicas

A identificação do triângulo dentro do hexágono define os vetores de chaveamento que serão sintetizados pelo inversor. O algoritmo define o modo de operação do inversor M_d (3.7) que, basicamente, opera da seguinte forma: $M_d = 0$, o inversor trabalha com dois níveis; $M_d = 1$, o inversor trabalha com três níveis, e assim por diante. Com a definição de M_d em conjunto com parte inteira das coordenadas V_g e V_h (3.8) observadas na Figura 3.9 define-se a localização do triângulo dentro do sextante (3.10).

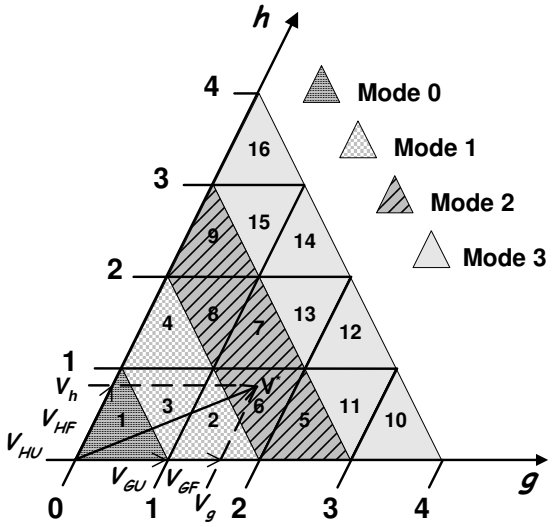


Figura 3.9 - Modo de operação do inversor (M_d) e parte inteira e fracionária de Vg e Vh.

O cálculo das razões cíclicas inicia-se obtendo o modo de operação do inversor dado pela seguinte expressão:

$$Md = floor\left(Vg + Vh\right) \tag{3.7}$$

A expressão *floor* retorna a parte inteira da soma de Vg e Vh. Em VHDL, essa função é implementada utilizando a instrução *downto,* selecionando os *bits* que correspondem à parte inteira da variável de soma das duas parcelas Vg e Vh e montando o número com apenas a parte inteira. A parte fracionária é completada com zeros.

Os cálculos seguem, separando a parte inteira e a parte fracionária de Vg e Vh, como mostram as expressões (3.7) e (3.8).

A equação (3.8) obtém, nas variáveis Vgu e Vhu, a parte inteira de Vg e Vh.

$$\begin{aligned} V_{GU} &= floor\,(V_g) \\ V_{HU} &= floor\,(V_h) \end{aligned} \tag{3.8}$$

A equação (3.9) separa a parte inteira da parte fracionária de Vg e Vh.

$$\begin{aligned} V_{GF} &= V_g - V_{GU} \\ V_{HF} &= V_h - V_{HU} \end{aligned} \tag{3.9}$$

De forma similar, as partes fracinárias Vgf e Vhf também são obtidas utilizando a instrução *downto* e selecionando a parte fracionária de Vg e Vh.

A Figura 3.10 apresenta o código em VHDL para determinar a parte fracionária e inteira das variáveis.

```
Vhf := "0000000000000000";  -- Q2.14
Vhf (13 downto  0) := vh_16(13 downto  0); -- Q2.14
Vhu := vh_16(15 downto 14); -- Q2.14
Vh_int := CONV_INTEGER(Vhu);
Vgf := "0000000000000000";  -- Q2.14
Vgf (13 downto  0) := vg_16(13 downto  0); -- Q2.14
Vgu := Vg_16(15 downto 14); --Q2.14
Vg_int := CONV_INTEGER(Vgu);
```

Figura 3.10 – Código em VHDL para determinar a parte inteira e fracionária de Vg e Vh.

A partir do cálculo do modo de operação do inversor, por meio da equação (3.6), e da separação da parte inteira da parte fracionária em (3.8) e (3.9), a localização do triângulo no sextante em que o vetor de referência esta localizado é calculada pela da expressão (3.10) apresentada a seguir:

$$L_S = Md^2 + Md + 1 + V_{HU} - V_{GU} \tag{3.10}$$

A numeração dos triângulos feita em [15] consiste em numerá-los no diagrama vetor de espaço, Figura 2.6, do centro para a borda do hexágono em

sentido anti-horário. Essa estratégia de numeração aproveita a numeração dos triângulos no diagrama vetor de espaço para extensão da topologia do inversor. A numeração dos triângulos para um inversor de três níveis pode ser observada na Figura 3.14.

Com a normalização dos valores de Vd e Vq nas coordenadas móveis não-ortogonais *Vg* e *Vh*, o cálculo das razões cíclicas torna-se muito simples, pois ele independe do sextante em que se encontra o vetor de referência. Portanto, o universo para o cálculo das razões cíclicas é reduzido para um sexto.

O número do triângulo dentro do hexágono é obtido usando o número do sextante onde o vetor de referência se encontra (Sext) e a localização do triângulo dentro do sextante (Ls). Essa relação é dada pela expressão a seguir [16]:

$$L_H = (Sext - 1).4 + L_S \tag{3.11}$$

A Figura 3.11 apresenta a localização do triângulo dentro do hexágono em que o vetor de referência se encontra para o índice de modulação 0,70 para um ciclo da frequência fundamental.

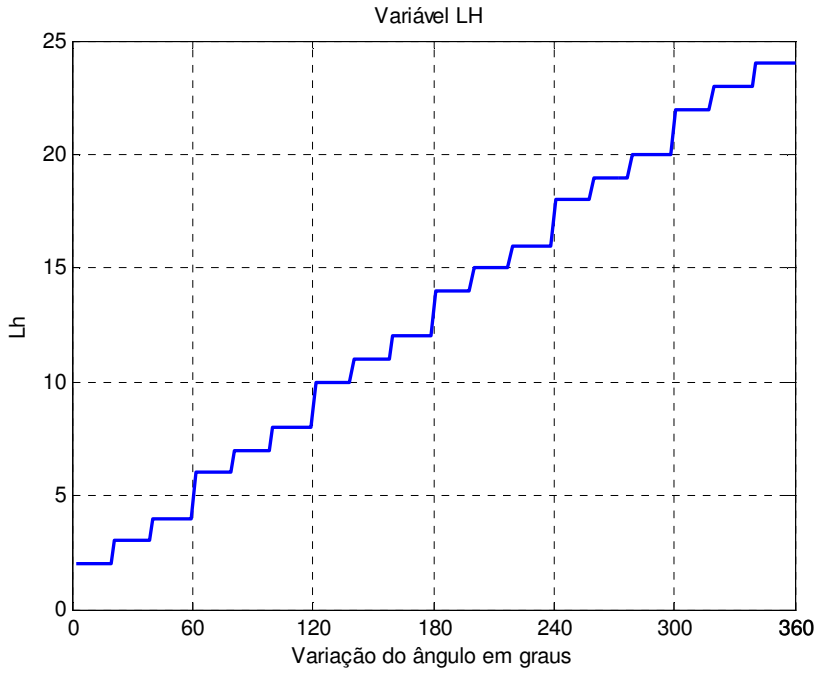


Figura 3.11 – Localização do vetor de referência dentro do hexágono para índice de modulação 0,70

A equação (3.11) fornece o número do triângulo no hexágono a partir do número do triângulo referente ao primeiro sextante mais certo *off-set* de acordo

com o número do sextante [16], em um inversor de três níveis para os sextantes 1, 2, 3, 4, 5 e 6 os *off-sets* são 0, 4, 8, 12, 16 e 20 respectivamente.

Na Figura 3.12 é apresentada uma parte do código em VHDL, responsável por encontrar os valores das variáveis Ls e Lh.

```
vgmaisvh := vg_16 + vh_16 ; --Q2.14 + Q2.14 = Q2.14
md :="000";
md(1 downto 0) := vgmaisvh(15 downto 14); --Q2.14
md_int := CONV_INTEGER (md);
Ls := (md_int*md_int) + md_int + umb + Vh_int - Vg_int;
sext_int := CONV_INTEGER (sext);
Lh := (sext_int - umb) * quatro + Ls;
```

Figura 3.12 – Código em VHDL para encontrar os valores das variáveis Ls e Lh.

E por fim, as razões cíclicas são obtidas desenvolvendo as equações dadas em (3.12).

$$\begin{aligned} t_g &= \left|triang\_type - V_{GF}\right| * T_S \\ t_h &= \left|triang\_type - V_{HF}\right| * T_S \\ t_{gh} &= \left(1 - t_g - t_h\right) * T_S \end{aligned} \tag{3.12}$$

Sendo que o tipo triângulo (*triang_type*) é obtido pelas relações a seguir [15]:

Se (Ls + Md) é impar : faz-se triang_type = 0
Se (Ls + Md) é par : faz-se triang_type = 1

Para definir se a operação (Ls + Md) resultou em um número ímpar ou par na linguagem VHDL, basta identificar o *bit* menos significativo do valor de resposta, caso seja 0, o número é par, caso seja 1, a operação resultou em um número ímpar.

A determinação do módulo da operação para cálculo das razões cíclicas é feita identificando o *bit* de sinal do resultado da operação, se for negativo, ou seja, se o bit de sinal for 1, deve-se fazer o complemento do número de resposta para que esse fique com seu valor positivo.

A Figura 3.13 apresenta o código em VHDL para o cálculo das razões cíclicas tg, th e tgh.

```
tipo     := Ls + md_int;
tipo_vec := CONV_SIGNED (tipo,16);
triang_type := "0000000000000000";
triang_type(14) := not(tipo_vec(0));
--------------Variável Tg----------------------------
var_tg := ( triang_type - Vgf ); -- Q2.14 - Q2.14
if (var_tg(15) = '1') then
    var_tg := -var_tg;
end if;
---------------Variável Th----------------------------
var_th := ( triang_type - Vhf ); -- Q3.13 - Q3.13
if (var_th(15) = '1') then
    var_th := -var_th;
end if;
----------------Variável Tgh-------------------------
var_tgh := umQ313 - var_tg - var_th; -- Q13.3
```

Figura 3.13 – Código em VHDL para cálculo das razões cíclicas.

3.2.5 Síntese do padrão de chaveamento para inversores multiníveis

A síntese do padrão de chaveamento de cada uma das chaves de um inversor multiníveis consiste nos cálculos das razões cíclicas e na determinação das suas constantes para cada um dos triângulos do hexágono.

Em cada triângulo devem ser determinados os três vetores mais próximos. A estratégia adotada no algoritmo para síntese dos padrões de chaveamento utiliza todos os vetores redundantes disponíveis para cada triângulo. Essa estratégia é chamada de padrão de chaveamento completo que assegura o número mínimo de chaveamento, um ótimo conteúdo harmônico da tensão de saída do inversor e o melhor balanceamento dos capacitores do elo *CC* [16].

Para definir o valor das constantes para cada razão cíclica de cada um dos triângulos do hexágono é preciso montar a sequência de vetores a ser sintetizada pelo algoritmo. Nos triângulos localizados nos sextantes ímpares do

hexágono, a síntese deve começar do nível mais baixo para o nível mais alto de tensão. Nos sextantes pares, a síntese deve começar do nível de tensão mais alto para o nível mais baixo.

Na síntese do padrão de chaveamento para um sextante ímpar, utiliza-se como exemplo o triângulo número 1 apresentado na Figura 3.14 em destaque, com seus respectivos vetores de chaveamento.

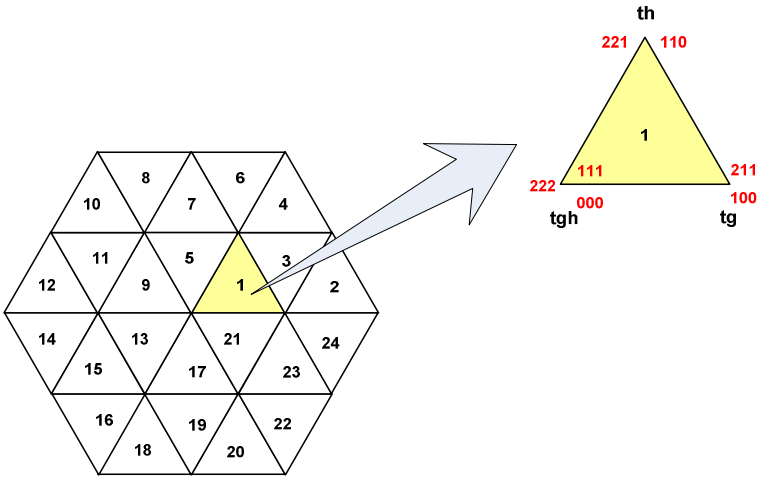


Figura 3.14 – Triângulo 1 e seus vetores de chaveamento.

O triângulo 1 está localizado no sextante ímpar, portanto, a síntese deve começar do nível de tensão mais baixo para o nível mais alto. A Figura 3.15 apresenta a sequência dos vetores e o estado das chaves 1 e 2 para cada nível de tensão a ser sintetizado pelo inversor.

Fase	Sequência direta							Sequência reversa						
A	0	1	1	1	2	2	2	2	2	2	1	1	1	0
B	0	0	1	1	1	2	2	2	2	1	1	1	0	0
C	0	0	0	1	1	1	2	2	1	1	1	0	0	0
Duty-cycle	**tgh/8**	**tg/4**	**th/4**	**tgh/4**	**tg/4**	**th/4**	**tgh/8**	**tgh/8**	**tg/4**	**th/4**	**tgh/4**	**tg/4**	**th/4**	**tgh/8**
Chave 1A	*off*	*off*	*off*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*	*off*	*off*
Chave 2A	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*

Figura 3.15 – Sequência de chaveamento das chaves 1 e 2 da fase A para o triângulo 1 (Sextante ímpar).

Para determinar os coeficientes das razões cíclicas para um sextante ímpar é preciso fazer a soma dos coefientes das razões cíclicas que estão compreendidos até a transição de *off* para *on* na sequência de estados de cada uma das chaves. Os coeficientes da chave 1A para o triângulo de número 1 são: Ktg1A = 1/4, Kth1A = 1/4 e o coeficiente de Ktgh1A dado por (1/8 + 1/4 = 3/8). Os

coeficientes para a chave 2 da fase A são: Ktg2A = 0, Kth2A = 0 e o coeficiente de Ktgh1A = 1/8.

Na síntese do padrão de chaveamento para um sextante par, utiliza-se como exemplo o triângulo de número 5 apresentado na Figura 3.16 em destaque, com seus respectivos vetores de chaveamento.

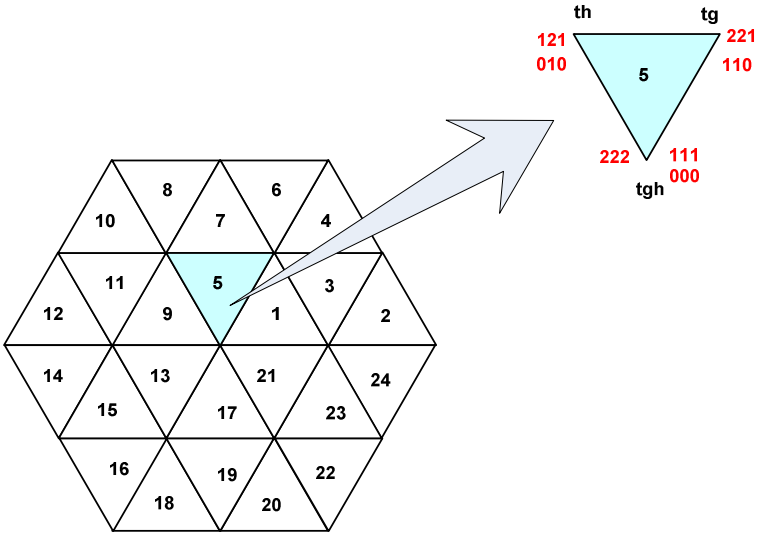


Figura 3.16 – Triângulo 5 e seus vetores de chaveamento

O triângulo 5 está localizado em um sextante par, portanto, a síntese deve começar no maior nível de tensão para o menor nível. A Figura 3.17 apresenta a sequência dos vetores e o estado das chaves 1 e 2 para cada nível de tensão a ser sintetizado pelo inversor.

Fase	Sequência direta							Sequência reversa						
A	2	2	1	1	1	0	0	0	0	1	1	1	2	2
B	2	2	2	1	1	1	0	0	1	1	1	2	2	2
C	2	1	1	1	0	0	0	0	0	0	1	1	1	2
Duty-cycle	**tgh/8**	**tg/4**	**th/4**	**tgh/4**	**tg/4**	**th/4**	**tgh/8**	**tgh/8**	**tg/4**	**th/4**	**tgh/4**	**tg/4**	**th/4**	**tgh/8**
Chave 2A	*on*	*on*	*on*	*on*	*on*	*off*	*off*	*off*	*off*	*on*	*on*	*on*	*on*	*on*
Chave 1A	*on*	*on*	*off*	*off*	*off*	*off*	*off*	*off*	*off*	*off*	*off*	*off*	*on*	*on*

Figura 3.17 – Sequência de chaveamento das chaves 1 e 2 da fase A para o triângulo 5 (Sextante par).

Para determinar os coeficientes das razões cíclicas para um sextante par, deve-se fazer a soma dos coefientes das razões cíclicas que estão compreendidos até a transição de *on* para *off* na sequência de estados de cada uma das chaves. Os coeficientes da chave 1A para o triângulo de número 5 são:

Ktg1A = 1/4, Kth1A = 0 e Ktgh1A = 1/8. Os coeficientes para a chave 2A do triângulo 5 são: Ktg2A = (1/4 + 1/4 = 1/2), Kth2A = 1/4 e Ktgh2A = (1/8 + 1/8 = 1/4).

Os coeficientes das razões cíclicas de cada uma das chaves devem ser armazenados em uma tabela de consulta e referenciados pelo número do triângulo. Essa etapa do algoritmo compreende a localização dos TVP e o cálculo das constantes Ktg, Kth e Ktgh para cada uma das chaves do inversor.

O modulador PWM define o tempo em que cada chave deve permanecer ativada. O contador de cada modulador PWM define o período de chaveamento das chaves e o valor do comparador define o tempo de ativação de cada uma delas. Nota-se que há diferença na ativação das chaves para um sextante ímpar e para um sextante par. Nos sextantes ímpares (1, 3 e 5), as chaves são ativadas em nível lógico 1 (um) depois da comparação e nos sextantes pares (2, 4 e 6), as chaves são ativadas em nível lógico 1 (um), antes e depois da comparação. A ilustração da ativação é apresentada na Figura 3.18.

Para sintetizar esses dois padrões seriam necessários dois tipos de modulador, um chamado “Ativo_Alto” para gerar os sinais PWM para os sextantes ímpar e outro “Ativo_Baixo” para sintetizar os sinais PWM para os sextantes par. Utiliza-se no entanto uma simplificação, os sinais PWM são gerados como se houvesse apenas o padrão de chaveamento “Ativo_Alto” e, então, o algoritmo utiliza uma variável para identificar o tipo de sextante. Essa variável recebe 0 (zero) se o sextante for par e recebe 1 (um) se o sextante for ímpar. Se o sextante for ímpar, é mantido o sinal PWM, caso contrário, o valor do PWM é invertido. A representação do modulador “Ativo_Alto” e “Ativo_Baixo” é apresentada na Figura 3.18.

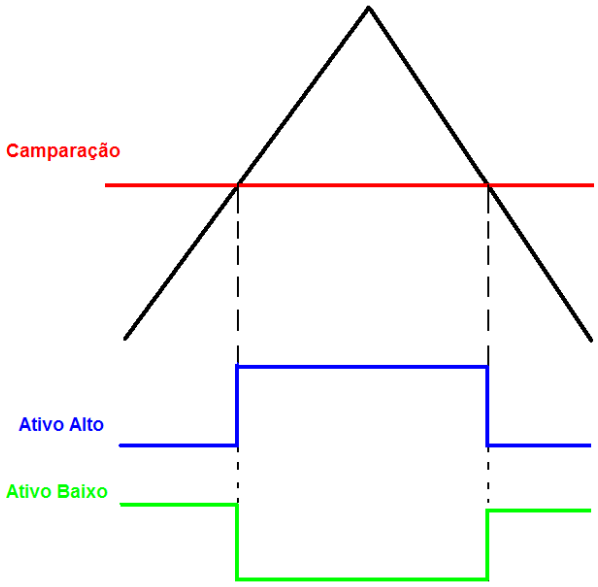


Figura 3.18 – Ilustração do modulador Ativo_Alto e Ativo_Baixo

3.2.6 Cálculos dos valores de comparação para geração dos sinais PWM

O cálculo do tempo de chaveamento é dado pela soma de produtos entre as razões cíclicas *tg*, *th* e *tgh* e seus respectivos coeficientes *Kg*, *Kh* e *Kgh*, pré-calculados, armazenados em vetores e referenciados pelo número do triângulo em que o vetor de referência se encontra dentro do hexágono (Lh).

A expressão utilizada para se encontrarem os valores dos comparadores, que serão comparados com a onda triangular simétrica, é a seguinte:

$$T_{SOxA} = K_{gxA}(Lh)tg + K_{hxA}(Lh)th + K_{ghxA}(Lh)tgh \qquad (3.12)$$

Em VHDL, um vetor para cada constante Kg, Kh e Kgh, de cada chave de todas as fases, foi declarado para o desenvolvimento do algoritmo, dessa maneira, os cálculos podem ser feitos de maneira mais eficiente e mais rápida.

Por meio da expressão para o cálculo dos valores do comparador são obtidos os valores a serem comparados com a onda triangular simétrica e gerar os sinais de PWM para controlar as chaves do inversor.

O sinal de PWM é gerado ao se comparar o valor do comparador obtido pela expressão (3.12) com uma onda triangular simétrica. A geração do sinal de PWM pode ser observada na Figura 3.19:

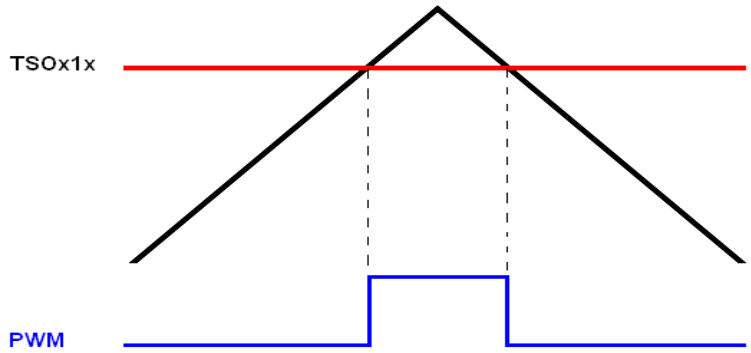


Figura 3.19 – Modulação da largura de pulso do sinal de controle das chaves do inversor

3.2.7 Geração do sinal de MLP com tempo morto

Essa etapa do algoritmo é composta por três processos diferentes que atuam tanto no desenvolvimento do algoritmo quanto no sincronismo da geração do sinal PWM. É nessa etapa que é gerada a onda triangular simétrica por meio de um contador crescente/decrescente para ser comparado com o valor do comparador, gerando o sinal de PWM que fará o controle das chaves do inversor.

Nessa etapa, também estão descritos os processos de inserção de tempo morto nas chaves complementares do inversor e o processo de sincronismo entre a geração do vetor de referência e a geração do sinal PWM. A seguir, serão discutidos detalhadamente esses três processos essenciais para o desenvolvimento do algoritmo.

3.2.7.1 Processo de geração do sinal PWM

O sinal PWM é gerado em um processo distinto, em que o valor do comparador é comparado com uma onda triangular simétrica, essa onda triangular é descrita na linguagem VHDL através de um contador do tipo crescente/decrescente. O valor do comparador é comparado com a onda triangular, gerando o sinal de PWM que fará o controle das chaves do inversor, como se demonstra na Figura 3.19.

O sinal de PWM é gerado da seguinte maneira: se o valor do comparador for menor que o valor do contador, onda triangular, então o sinal de PWM recebe 0 (zero) caso contrário, se o valor do comparador for maior que o valor do

contador então o sinal de PWM recebe 1 (um). Dessa maneira é formado o sinal de PWM modulado de acordo com o vetor de referência calculado pelo algoritmo.

3.2.7.2 Processo de geração do tempo morto

Nos pares de chaves complementares do inversor é possível que haja a comutação simultânea dessas chaves e isso pode acarretar em um curto-circuito danificando as chaves do inversor. Para que isso não ocorra, é preciso que seja inserido um tempo morto entre os sinais PWM das chaves complementares, ou seja, ambas as chaves são desligadas.

A Figura 3.20 ilustra a inserção do tempo morto em um sinal de PWM.

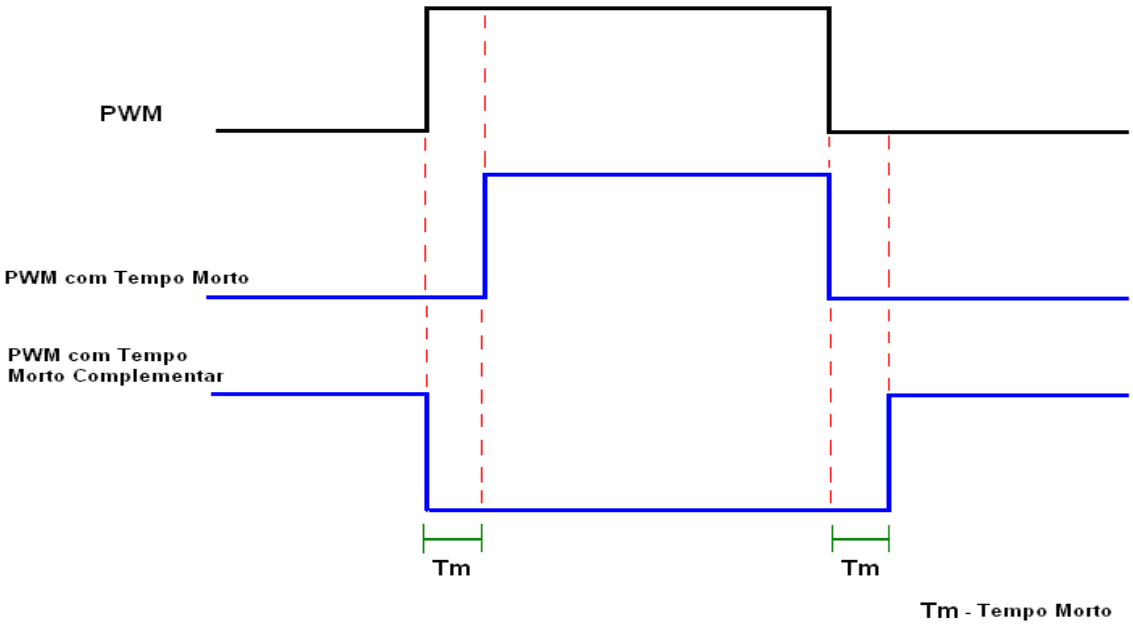


Figura 3.20 – Ilustração da inserção de tempo morto no sinal de PWM.

A cada transição do sinal PWM é inserido o tempo morto nos sinais de *"PWM com tempo morto"* e *"PWM com tempo morto complementar"*, os quais controlam as chaves do inversor.

Na transição positiva do sinal PWM, ou seja, de zero para um, o sinal de *"PWM com tempo morto"* recebe o atraso, tempo morto, e só então depois desse tempo pré-definido esse sinal passa para o valor um. Na transição negativa do sinal de PWM, ou seja, de um para zero, o sinal *"PWM com tempo morto complementar"* recebe o atraso, tempo morto, e só então depois desse tempo pré-definido é que esse sinal passa a valer um.

O fluxograma abaixo apresentado no trabalho [17] foi descrito na linguagem VHDL para inserção do tempo morto entre as chaves complementares do inversor.

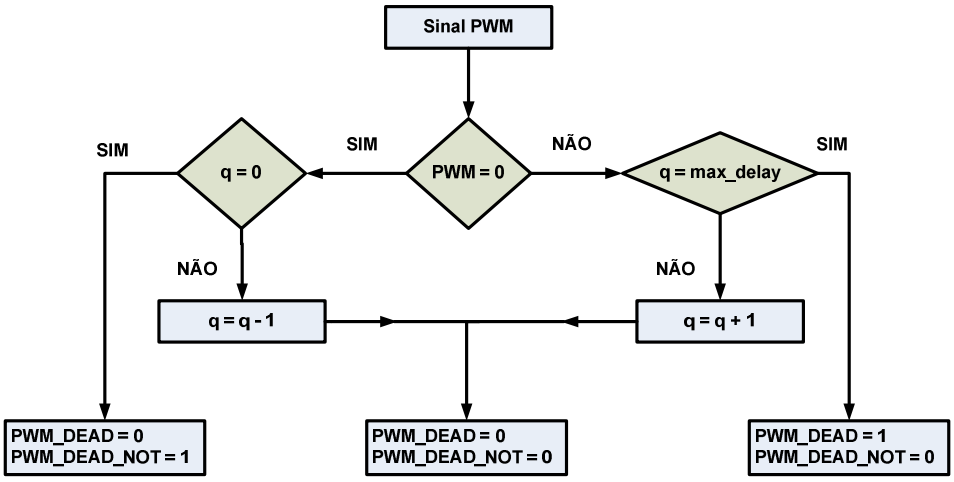


Figura 3.21 – Diagrama em blocos do algoritmo para inserção do tempo morto nas chaves complementares.

3.2.7.3 Processo para Sincronismo entre o Cálculo do Vetor de Referência e a Geração do Sinal PWM

Para cada vetor de referência fornecido ao algoritmo da Modulação Vetorial deve ser gerado um sinal de PWM. O algoritmo, porém, leva um determinado tempo para fazer os cálculos e encontrar o valor do comparador correspondente ao vetor de referência calculado.

O sinal de PWM deve ser gerado continuamente, ou seja, quando um sinal de PWM termina, um novo valor de comparação já deve estar disponibilizado no barramento para que possa ser comparado com a onda triangular simétrica e gerar um novo sinal de PWM para controle das chaves do inversor.

Todo esse sincronismo deve ser levado em conta para que o algoritmo funcione corretamente, gerando o sinal de PWM no período correto e correspondente a cada vetor de referência calculado pelo algoritmo. Utilizando a flexibilidade de desenvolvimento do FPGA, foi descrito em VHDL um processo para fazer esse sincronismo. Nesse processo, foi descrita uma máquina de estado onde são gerados todos os sinais de controle, tanto para geração de um

novo sinal PWM quanto para o cálculo de um novo vetor de referência, e cálculo de um novo valor de comparação.

A máquina de estados descrita para fazer este sincronismo esta representada na Figura 3.22.

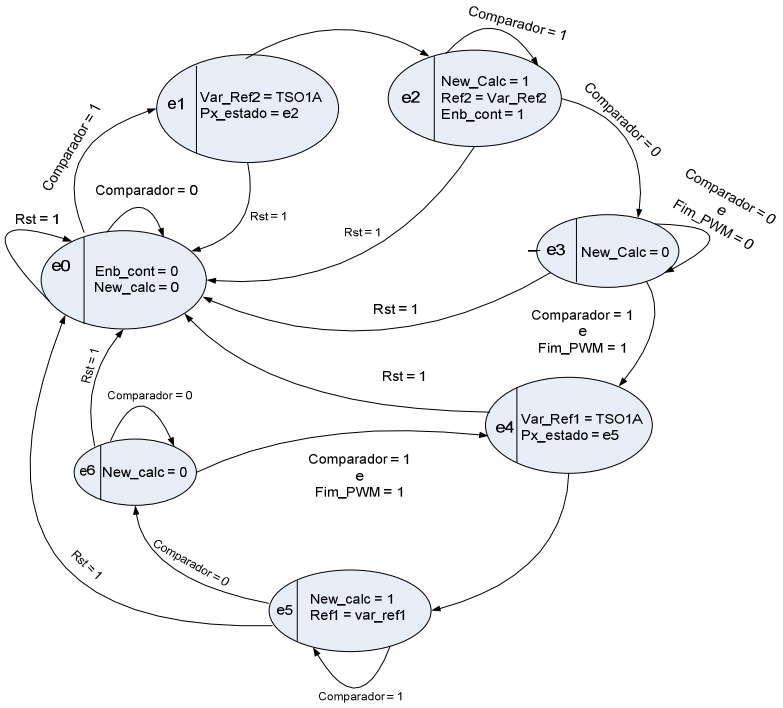


Figura 3.22 – Máquina de estado para sincronismo entre a geração do vetor de referência e a geração do sinal MLP.

O sincronismo feito pela máquina de estados compreende os cálculos do algoritmo da Modulação Vetorial, a atualização do valor do comparador e o final de um ciclo de PWM. Enquanto o algoritmo está gerando um ciclo de PWM, o algoritmo faz os novos cálculos para geração de um novo sinal PWM. No final do ciclo de geração do PWM, o registrador do comparador é atualizado com um novo valor de comparação, dessa maneira, é gerado um novo sinal de PWM.

A máquina de estados desenvolvida para o sincronismo entre os processos de geração do vetor de referência e a geração do sinal de PWM é composta de sete estados.

O estado inicial “e0” tem as funções de desabilitar o contador por meio do sinal *Enb_cont* em zero e de não habilitar um novo cálculo do tempo de chaveamento pelo algoritmo, pelo do sinal *New_calc* em zero. Com o sinal

Comparador igual a zero a máquina permanece no estado “e0” e caso ele seja igual a um, a máquina passa para o estado “e1”.

O estado “e1” tem apenas a função de armazenar o valor do tempo de chaveamento em uma variável auxiliar *Var_vref2*. Após o armazenamento, a máquina segue para o estado “e2”, onde fica em um *loop*.

O estado “e2” habilita o contador pelo sinal *Enb_cont* em um e passa o valor armazenado na variável *Var_ref2* para ser comparada com a onda triangular. A máquina permanece no estado “e2” enquanto o sinal *Comparador* for um. Caso ele se torne zero, a máquina de estados passa para o estado “e3”.

O estado “e3” desabilita o algoritmo para um novo cálculo, colocando o sinal *new_calc* em zero até que ambos os sinais *Comparador* e *fim_PWM* se tornem um, ou seja, o algoritmo já terminou os cálculos para um novo vetor de referência e o valor do tempo de chaveamento anterior já terminou de ser comparado à onda triangular.

O estado “e4” carrega o valor do novo tempo de chaveamento na variável *Var_ref1* e a máquina segue para o estado “e5”, onde a máquina de estados fica em um *loop*.

No estado “e5”, a máquina de estado habilita o algoritmo para um novo cálculo de tempo de chaveamento por meio do sinal *New_calc* em um e coloca o valor do tempo de chaveamento previamente calculado em uma variável auxiliar para ser comparado à onda triangular, permanecendo no estado “e5” até que o sinal de comparador se torne zero.

O estado “e6” desabilita o algoritmo para um novo cálculo pelo sinal *New_calc* em zero, e permanece nesse estado até que os sinais de *Comparador* e *Fim_PWM* se tornem um, fazendo com que a máquina de estados siga para o estado “e4”. A partir desse ponto a máquina de estados fica alternando entre os estados “e4”, “e5” e “e6”.

Em qualquer estado da máquina, se o sinal de *rst* (*reset*) for igual a um, a máquina de estados vai para o estado inicial “e0”.

3.3 **Conclusões**

Neste capitulo é apresentado o desenvolvimento do algoritmo da modulação vetorial utilizando a linguagem de descrição de *hardware* VHDL. Apresenta-se também a divisão do algoritmo em parte seqüencial, correspondente aos cálculos do algoritmo, e parte paralela, correspondente ao chaveamento do inversor. O sincronismo entre a geração de um vetor de referência e a geração do seu respectivo sinal PWM é feito por meio de uma máquina de estados.

A descrição do algoritmo da MV em VHDL exige a utilização de conhecimentos básicos em eletrônica digital e certa intimidade com a linguagem VHDL, uma vez que, algumas funções utilizadas para desenvolver o algoritmo não são disponíveis por esta linguagem, fazendo-se necessário buscar alternativas para o desenvolvimento destas funções.

O tempo computacional exigido pelo algoritmo é reduzido pela descrição de algumas etapas do algoritmo em paralelo e a precisão no desenvolvimento dos cálculos é obtida pelo uso da flexibilidade da linguagem VHDL.

4 IMPLEMENTAÇÃO DO ALGORITMO DA MODULAÇÃO VETORIAL EM FPGA

Neste capítulo são apresentadas as especificações do projeto, as ferramentas utilizadas para desenvolvimento e verificação, as adequação feitas para operação do circuito na frequência de operação desejada e também os recursos internos do FPGA EP2C20F484C7N utilizados.

4.1 **Especificações do projeto desenvolvido em FPGA**

O projeto foi implementado para um inversor de três níveis que sintetiza uma tensão *CA* com uma frequência fundamental de 60 Hz. A freqüência de chaveamento do inversor é em torno de 10 khz.

Sabendo que o período de um sinal de 60 Hz é, aproximadamente 16.666 ms, é preciso que cada amostra da tensão de referência seja sintetizada num período de 99,206 µs.

A frequência de operação do modulador PWM está relacionada com o valor do contador que, por sua vez, está relacionado com o período de amostragem da tensão de saída do inversor. Nesse projeto o modulador trabalha com uma frequência de 50 MHz, portanto é preciso ajustar o valor máximo do contador para que cada amostra seja sintetizada no período de 99,206 µs. Fazendo os devido cálculos, o valor máximo do contador é 2465.

O tempo morto entre as chaves complementares do inversor utilizado neste projeto é de 1,4 µs e também está relacionado com a freqüência de operação do bloco de modulação. Para atingir o tempo de 1,4 µs, a variável responsável pela determinação deste tempo deve contar até 70, isso garante o tempo morto necessário entre as chaves complementares.

No desenvolvimento deste trabalho são acrescentados dois pinos adicionais ao projeto para acompanhar o período da síntese de cada ponto de amostragem da tensão de referência e o período completo da geração da tensão de saída do inversor. Esses pinos são apresentados na sessão 4.2.4 deste capítulo.

4.2 Ferramentas Utilizadas para o Desenvolvimento do Algoritmo da Modulação Vetorial no FPGA

Para o desenvolvimento deste projeto foram utilizadas ferramentas para auxiliar o seu desenvolvimento. O MatLab® foi utilizado para desenvolver o algoritmo da MV e fornecer dados para comparação com o algoritmo desenvolvido em VHDL, o Quartus II® foi utilizado para desenvolver o código em VHDL do algoritmo e fazer simulações. O ModelSim® foi utilizado para simulação e verificação dos dados de simulação fornecidos pelo algoritmo, e por fim, o *Quartus TimeQuest Timing Analyser®* foi utilizado para fazer a análise da frequência máxima de operação do circuito sintetizado no FPGA.

Este trabalho foi implementado no FPGA EP2C20F484C7N da família *Cyclone II®*, e o desenvolvimento do *hardware* deste projeto foi feito pelo do *kit* de desenvolvimento da Altera® DK-CYCII-2C20N**.**

A seguir serão apresentados os recursos utilizados de cada uma dessas ferramentas e a sua utilização no desenvolvimento deste projeto.

4.2.1 Ferramenta para determinação da frequência máxima de operação do circuito

Apesar do FPGA da família *CycloneII®* poder trabalhar com frequências até 260 MHz [18], o circuito desenvolvido nele pode não suportar trabalhar até essa freqüência devido aos atrasos de propagação de sinais entre seus registradores internos utilizados para desenvolver o algoritmo. Para o correto funcionamento do projeto desenvolvido no FPGA é preciso determinar a sua freqüência de operação máxima.

A ferramenta *Quartus II TimeQuest Timing Analyzers®* identifica as fontes de *clock* e determina a frequência máxima de operação de cada *clock* do circuito. A sua frequência máxima é a maior velocidade em que o *clock* do projeto pode operar sem violar os tempos de *setup time* e *hold time* requeridos pelo circuito [19].

Em uma primeira análise da freqüência de operação do circuito observou-se que sua frequência máxima de operação não atingia 20 MHz, que era a frequência de operação mínima desejada para operação do modulador PWM, portanto, utilizando apenas uma fonte de *clock* não era possível atingir as

necessidades do projeto. De acordo com a natureza do algoritmo, nota-se que ele pode ser dividido em duas partes, uma parte de cálculo para o algoritmo da MV e uma parte de modulação dos sinais das chaves.

Fazendo essa divisão no algoritmo desenvolvido e aplicando duas fontes de *clock* distintas no FPGA foi possível fornecer um sinal de *clock* para o modulador maior que 20 MHz e, para a parte de cálculos um sinal de 10 MHz. Dessa maneira, o circuito desenvolvido no FPGA atende aos requisitos desejáveis para a operação do algoritmo da MV.

Essa estratégia de desenvolvimento proporcionou uma maior precisão na geração dos sinais PWM de controle das chaves do inversor, uma vez que, em comparação com o trabalho [15] para a implementação do algoritmo em um DSP, o contador passa de 991, com uma freqüência de 20 MHz, para 2465, com uma frequência de 50 MHz na implementação do algoritmo em FPGA.

Na Figura 4.1 é apresentado o esquema das ligações dos *clocks* na parte de cálculos do algoritmo e na parte de geração dos sinais PWM das chaves do inversor.

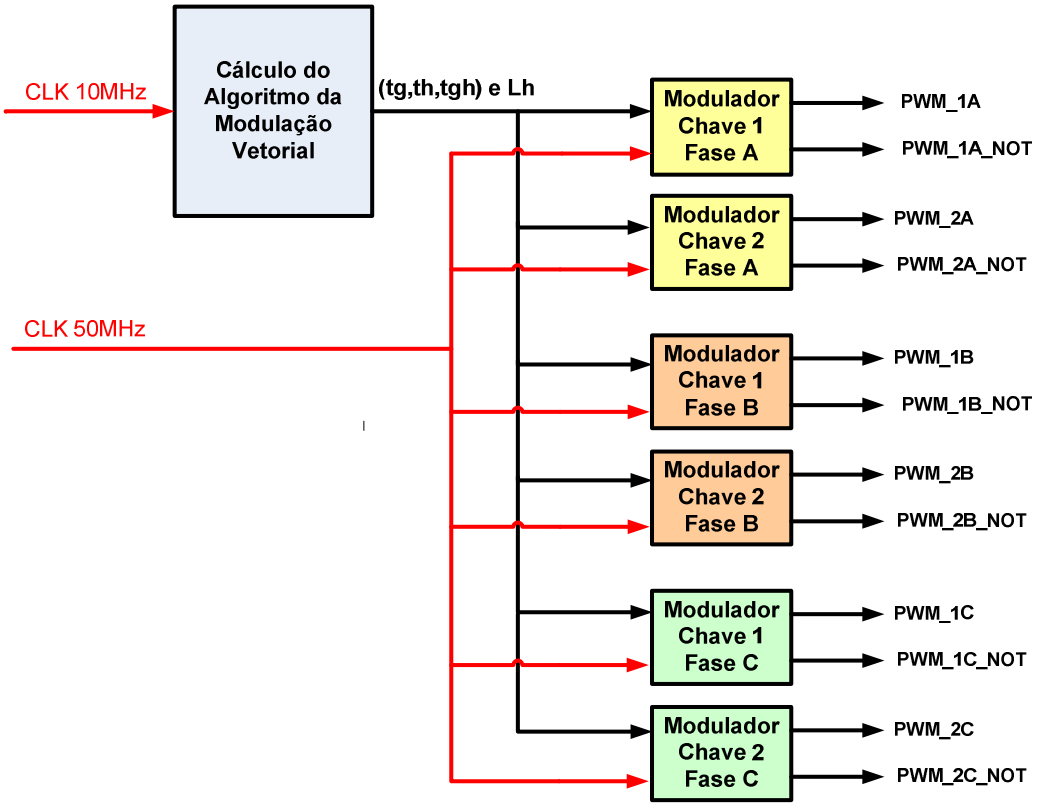


Figura 4.1 – Ligação dos *clocks* na parte serial e paralela do algoritmo.

4.2.2 Desenvolvimento do Algoritmo da Modulação Vetorial no MatLab®

O algoritmo da Modulação Vetorial foi implementado no MatLab® para fornecer dados de simulação para comparação e validação dos dados de simulação obtidos com o algoritmo desenvolvido em VHDL.

Foi desenvolvido um *script* no MatLab® para desenvolver o algoritmo da MV e armazenar, em matrizes alguns resultados de simulação do algoritmo. Dessa maneira, são fornecidos dados para serem comparados com os dados de simulação do algoritmo descrito em VHDL.

Para uma melhor visualização da comparação dos dados obtidos com a simulação do algoritmo desenvolvido no MatLab® e descrito em VHDL, utilizou-se o MatLab® também para geração de gráficos com os valores dos dados de simulação de ambos os algoritmos.

4.2.3 Verificação dos Dados de Simulação do Algoritmo da Modulação Vetorial com o ModelSim®

Para verificação dos resultados da simulação, utilizou-se o programa ModelSim®, em que uma estrutura de verificação foi elaborada *(testbench)* para comparar os dados obtidos com a simulação do algoritmo descrito em VHDL e o algoritmo implementado em MatLab®.

Os dados dos valores de comparação com a onda triangular foram armazenados em um arquivo-texto com a formatação de uma matriz para ser utilizada no programa MatLab®.

Um pequeno *script"* foi desenvolvido no MatLab® para ler e tratar as matrizes criadas pelo *testbench*, desenvolvido no ModelSim®. Após a execução do script no MatLab®, foram gerados gráficos com os valores dos comparadores de todas as chaves do inversor. Esses gráficos foram comparados com os gráficos obtidos com o algoritmo da Modulação Vetorial desenvolvido no MatLab®.

A Figura 4.2 mostra os gráficos gerados pelo MatLab® com os dados obtidos com o ModelSim® de todos os valores de comparação das chaves, com valor do índice de modulação 0,70.

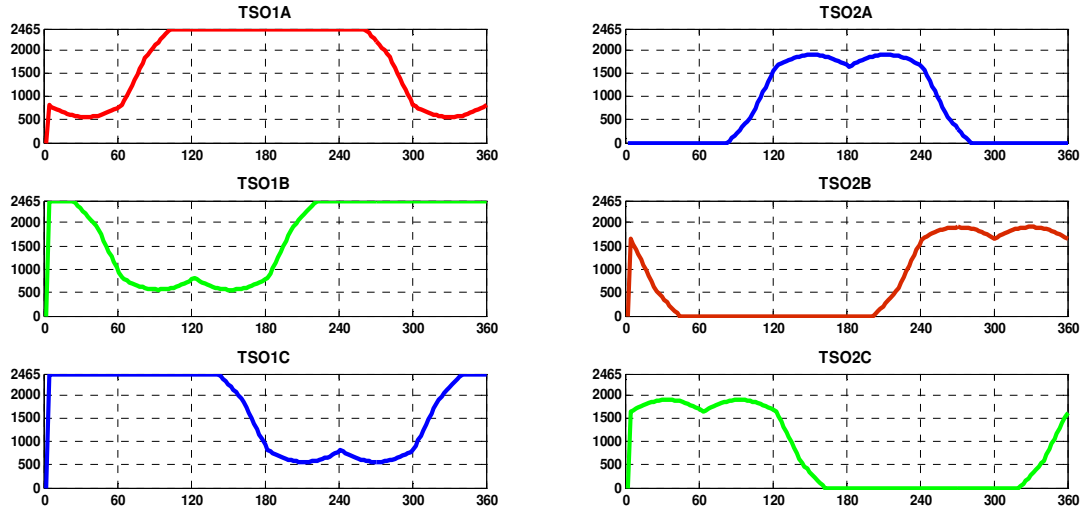


Figura 4.2 – Gráfico com todos os valores das chaves do inversor para índice de modulação igual a 0,70.

O *scrip"* desenvolvido em MatLab® e o *testbench* desenvolvido no ModelSim® encontram-se anexos no final deste documento.

4.2.4 Descrição e Simulação do Algoritmo da MV no Quartus II

O algoritmo da Modulação Vetorial foi descrito na linguagem VHDL pelo editor do programa Quartus II®. Após o algoritmo ser compilado e sintetizado, o mesmo foi simulado para observar se os dados fornecidos pelo algoritmo estavam coerentes e se os tempos de execução de cada etapa do algoritmo estavam sendo atingidos.

A análise da simulação do algoritmo da Modulação Vetorial no FPGA foi feita pelo *Waveform*. Pinos extras foram adicionados ao projeto para análise dos tempos de resposta do algoritmo com relação ao tempo de geração de cada ciclo PWM e também de todos os ciclos PWM que formam a tensão *Van* na frequência de 60 Hz.

Na Figura 4.3 é apresentado o tempo para apenas um padrão de chaveamento.

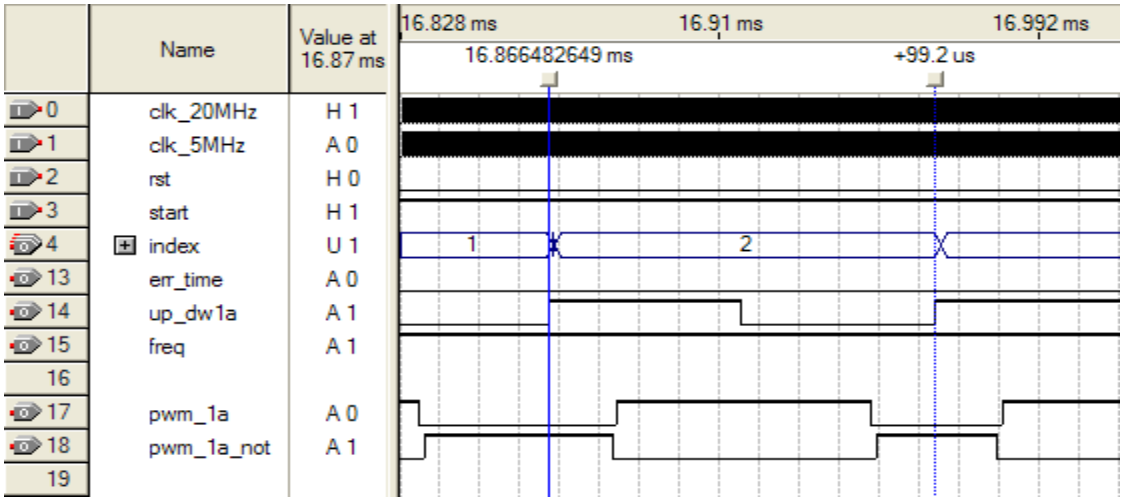


Figura 4.3 – Período para geração de um padrão de chaveamento.

O pino “up_dw1a”, relacionado na figura 4.3, foi criado para análise do tempo de geração de cada padrão de chaveamento. O sinal desse pino recebe 1 (um) quando o contador está contando em modo crescente, e recebe 0 (zero) quando o contador está contando em modo decrescente. Desta maneira, é possível observar o período de geração de um ciclo do PWM.

Na Figura 4.4 é apresentada a simulação para validação da frequência da tensão de saída sintetizada pelo inversor.

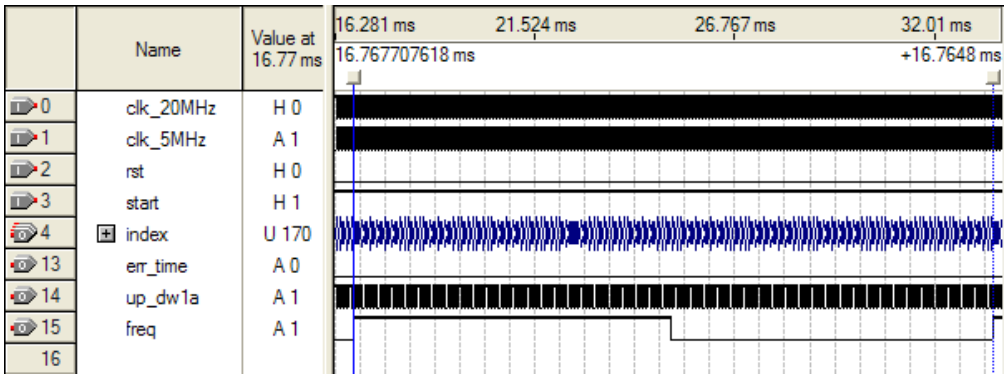


Figura 4.4 – Período da geração da forma de onda da tensão do inversor.

A tensão de saída do inversor deve apresentar uma frequência de 60 Hz, ou seja, essa forma de onda deve ter um período de, aproximadamente, 16,66 ms. Para que isso ocorra o algoritmo deve sintetizar todos os padrões nesse intervalo de tempo.

O pino *freq*, apresentado na Figura 4.4, foi adicionado ao projeto para observar a frequência da forma de onda gerada na saída do inversor. Esse pino recebe 1 (um) quando o vetor de referência encontra-se nos quadrantes 1 e 2 e

recebe 0 (zero) quando ele se encontra nos quadrantes 3 e 4 do círculo trigonométrico. Observa-se que o pino *freq* apresenta um período de 16,76 ms, muito próximo dos 16,66 ms esperados, apresentando apenas um pequeno erro de aproximadamente 0,1 ms.

4.3 **Estrutura utilizada para síntese do algoritmo da Modulação Vetorial no FPGA**

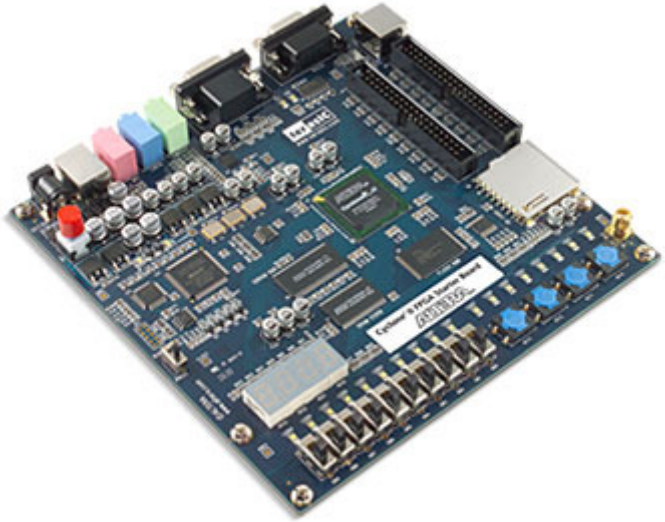
Para ganhar tempo e agilidade no desenvolvimento deste projeto utilizou-se o *kit* de desenvolvimento DK-CYCII-2C20N da Altera® para o desenvolvimento do *hardware* deste projeto.

O *kit* contém as seguintes características:

a) Placa de Desenvolvimento *Cyclone II®* Starter Development
b) FPGA Cyclone II EP2C20F484C7N
c) Configuração
    i. Cabo de gravação USB-Blaster™
    ii. Dispositivo de configuração EPCS4
d) Memória
    i. 8-Mbyte SDRAM
    ii. 512-Kbit SRAM
    iii. 4-Mbyte flash
e) Clock
    i. Conector SMA (entrada de clock externa)
f) Áudio
    i. Codificador/Decodificador (CODEC)
g) Chaves e Indicadores
    i. Dez chaves e quatro botões de toque
    ii. Quatro *displays* de 7 segmentos
    iii. Dez *Leds* vermelhos e oito *Leds* verdes
h) Conectores
    i. Portas para VGA, RS-232, e PS/2
    ii. Duas portas de expansão com 40 pinos cada uma
    iii. Soquete SD/MMC
i) Cabos e Alimentação

i. Alimentação a partir do cabo USB ou alimentação externa pelo cabo de alimentação (recomendado quando usa o *kit* com acessórios)

Na Figura 4.5 há a ilustração da placa de desenvolvimento utilizada no projeto.



Fonte: www.altera.com

Figura 4.5 – Figura de ilustração da placa de desenvolvimento.

A tabela 4.1 apresenta as características internas do FPGA Cyclone EP2C20F484C7N.

Tabela 4.1– Tabela com as características do FPGA EP2C20F484C7N.

Características	Quantidade
Elementos Lógicos	18.752
Blocos de memória RAM	52
Total de *bits* de RAM	239.616
Multiplicadores Embarcados	52
PLL	4
Pinos de entrada e saída	315

4.4 **Características do FPGA com o Algoritmo de Modulação Vetorial**

A simulação e o desenvolvimento de todo o projeto até a gravação no *kit* de desenvolvimento foi feito utilizando o programa Quartus® II. O dispositivo FPGA EP2C20F484C7N da família *Cyclone II*® é o dispositivo fornecido na placa de desenvolvimento.

A Figura 4.6 ilustra o algoritmo desenvolvido no FPGA, nela, é possível observar a disposição dos blocos no FPGA e o fluxo dos dados no desenvolvimento do algoritmo e também os pinos de entrada e saída do FPGA.

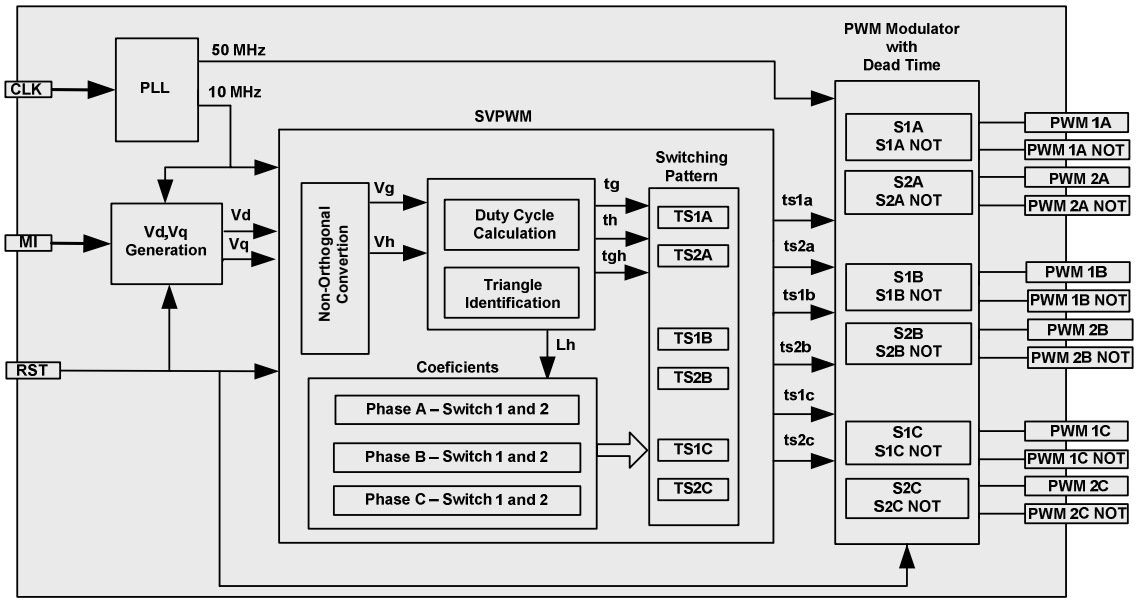


Figura 4.6 – Diagrama em blocos do algoritmo da MV para inversores de três níveis desenvolvido em FPGA.

Na ilustração do algoritmo da MV desenvolvido, observam-se os pinos de entrada, *CLK* corresponde à entrada do sinal de *clock* principal, *MI* corresponde ao valor de índice de modulação e *RST* corresponde ao sinal de *reset*. Os pinos de saída do FPGA são os sinais de controle de todas as chaves do inversor, são eles: para a fase A: PWM_1A, PWM_1A_NOT, PWM_2A, PWM_2A_NOT, para a fase B: PWM_1B, PWM_1B_NOT, PWM_2B, PWM_2B_NOT e para a fase C: PWM_1C, PWM_1C_NOT, PWM_2C, PWM_2C_NOT.

O projeto completo do algoritmo da MV desenvolvido para um inversor trifásico de três níveis utilizou um total de 8000 elementos lógicos do FPGA. Foram compilados todos os blocos de desenvolvimento do algoritmo, bem como

blocos de inicialização e interconexões entre todos eles. A Figura 4.7 apresenta o resumo dessa compilação.

Flow Status	Successful - Fri Nov 14 09:50:20 2008
Quartus II Version	7.2 Build 151 09/26/2007 SJ Web Edition
Revision Name	Inversor3Niveis
Top-level Entity Name	Inversor3Niveis_FINAL
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	No
Total logic elements	8,000 / 18,752 (43 %)
Total combinational functions	7,802 / 18,752 (42 %)
Dedicated logic registers	868 / 18,752 (5 %)
Total registers	868
Total pins	66 / 315 (21 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	52 / 52 (100 %)
Total PLLs	1 / 4 (25 %)

Figura 4.7 – Resumo da compilação do projeto completo para um inversor de três níveis.

Para observar a simplificação do algoritmo da MV desenvolvido pelo trabalho [4], foi feita a compilação apenas do bloco responsável pelo desenvolvimento do algoritmo. Nesse bloco, são feitos todos os cálculos para localizar o sextante em que o vetor de referência esta localizado, a normalização do vetor de referência, a identificação do triângulo no hexágono e o cálculo das razões cíclicas.

Flow Status	Successful - Tue Feb 17 09:54:38 2009
Quartus II Version	7.2 Build 151 09/26/2007 SJ Web Edition
Revision Name	Inversor3Niveis
Top-level Entity Name	svpwm
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	No
Total logic elements	316 / 18,752 (2 %)
Total combinational functions	256 / 18,752 (1 %)
Dedicated logic registers	126 / 18,752 (< 1 %)
Total registers	126
Total pins	115 / 315 (37 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	11 / 52 (21 %)
Total PLLs	0 / 4 (0 %)

Figura 4.8 – Resumo da compilação do bloco de desenvolvimento do algoritmo da MV simplificado.

Observando a compilação do código em VHDL apenas do algoritmo da MV, Figura 4.8, comprova-se sua simplicidade refletida sob a baixa utilização de recursos lógicos do FPGA.

O modulador PWM de cada chave do algoritmo foi desenvolvido separadamente do algoritmo, uma vez que ele pode ser desenvolvido para cada par de chaves do inversor. Esse bloco compreende o cálculo do tempo de chaveamento e a geração do sinal PWM com tempo morto das chaves complementares. A Figura 4.9 apresenta o resumo da compilação do bloco modulador para a chave 1 da fase A e sua chave complementar.

Flow Status	Successful - Tue Feb 17 10:16:11 2009
Quartus II Version	7.2 Build 151 09/26/2007 SJ Web Edition
Revision Name	Inversor3Niveis
Top-level Entity Name	Modulador
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	Yes
Total logic elements	295 / 18,752 (2 %)
Total combinational functions	258 / 18,752 (1 %)
Dedicated logic registers	134 / 18,752 (< 1 %)
Total registers	134
Total pins	127 / 315 (40 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	10 / 52 (19 %)
Total PLLs	0 / 4 (0 %)

Figura 4.9 – Resumo da compilação do bloco modulador PWM da chave 1 da fase A e sua chave complementar.

4.5 Visão Geral do *Hardware* Utilizado para Desenvolvimento do Projeto

A partir do algoritmo de modulação vetorial gravado no FPGA, os pinos correspondentes aos sinais de PWM para controle das chaves do inversor são ligados a uma outra placa para isolamento da parte digital do projeto da parte de potência. Isso é feito utilizando uma placa com acopladores ópticos. Um acoplador é utilizado para cada sinal de PWM, totalizando 12 acopladores na placa.

Somente após os sinais de PWM passarem pelos acopladores, esses sinais são enviados para a placa do inversor. Utilizando a placa com os acopladores, o kit de desenvolvimento fica isolado de qualquer problema que possa acontecer no inversor.

Na figura 4.10 é apresentada a visão geral da disposição dos *hardwares* envolvidos no desenvolvimento deste projeto.

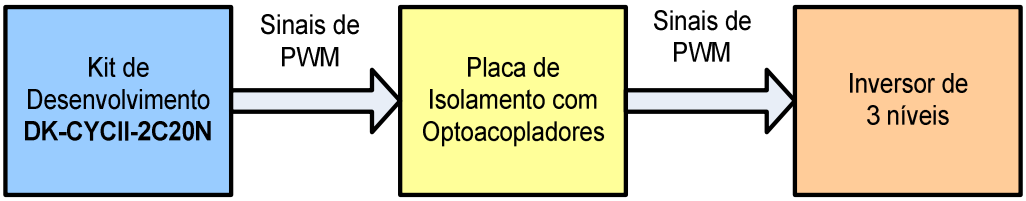


Figura 4.10 – *Hardware* utilizado para desenvolvimento do projeto.

Nas Figuras 5.28 e 5.29, no próximo capítulo, é apresentado ilustrações do protótipo desenvolvido, em que é observado o *kit* de desenvolvimento, a placa com os acopladores óticos e o inversor de três níveis.

4.6 **Conclusões**

Neste capítulo foram apresentadas a especificação do FPGA e as ferramentas utilizadas para o desenvolvimento do projeto. O ModelSim®, para verificação, o MatLab®, para gerar dados de comparação e o Quartus® II para o desenvolvimento do código VHDL.

A utilização dos programas ModelSim® em conjunto com o MatLab® para a verificação do projeto foi de grande importância pois houve um ganho de tempo para fazer as verificações dos cálculos do algoritmo sintetizado no FPGA.

A divisão do algoritmo em uma parte paralela e outra seqüencial proporcionou uma melhor precisão na geração dos sinais PWM uma vez que o modulador passa trabalhar em uma maior freqüência.

Os resumos das compilações parciais e total do projeto desenvolvido mostram a simplicidade do algoritmo, pois a síntese do algoritmo no utilizou poucos recursos lógicos do FPGA.

5 RESULTADOS EXPERIMENTAIS

Para validação do projeto e análise das formas de onda da tensão de saída do inversor foi utilizado um inversor de três níveis diodo de grampeamento. O projeto utiliza o *kit* de desenvolvimento da Altera® DK-CYCII2C20N, uma interface de acoplamento óptico e o inversor de três níveis, apresentados na Figura 5.29.

A carga utilizada para obtenção da forma de onda da tensão de saída do inversor é uma carga resistiva trifásica conectada em Y. Essa carga é obtida pela conexão de elementos resistivos de 270 Ω e 25 W, resultando em uma resistência por fases de, aproximadamente, 136 Ω e 200 W.

O inversor de três níveis é composto de 12 interruptores MOSFETs (k2740), seis diodos de grampeamento ultra-rápidos (MUR260). Os sinais de controle das chaves do inversor são isolados individualmente por meio de optoacopladores (HPCL 3180).

Os resultados obtidos com o desenvolvimento deste projeto estão divididos em dois grupos, um grupo apresenta as simulações utilizando os programas: MatLab®, ModelSim® e QuartusII® e outro grupo mostra os resultados práticos, onde são apresentadas as simulações para o controle do inversor de três níveis utilizando o algoritmo da MV sintetizado no FPGA.

O programa ModelSim® foi utilizado para capturar e armazenar em arquivo os resultados de simulação do projeto em conjunto com o programa MatLab®, no qual foi desenvolvido um *script* para transformar os dados de simulação em gráficos, facilitando as análises dos resultados obtidos.

A partir do programa ModelSim®, os dados de simulação foram armazenados em arquivo para análise. As seguintes variáveis do desenvolvimento do projeto foram analisadas:

a) vetor de referência, (Vd,Vq);
b) componentes normalizadas, Vg e Vh;
c) identificação do sextante, Ls;
d) identificação do triângulo dentro do hexágono, Lh;
e) tempos de chaveamento, tg,th r tgh;
f) valor de comparação para formar o PWM.

As variáveis citadas foram calculadas também com a utilização do programa MatLab®. Nas figuras a seguir são apresentados os gráficos com os valores dessas variáveis calculadas no MatLab® e em VHDL.

5.1 Resultados de simulação obtidos com os programas ModelSim® e MatLab®.

Os gráficos a seguir apresentam os resultados de simulação do algoritmo implementado em MatLab® e descrito em VHDL. Um gráfico com a diferença entre essas duas implementações é apresentado para que seja feita uma análise da performance dos resultados obtidos com o desenvolvimento em VHDL.

5.2.1 Simulação do vetor de referência

A Figura 5.1 mostra o resultado da simulação para o cálculo do vetor de referência (Vd,Vq) no MatLab.

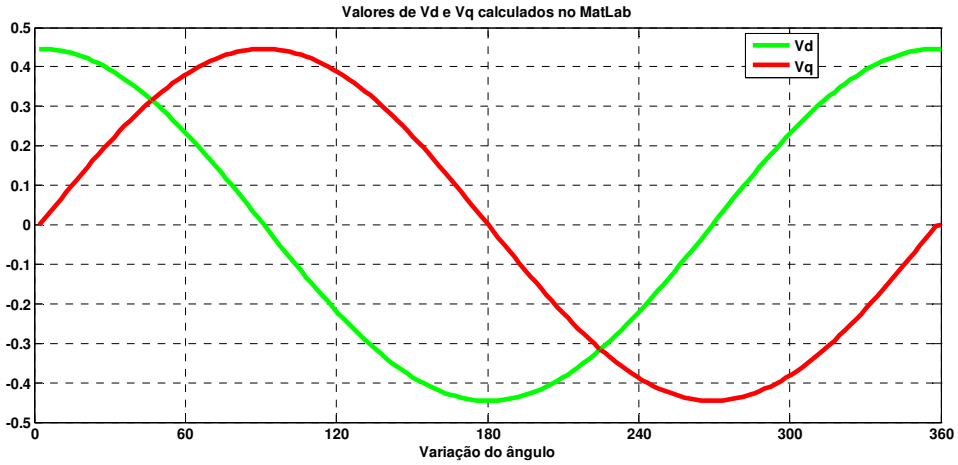


Figura 5.1 – Cálculo do vetor de referência utilizando o MatLab®.

A Figura 5.2 mostra o resultado da simulação do cálculo do vetor de referência em VHDL.

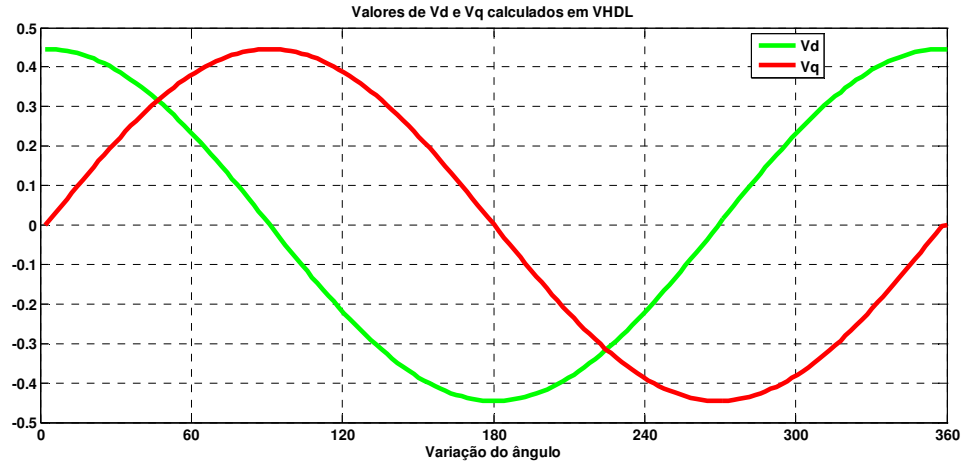


Figura 5.2 – Cálculo do vetor de referência em VHDL para índice de modulação 0,70.

Observa-se na Figura 5.3 que a diferença entre os valores do vetor de referência calculado por MatLab® e em VHDL é muito pequena, chega à ordem de apenas 0,002 de diferença.

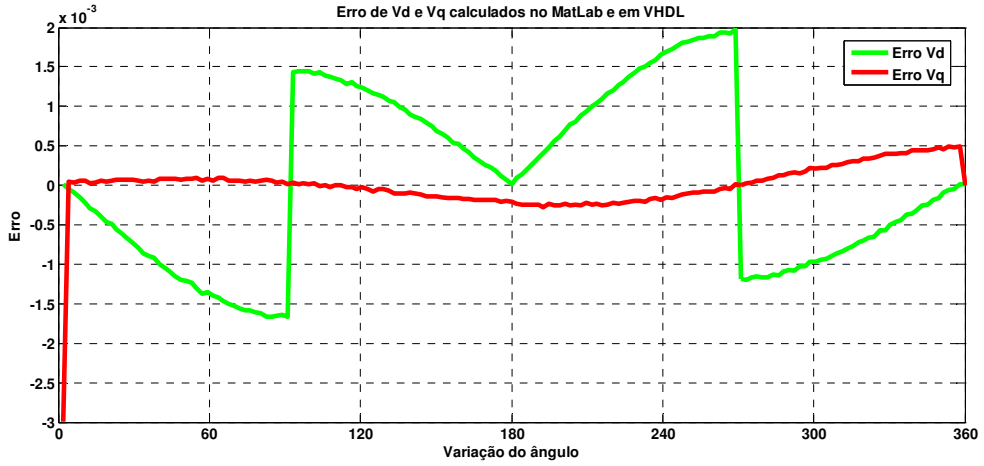


Figura 5.3 – Diferença entre o cálculo do vetor de referência no MatLab e em VHDL para índice de modulação 0,70.

5.2.2 Simulação da normalização do vetor de referência nas coordenadas moveis não-ortogonais.

A Figura 5.4 apresenta a figura característica do vetor de referência normalizado na coordenada Vg calculado pelo MatLab.

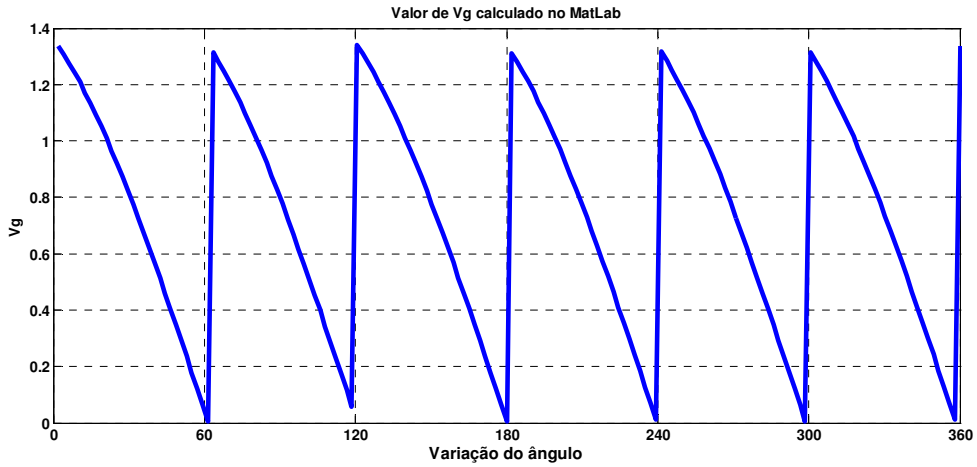


Figura 5.4 – Cálculo de Vg no MatLab com índice de modulação 0,70.

A Figura 5.5 apresenta a figura característica com os valores do vetor de referência normalizado na coordenada Vg calculado em VHDL.

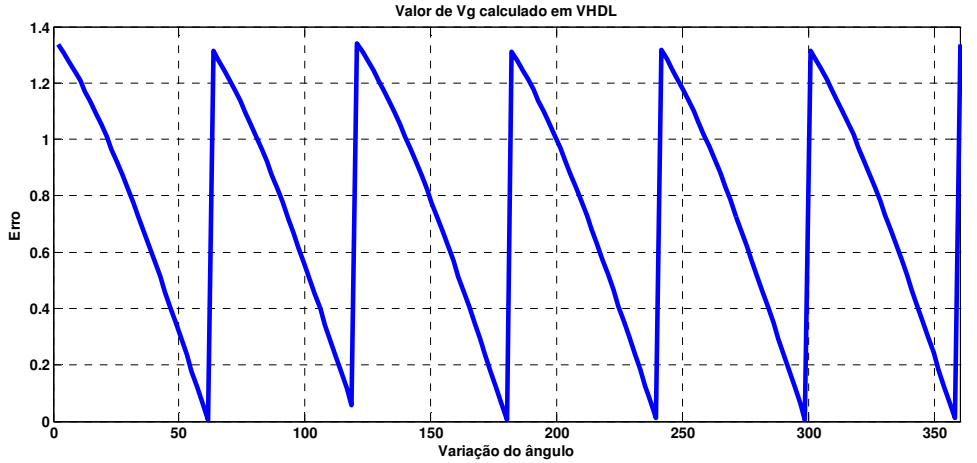


Figura 5.5 – Cálculo de Vg em VHDL com índice de modulação 0,70.

Observa-se, na Figura 5.6, a diferença entre os cálculos da coordenada Vg feito em MatLab® e em VHDL.

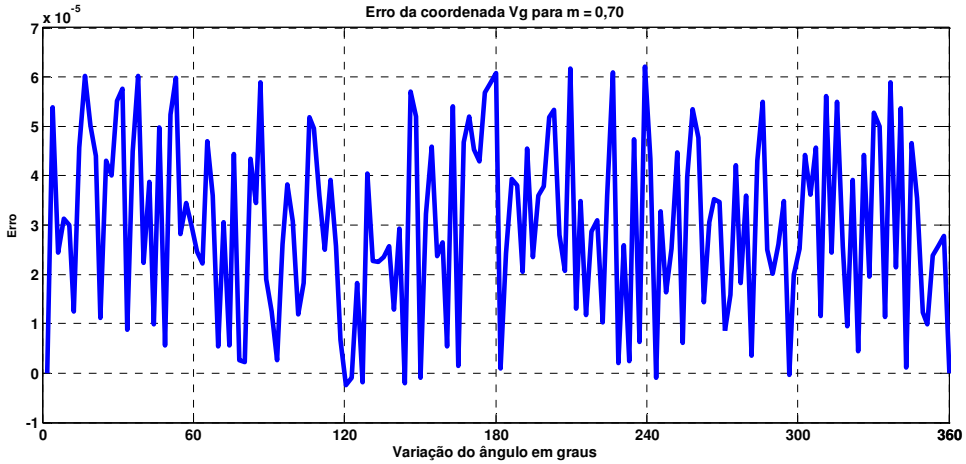


Figura 5.6 – Diferença da coordenada Vg calculada pelo MatLab e em VHDL com índice de modulação 0,70.

Na Figura 5.7 é apresentado o gráfico com os valores de Vh calculado pelo MatLab®.

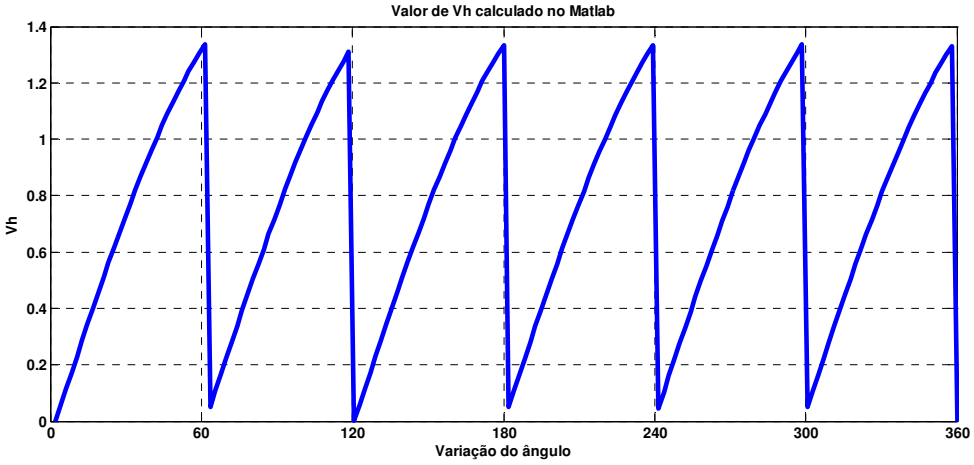


Figura 5.7 – Coordenada Vh calculada no MatLab para índice de modulação 0,70.

A Figura 5.8 apresenta a figura com os valores da coordenada Vh calculada em VHDL.

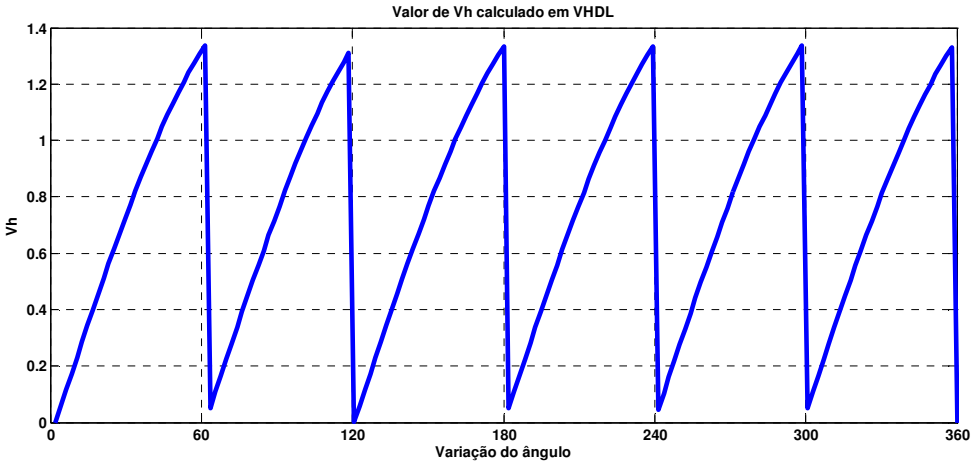


Figura 5.8 – Coordenada Vh calculada em VHDL para índice de modulação 0,70.

A Figura 5.9 apresenta o erro entre os valores da coordenada Vh calculada em VHDL e em MatLab®.

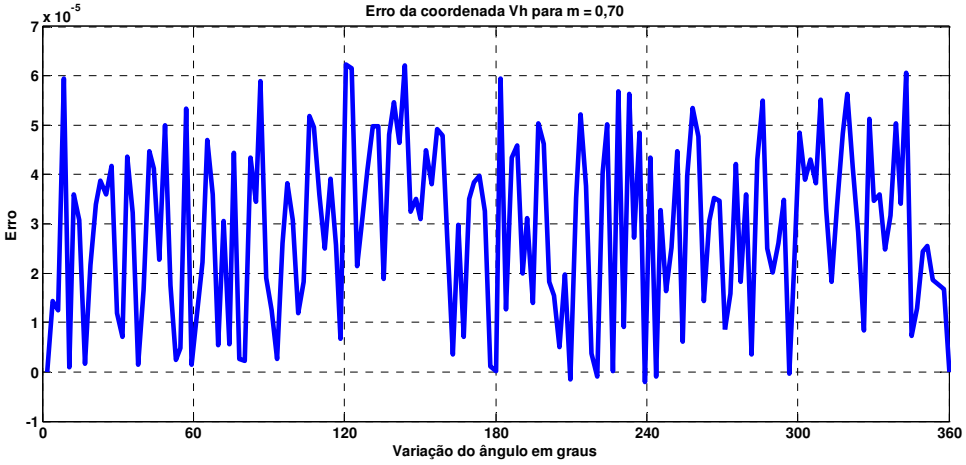


Figura 5.9 – Erro entre Vh calculado em VHDL e em MatLab® com índice de modulação 0,70.

A análise do desempenho dos cálculos para a normalização do vetor de referência em coordenadas móveis não-ortogonais desenvolvida em VHDL pode ser feita por meio das Figuras 5.3, 5.6 e 5.9. Nessas figuras, é observado um erro muito pequeno entre os cálculos feitos em MatLab® e feito em VHDL, o maior erro para Vg e Vh chega próximo de 6x10e-6.

5.2.3 Simulação da identificação do triângulo e sextante

A Figura 5.10 apresenta o valor do sextante calculado no MatLab®.

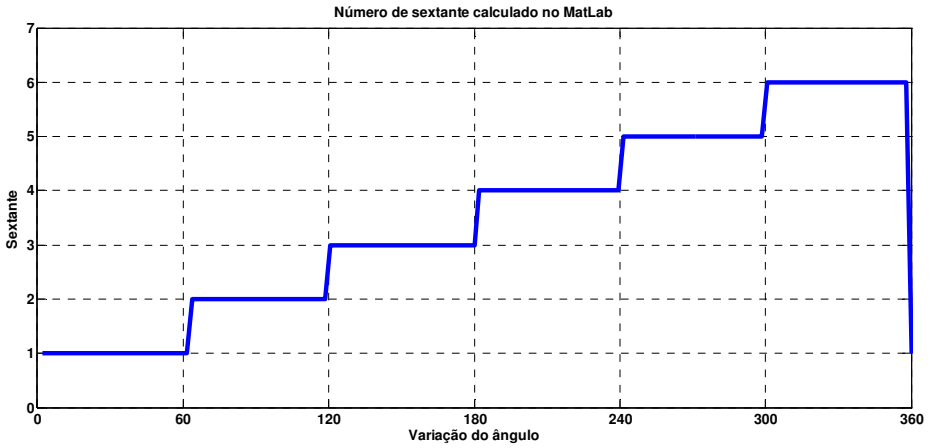


Figura 5.10 – Cálculo do sextante em MatLab®.

A Figura 5.11 apresenta o cálculo do número do sextante calculado em VHDL.

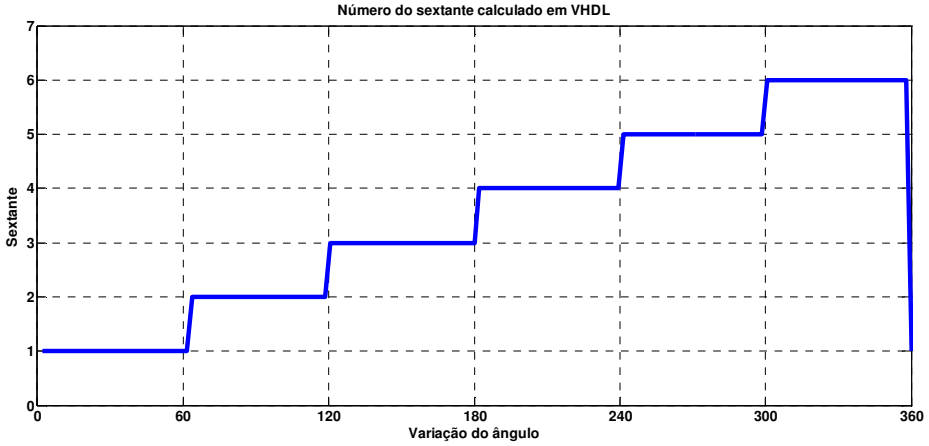


Figura 5.11 – Cálculo do sextante em VHDL.

Na Figura 5.12 é apresentado o erro entre os cálculos do valor do sextante no MatLab® e em VHDL.

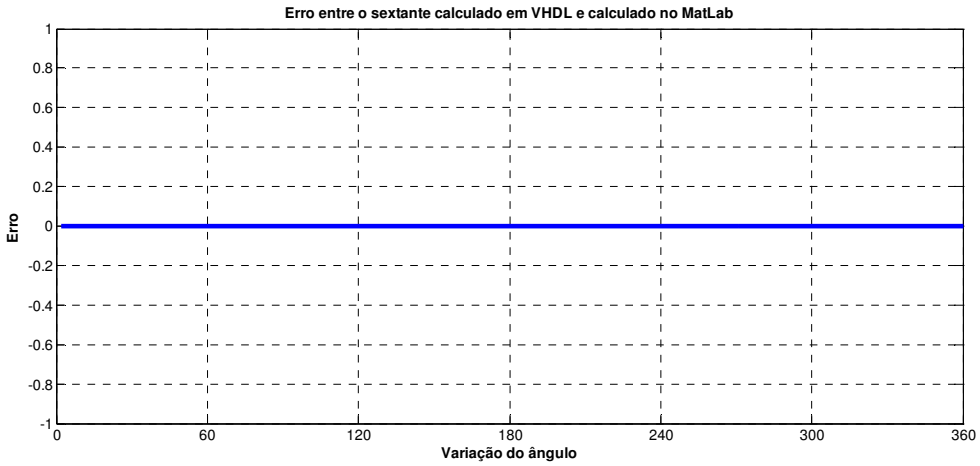


Figura 5.12 – Diferença entre os cálculos do valor do sextante em VHDL e em MatLab® para índice de modulação 0,70.

A Figura 5.13 a seguir mostra os valores do número do triângulo dentro do hexágono calculado em MatLab®.

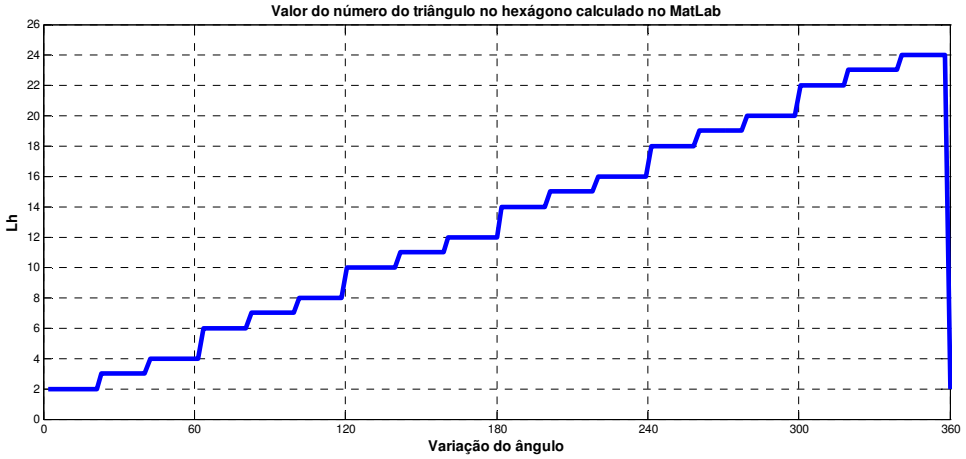


Figura 5.13 – Valor de Lh calculado em MatLab® para índice de modulação 0,70 para índice de modulação 0,70.

A Figura 5.14 apresenta o valor de Lh calculado em VHDL.

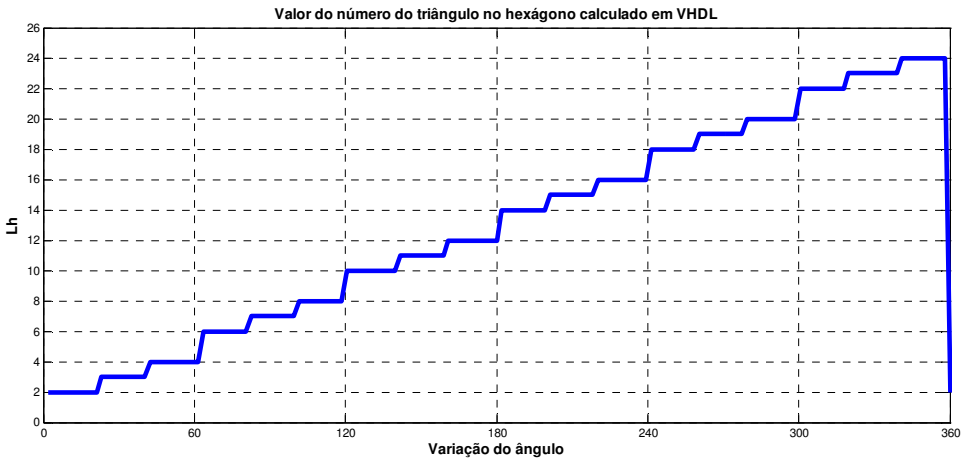


Figura 5.14 – Variável Lh calculada em VHDL para índice de modulação 0,70.

A Figura 5.15 apresenta o erro entre os cálculos da variável Lh feito em MatLab e feito em VHDL.

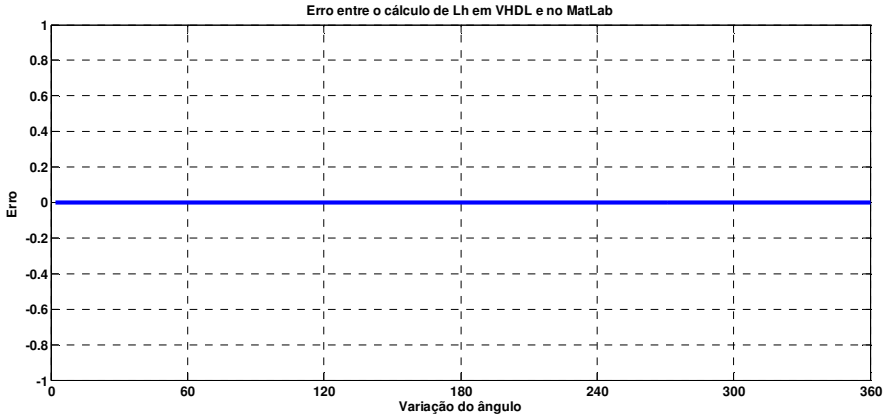


Figura 5.15 – Erro entre o cálculo de Lh feito em VHDL e MatLab® para índice de modulação 0,70.

A performance dos cálculos da identificação do sextante e a localização do triângulo feitas em VHDL pode ser observada na Figura 5.15 e Figura 5.12, nelas se observa que o erro foi zero, ou seja, os cálculos feitos em VHDL e em MatLab® tiveram o mesmo resultado, o que comprova a eficiência dos cálculos feitos em VHDL.

5.2.4 Simulação das razões cíclicas tg, th e tgh.

A Figura 5.16 apresenta o valor da variável tg calculada em MatLab.

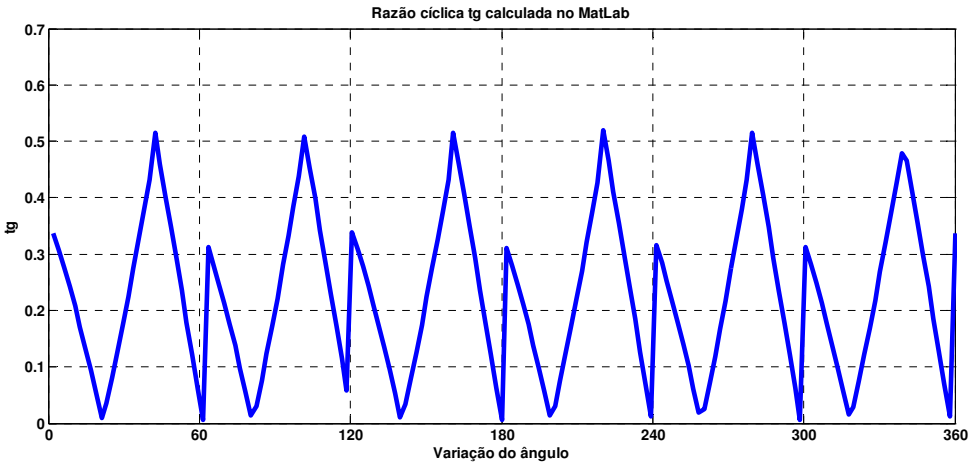


Figura 5.16 – Valor de tg calculado em MatLab® para índice de modulação 0,70.

A Figura 5.17 apresenta o valor do cálculo da variável tg em VHDL.

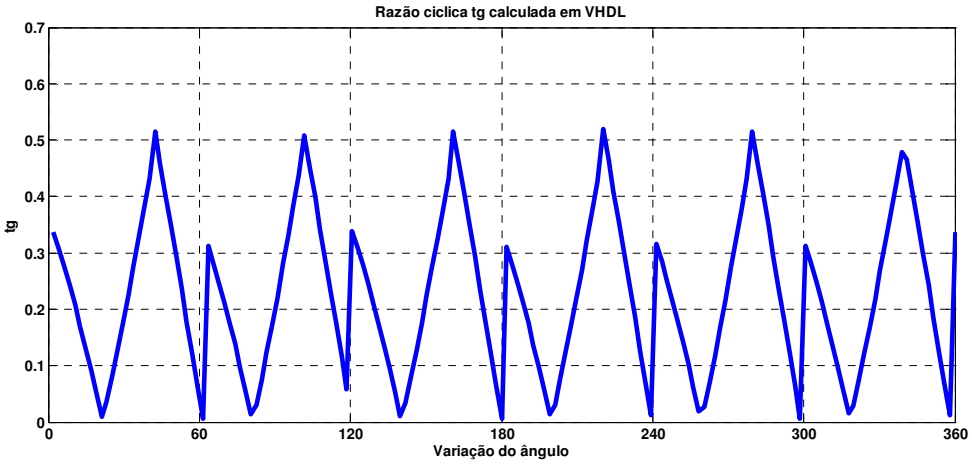


Figura 5.17 – Valor de tg calculado em VHDL para índice de modulação 0,70.

A Figura 5.18 apresenta o erro de cálculo entre a variável tg calculada em MatLab® e em VHDL.

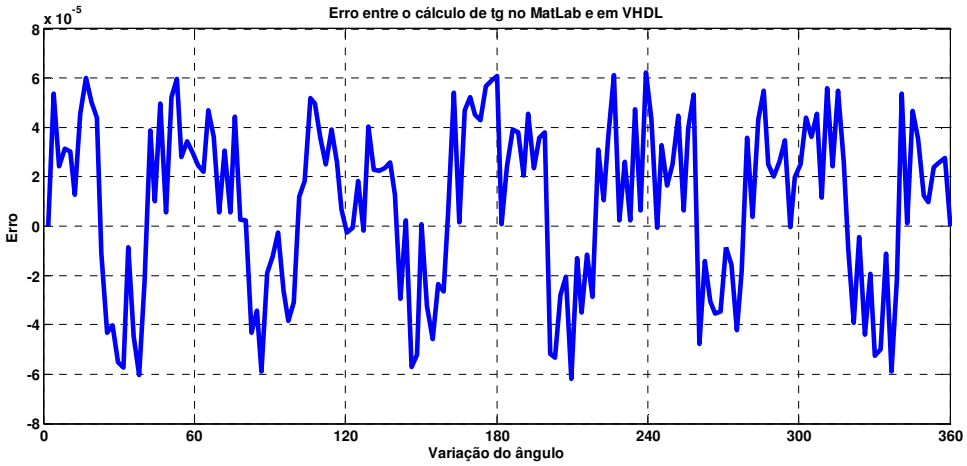


Figura 5.18 – Erro de cálculo entre a variável tg calculada em MatLab e em VHDL.

A Figura 5.19 apresenta o valor da variável th calculada em MatLab.

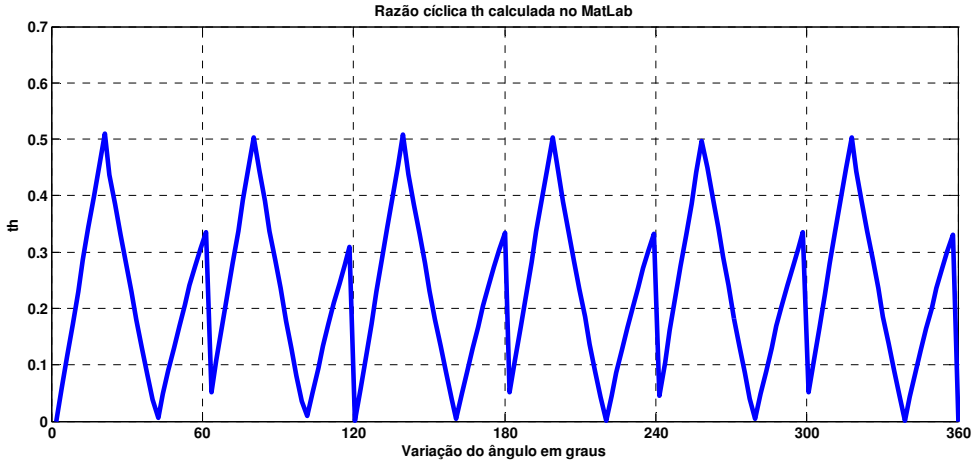


Figura 5.19 – Valor de th calculado em MatLab® para índice de modulação 0,70.

A Figura 5.20 apresenta o cálculo da variável th calculada em VHDL.

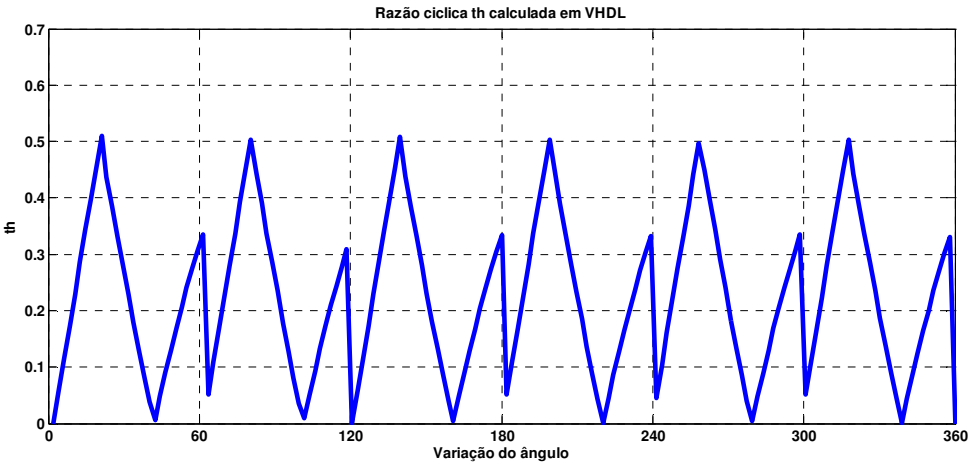


Figura 5.20 – Valor de th calculado em VHDL para índice de modulação 0,70.

A Figura 5.21 apresenta o erro de cálculo entre a variável th calculada no MatLab® e em VHDL.

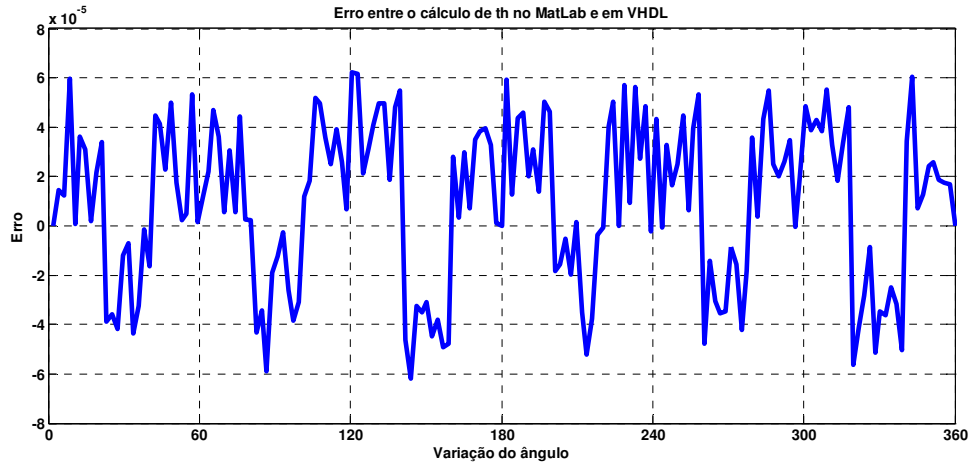


Figura 5.21 – Erro de cálculo entre a variável th calculada em MatLab® e em VHDL.

A Figura 5.22 apresenta o valor de tgh calculado no MatLab®.

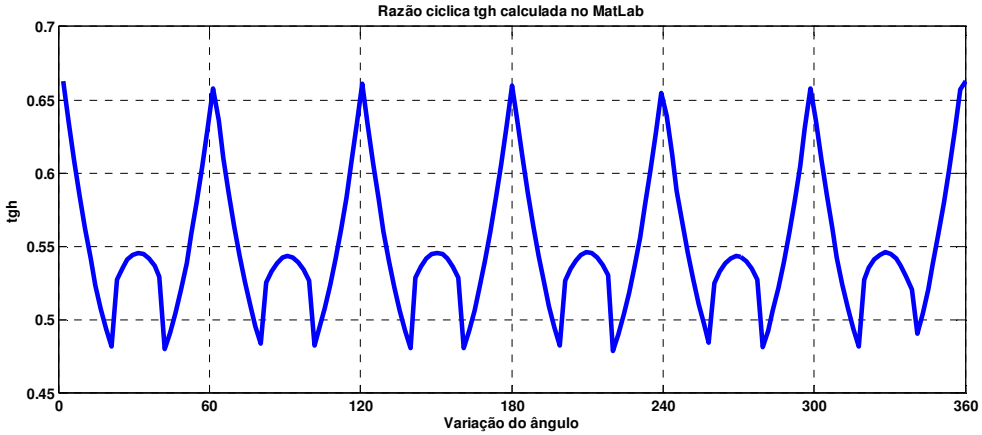


Figura 5.22 – Valor de tgh calculado em MatLab® para índice de modulação 0,70.

A Figura 5.23 apresenta o cálculo da variável tgh calculado em VHDL.

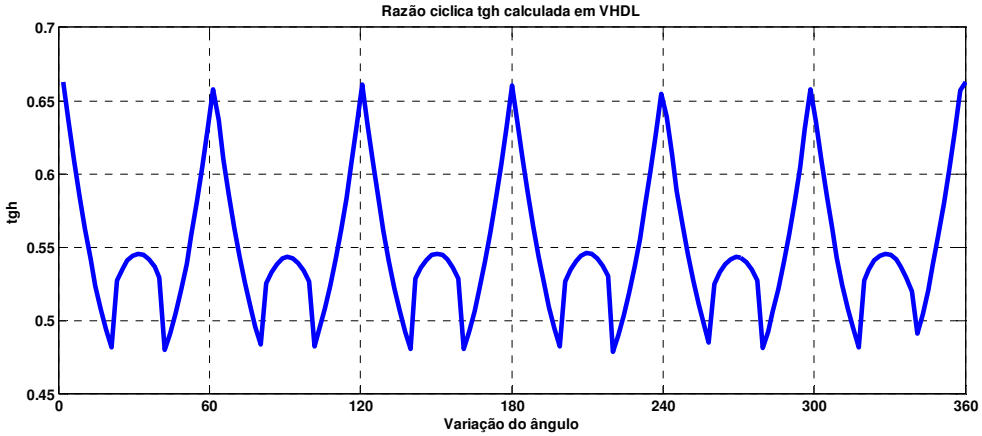


Figura 5.23 – Valor de tgh calculado em VHDL para índice de modulação 0,70.

A Figura 5.24 apresenta o erro de cálculo entre a variável tgh calculada em MatLab® e em VHDL.

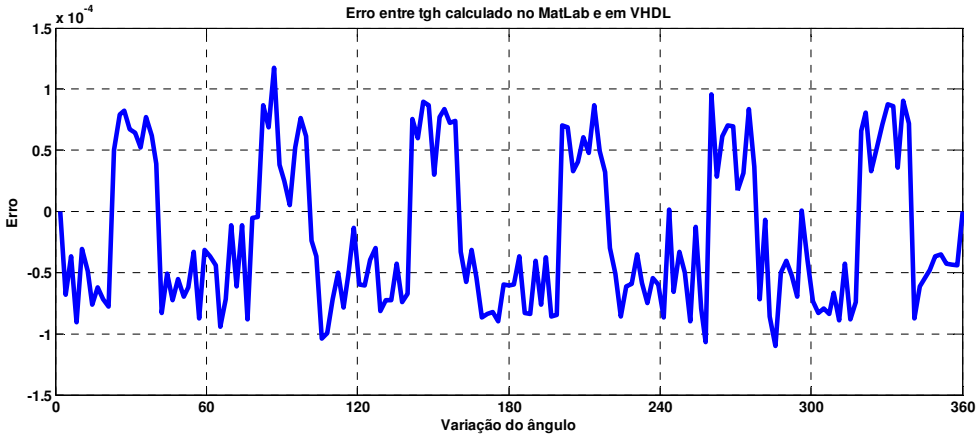


Figura 5.24 – Erro de cálculo entre a variável tgh calculada em MatLab e em VHDL.

A eficiência dos cálculos das razões cíclicas tg, th e tgh feitas em VHDL pode ser observada na Figura 5.18 para tg, na Figura 5.21 para th e na Figura 5.24 para tgh. O erro máximo para tg e th é de, aproximadamente, 6x10e-6 e para tgh é de, aproximadamente, 1x10e-4. Observa-se, pelos erros apresentados, um excelente resultado, uma vez que eles são inexpressivos.

5.2.5 Simulação para o valor do comparador

No desenvolvimento deste projeto faz-se o cálculo para cada par de chaves de cada uma das fases do inversor, totalizando seis cálculos. Apresenta-se aqui apenas o valor do comparador para a chave 1 da fase A para efeito de validação dos cálculos.

A Figura 5.25 apresenta os valores de comparação para gerar o sinal de PWM da chave 1 da fase A, calculados em MatLab®.

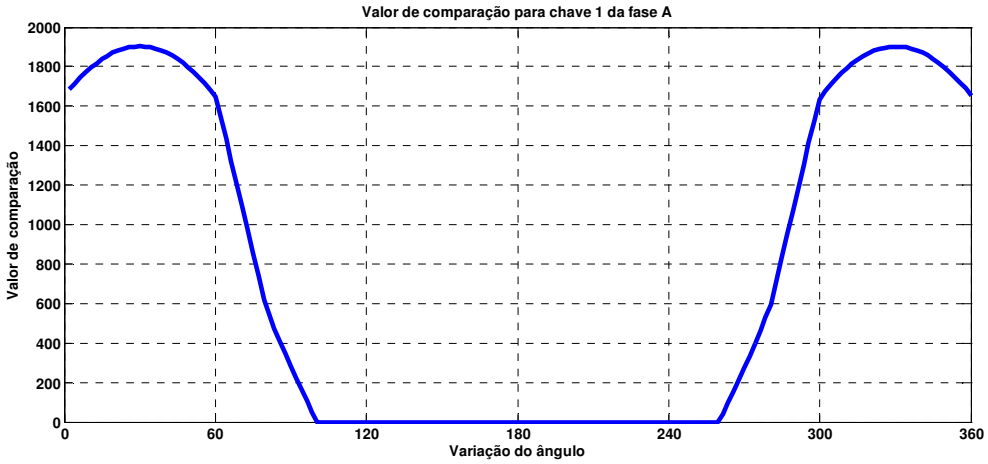


Figura 5.25 – Valor de comparação da chave 1 da fase A calculado em MatLab®.

A Figura 5.26 apresenta os valores de comparação para gerar o sinal de PWM da chave 1 da fase A, calculados em VHDL.

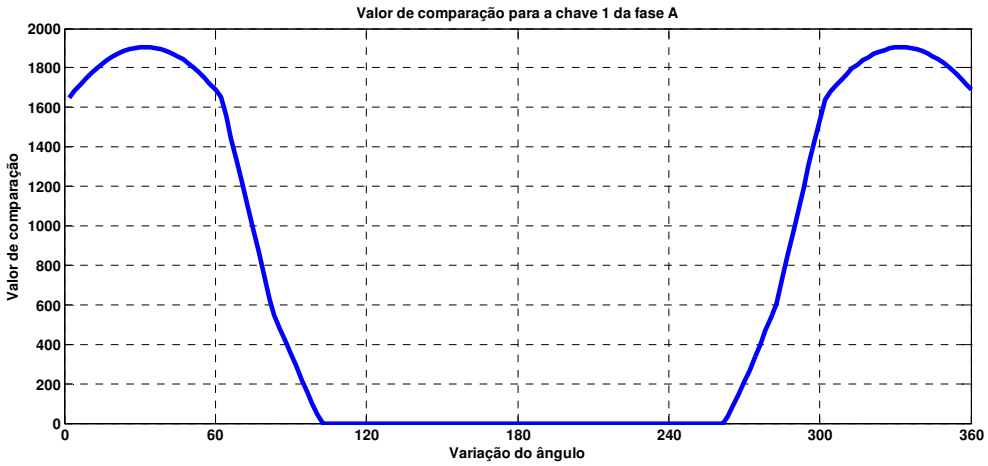


Figura 5.26 – Valor de comparação da chave 1 da fase A calculado em VHDL.

A Figura 5.27 apresenta o erro entre os cálculos do valor do comparador para chave 1 da fase A feito em MatLab e em VHDL.

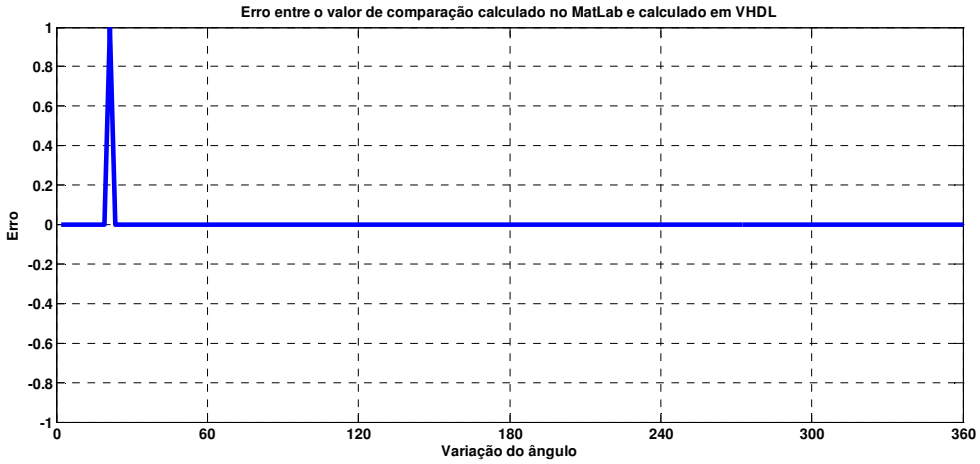


Figura 5.27 – Erro entre os valores do comparador da chave 1 da fase A calculados em MatLab® e em VHDL.

Frente aos bons resultados apresentados anteriormente, a Figura 5.27 apresenta um erro insignificante para o cálculo do valor do comparador da chave 1 da fase A. Nota-se, nessa figura, um excelente resultado, nela apenas um resultado é diferente do cálculo feito em VHDL e do cálculo feito em MatLab®, isso comprova a excelente performance do algoritmo desenvolvido em VHDL e implementado no FPGA.

5.2 **Resultados experimentais com o algoritmo sintetizado no FPGA utilizando o *kit* de desenvolvimento DK-CYCII2C20N.**

Os resultados apresentados a seguir são os resultados obtidos com a simulação do algoritmo sintetizado no FPGA, pelo *kit* de desenvolvimento e do inversor de três níveis. Os resultados apresentados comprovam o correto funcionamento do projeto, uma vez que ele controla o inversor de três níveis e fornece, em sua saída, a tensão modulada de acordo com o índice de modulação escolhido.

O algoritmo da MV foi testado em um inversor com diodo de grampeamento de três níveis fonte de tensão. A validação da implementação do algoritmo em FPGA foi feita por meio de testes com índices de modulação 0,30, 0,48, 0,55, 0,63, 0,70 e 0,90.

As figuras a seguir mostram as formas de onda da tensão na saída do inversor entre as fases A e B. Para cada uma das formas de onda de tensão, segue a análise do espectro harmônico.

A figura 5.28 mostra a visão geral de todo o protótipo implementado em laboratório. Nessa figura são observados a placa de desenvolvimento, o acoplamento óptico, o inversor de três níveis e a carga resistiva.

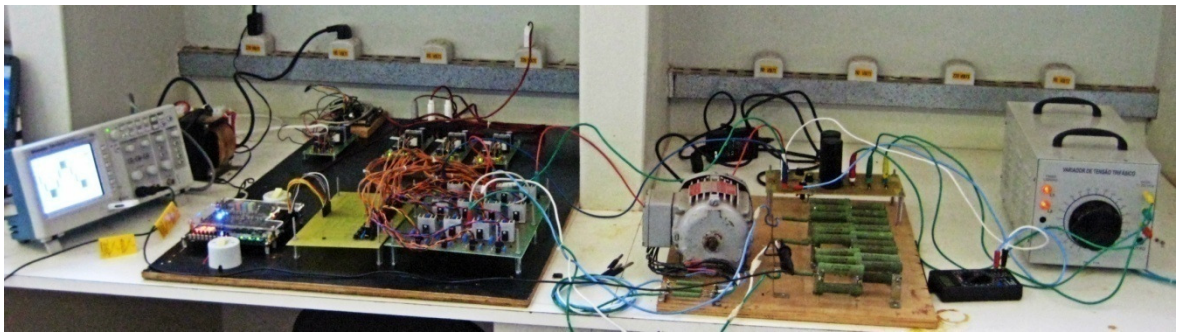


Figura 5.28 – Visão geral do protótipo implementado.

A Figura 5.29 apresenta, com mais detalhes, a placa de desenvolvimento conectada ao acoplamento óptico e ao inversor de três níveis.

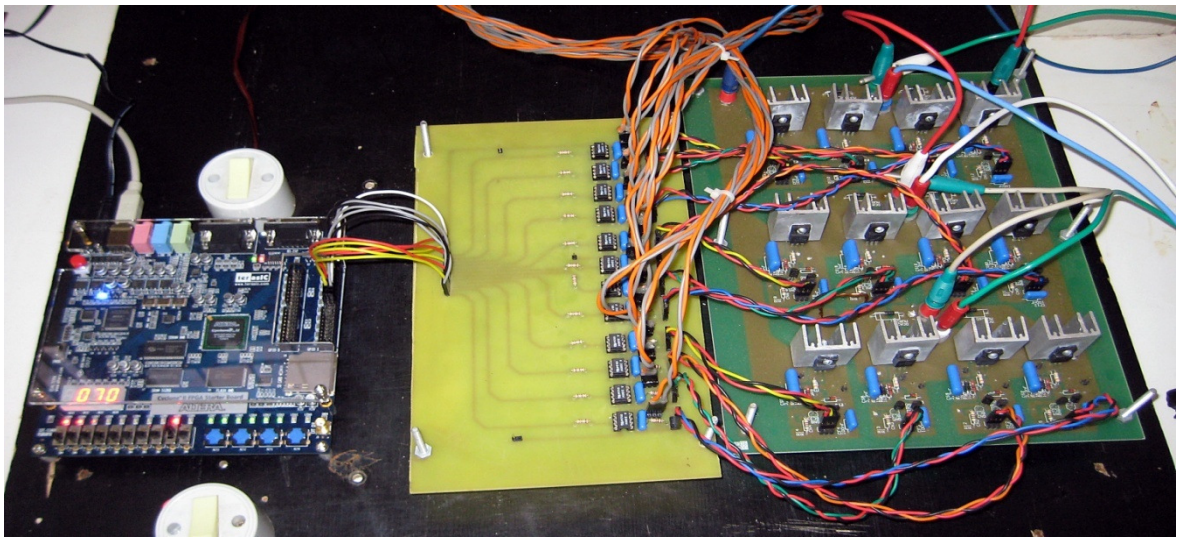


Figura 5.29 – FPGA, Acoplamento Óptico e Inversor de Três Níveis.

5.2.1 Tempo Morto

Na Figura 5.30 são mostrados os sinais entre as chaves complementares com a inserção do tempo morto.

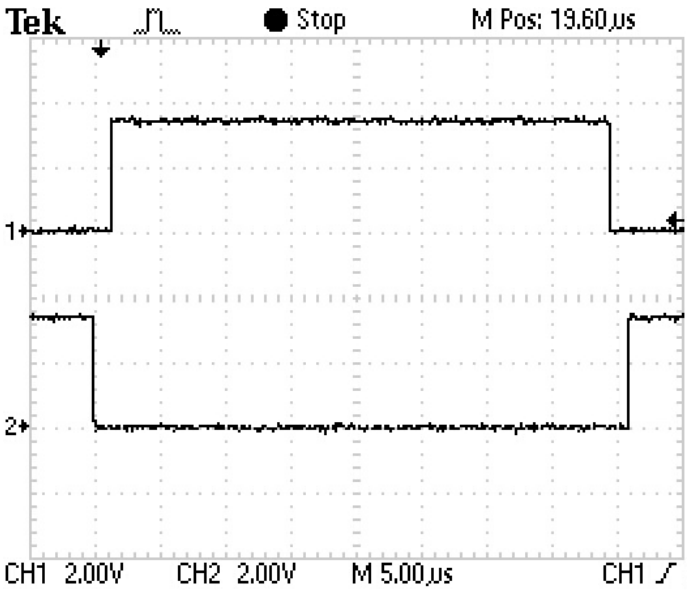


Figura 5.30 - Tempo morto entre as chaves complementares.

Nota-se, na Figura 5.30, que os dois sinais permanecem em um pequeno intervalo de tempo em zero. Esse pequeno intervalo retrata a inserção do tempo morto entre os sinais de controle das chaves complementares do inversor.

5.2.2 Tensão entre fase para índice de modulação 0,30

Na Figura 5.31 é apresentada a forma de onda entre as fases A e B do Inversor de três níveis para índice de modulação 0,30.

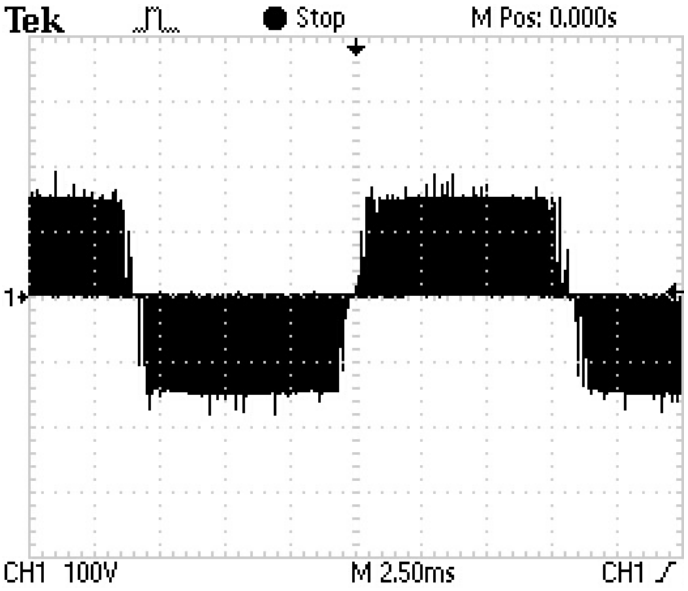


Figura 5.31 – Tensão de linha para índice de modulação 0,30.

Observa-se, na Figura 5.31, a operação do inversor com apenas dois níveis, para um índice de modulação 0,30. São observados na tensão de saída do inversor apenas os níveis de tensões 0, $+V_{dc}/2$ e $-V_{dc}/2$.

Na Figura 5.32 é apresentada a análise do espectro harmônico da tensão de saída do inversor para um índice de modulação 0,30. Observa-se que o conteúdo harmônico na tensão de saída se torna visível a partir da banda harmônica múltipla de duas vezes a frequência de chaveamento do inversor.

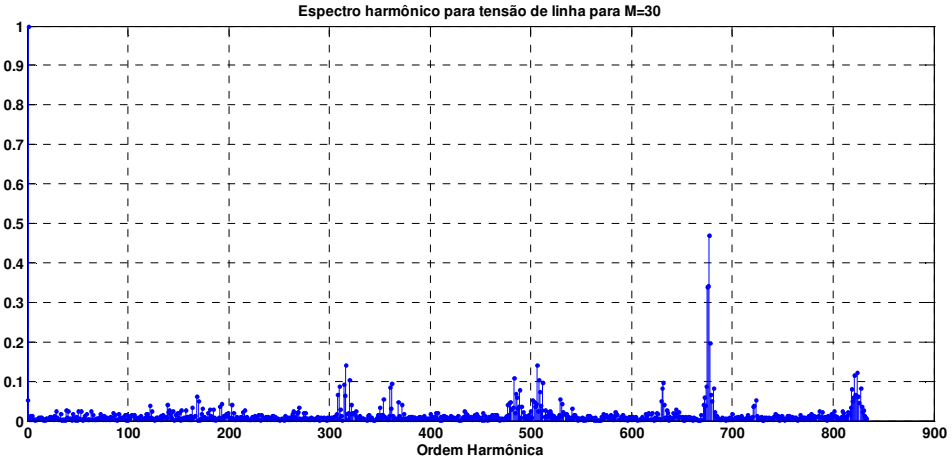


Figura 5.32 – Espectro harmônico para índice de modulação 0,30.

5.2.3 Tensão entre fase para índice de modulação 0,48

Na Figura 5.33, é apresentada a forma de onda entre as fases A e B do Inversor de três níveis para índice de modulação 0,48.

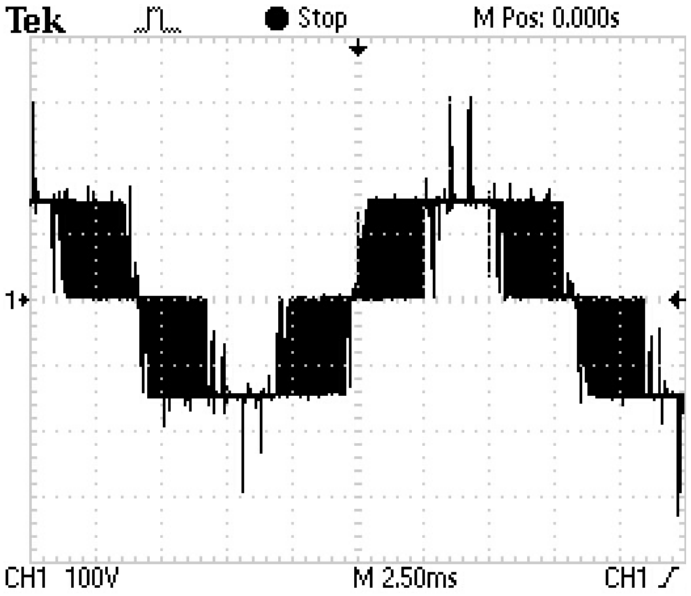


Figura 5.33 – Tensão de linha para índice de modulação 0,48.

Para o índice de modulação 0,48, o inversor apresenta, por um pequeno intervalo de tempo na tensão de saída, a síntese de alguns pontos de tensão V_{dc}, ou seja, o inversor começa a operar com três níveis.

Na Figura 5.34 é apresentada a análise do espectro harmônico da tensão de saída do inversor para um índice de modulação 0,48. Observa-se que o conteúdo harmônico na tensão de saída se torna visível a partir da banda harmônica múltipla de uma vez a frequência de chaveamento do inversor.

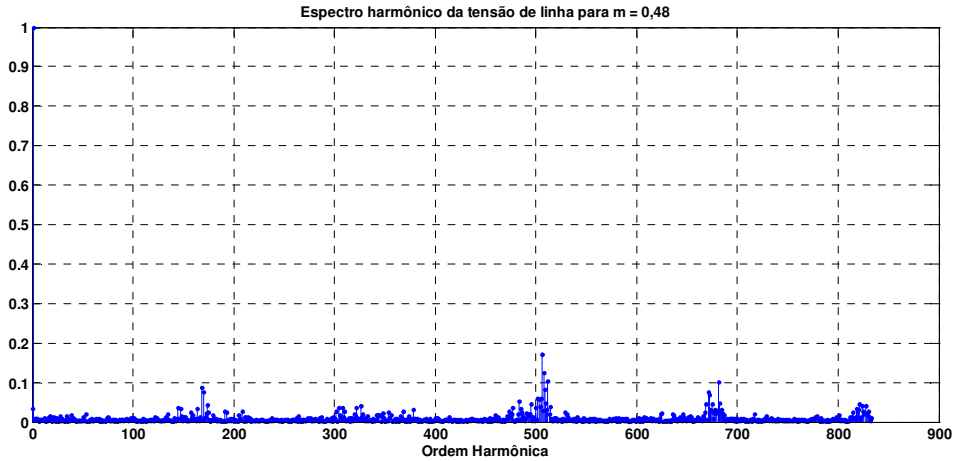


Figura 5.34 – Espectro harmônico para índice de modulação 0,48.

5.2.4 Tensão entre fase para índice de modulação 0,55

Na figura 5.35 é apresentada a forma de onda da saída do inversor para índice de modulação 0,55.

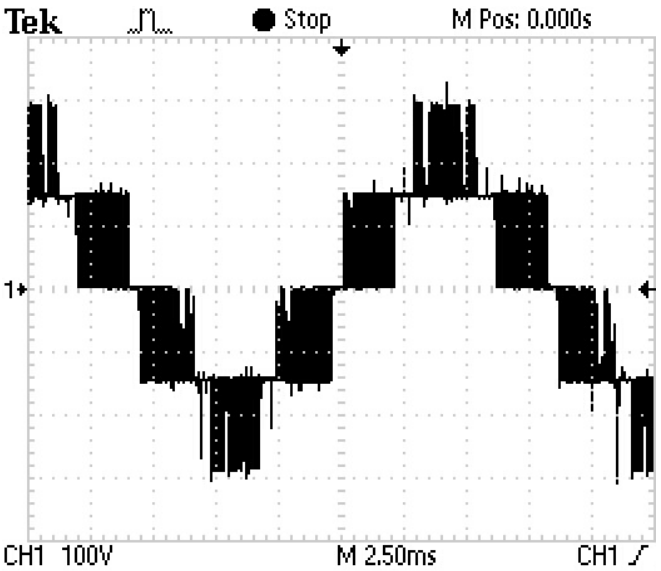


Figura 5.35 – Tensão de linha para índice de modulação 0,55.

Utilizando um índice de modulação igual a 0,55, o inversor trabalha com três níveis, sintetizando, na sua saída, os níveis de tensão 0, + $V_{dc/2}$,-$V_{dc/2}$ e + V_{dc} e -V_{dc}.

A Figura 5.36 apresenta a análise do espectro harmônico da tensão de saída do inversor para índice de modulação 0,55. Observa-se que o conteúdo

harmônico na tensão de saída se torna visível a partir da banda harmônica múltipla de duas vezes a frequência de chaveamento do inversor.

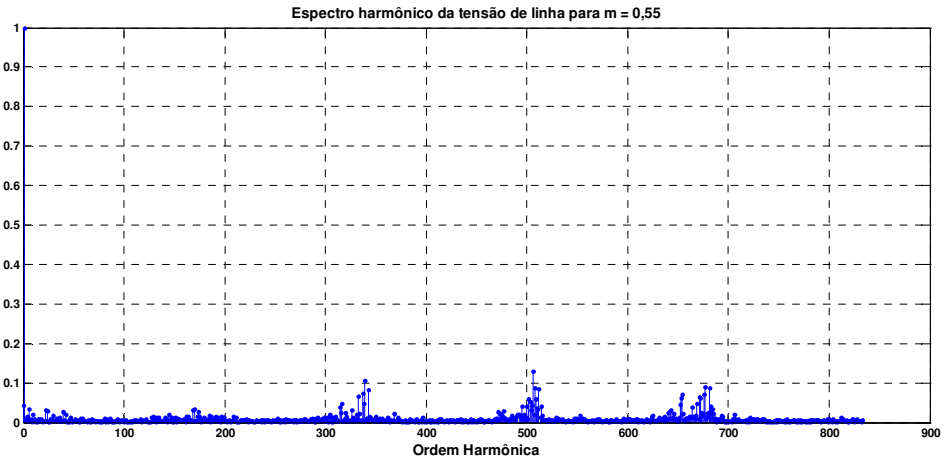


Figura 5.36 – Espectro harmônico para índice de modulação 0,55.

5.2.5 Tensão entre fase para índice de modulação 0,63

Na figura 5.37, é a apresentada a forma de onda da saída do inversor para índice de modulação 0,63.

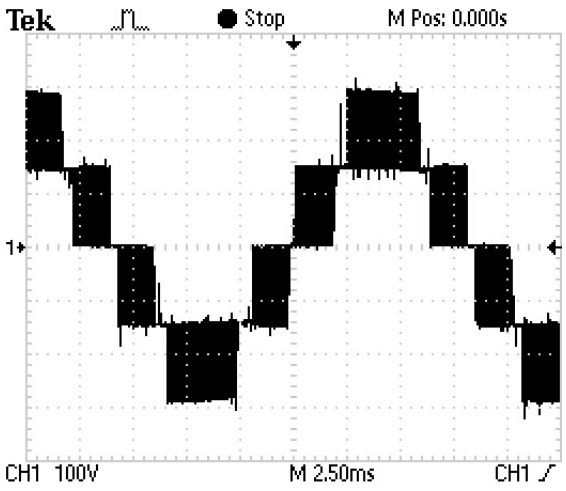


Figura 5.37 – Tensão de linha para índice de modulação 0,63.

Na Figura 5.37 é apresentada a tensão de saída do inversor para índice de modulação 0,63, observa-se que o inversor opera com todos os níveis de tensão disponíveis 0, $+V_{dc}/2$ $-V_{dc}/2$, $+V_{dc}$ e $-V_{dc}$.

A Figura 5.38 apresenta a análise do espectro harmônico da tensão de saída do inversor para índice de modulação 0,63. Observa-se que o conteúdo

harmônico na tensão de saída se torna visível a partir da banda harmônica múltipla de duas e quatro vezes a frequência de chaveamento do inversor.

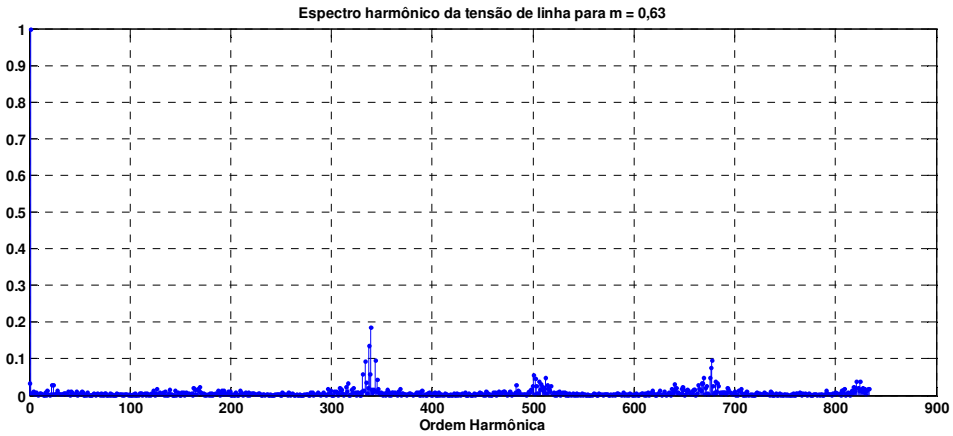


Figura 5.38 – Espectro harmônico para índice de modulação 0,63.

5.2.6 Tensão entre fase para índice de modulação 0,70

Na figura 5.39 é apresentada a forma de onda da saída do inversor para o índice de modulação 0,70.

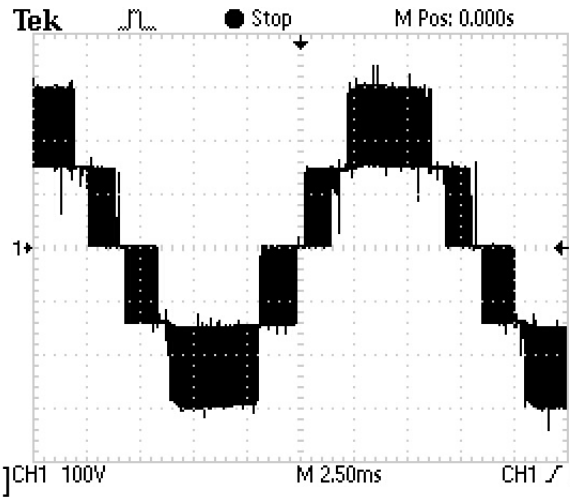


Figura 5.39 – Tensão de linha para índice de modulação 0,70.

A Figura 5.40 apresenta a análise do espectro harmônico da tensão de saída do inversor para índice de modulação 0,70. Observa-se que o conteúdo harmônico na tensão de saída se torna visível a partir da banda harmônica múltipla de duas vezes a frequência de chaveamento do inversor.

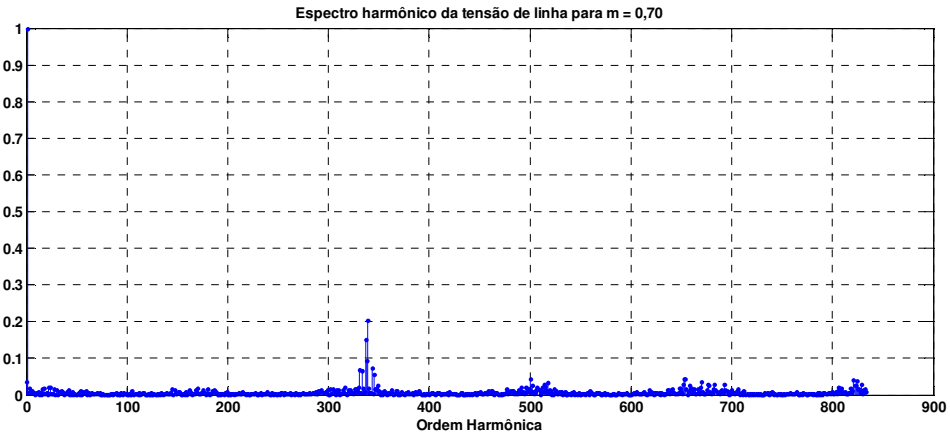


Figura 5.40 – Espectro harmônico para índice de modulação 0,70.

5.2.7 Tensão entre fase para índice de modulação 0,90

Na figura 5.41 é apresentada a forma de onda da saída do inversor para o índice de modulação 0,90.

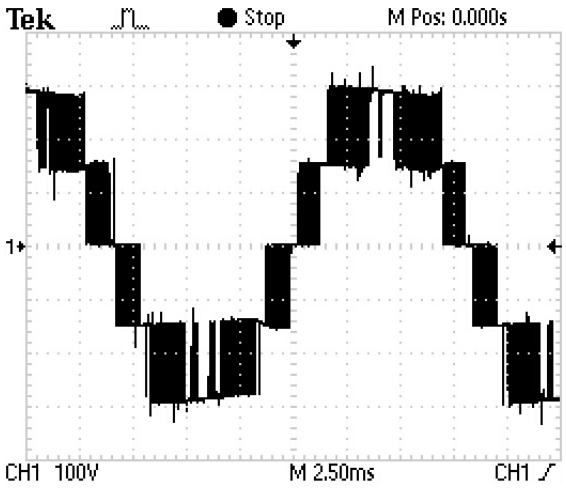


Figura 5.41 – Tensão de linha para índice de modulação 0,90.

A operação do inversor para um índice de modulação 0,90 está na região bem próxima da faixa máxima de operação linear, observa-se que a tensão de saída do inversor permanece muito mais tempo em $+V_{dc}$ e $-V_{dc}$ que os outros índices de modulação.

A Figura 5.42 apresenta a análise do espectro harmônico da tensão de saída do inversor para o índice de modulação 0,90. Observa-se que o conteúdo harmônico na tensão de saída se torna visível a partir da banda harmônica múltipla de uma vez a frequência de chaveamento do inversor.

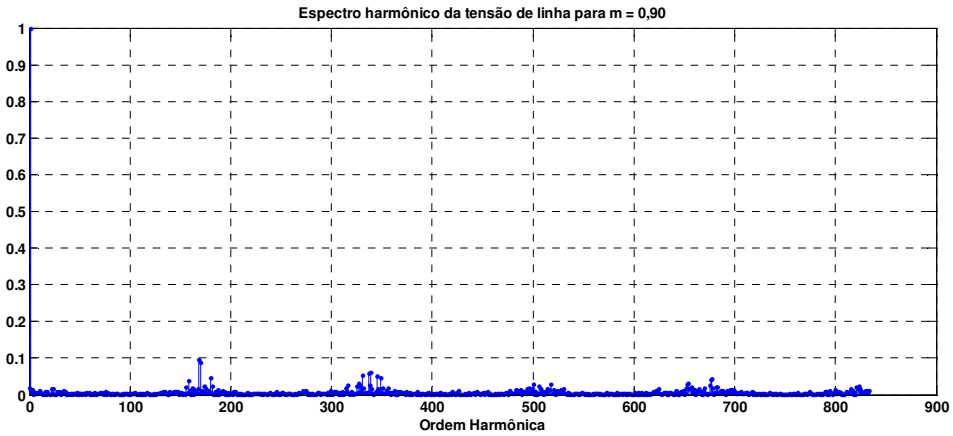


Figura 5.42 – Espectro harmônico para índice de modulação 0,90.

5.3 **Conclusões**

Neste capitulo foram apresentados os resultados em *hardware* e em *software* obtidos com o desenvolvimento deste trabalho.

Para acompanhar o desempenho dos cálculos feitos em VHDL foram feitas comparações de várias etapas do algoritmo e gráficos com o erro absoluto (diferença entre o MatLab e o ModelSim) são apresentados, nestes gráficos, observa-se que o algoritmo da MV sintetizado no FPGA obteve pouquíssimos erros, validando o desenvolvimento do projeto.

Observa-se que a metodologia adotada testando de várias etapas do desenvolvimento do projeto proporcionou um bom desempenho do algoritmo sintetizado no FPGA, uma vez que, a tensão de saída do inversor apresenta o comportamento esperado, de acordo com os índices de modulação e com a análise do espectro harmônico, uma vez que, para esta topologia de inversores multiníveis o espectro harmônico dever ser centrado em duas vezes a freqüência de chaveamento.

O algoritmo utilizado para a inserção do tempo morto funciona corretamente, adicionando um tempo morto entre as chaves complementares do inversor.

6 CONCLUSÕES

Este trabalho apresentou o desenvolvimento do algoritmo da modulação vetorial utilizando coordenadas móveis não-ortogonais em FPGA para um inversor de três níveis diodo de grampeamento. As simplificações propostas para o algoritmo facilitaram o seu desenvolvimento no FPGA e ainda contribuíram para baixa utilização dos recursos internos do FPGA, possibilitando ainda a implementação da geração dos vetores de referência no eixo dq no próprio FPGA.

A utilização da flexibilidade do FPGA para fazer os cálculos do algoritmo proporcionou uma maior precisão nos seus resultados, apresentando uma diferença muito pequena em comparação ao algoritmo desenvolvido no MatLab®.

O uso de duas bases de tempo no desenvolvimento do projeto proporcionou o uso de uma frequência maior para o modulador PWM, isso ocasionou uma melhor precisão na geração dos sinais PWM comparado com o trabalho [15], uma vez que o valor máximo do comparador passa de 991 pontos com o modulador operando a 20 MHz, para 2465 pontos com o modulador operando a 50 MHz.

O desenvolvimento modularizado do algoritmo proporciona uma maior facilidade para a extensão desse desenvolvimento para topologias maiores que três níveis, uma vez que o algoritmo para tais implementações altera apenas alguns blocos do algoritmo já desenvolvido.

Na extensão do desenvolvimento deste trabalho para inversores de ordem superior a três níveis, o tempo computacional do algoritmo permanecerá o mesmo devido à implementação em paralelo dos cálculos dos tempos de

chaveamento de cada par de chaves complementares do inversor, porém, são utilizados mais recursos lógicos do FPGA.

A utilização das ferramentas Quartus II®, MatLab® e ModelSim® auxiliou o desenvolvimento do projeto em relação à codificação do algoritmo em VHDL, verificação e validação na implementação para inversores de três níveis.

Para estudos futuros, sugerem-se os seguintes temas:

a) desenvolver em um ASIC o algoritmo da MV para inversores de três níveis;
b) desenvolver o algoritmo da MV em FPGA para inversores de cinco níveis;
c) otimizar o desenvolvimento do algoritmo buscando o seu desenvolvimento com a menor quantidade possível de células lógicas;
d) incorporar ao FPGA um controle simples em malha fechada Volts/Hertz.

Este trabalho teve como produções bibliográficas os artigos que serão publicados nos seguintes congressos:

a) **IEEE IEMDC 2009 - Publicado.**
   IEEE International Electric Machines and Drives Conference
   Miami – Florida 3 a 6 de Maio.

b) **IMETI 2009 – Sinopse aceita.**
   The 2nd International Multi-Conference on Engineering and Technological Innovation.
   Orlando – Florida 10 a 13 de Junho.

c) **SNCA 2009 – Sinopse aceita.**
Seminário Nacional de Controle e Automação.
Salvador – Bahia 14 a 16 de outubro.

d) **Cobep 2009 – Em fase de avaliação.**
Congresso Brasileiro de Eletrônica de Potência
Bonito – Mato Grosso do Sul 27 de setembro a 2 de outubro

REFERÊNCIAS

[1] J. Rodriguez, J. S. Lai, F. Z. Peng, “Multilevel Inverters: A Survey of Topologies, Controls, and Applications”, *IEEE Trans. Ind. Eletronics.*, vol. 49, pp. 724–738, Agosto. 2002.

[2] J.Rodríguez, S. Bernet, B. Wu, J. O. Pontt e S. Kouro – “Multilevel Voltage-Source-Converter Topologies for Industrial Medium-Voltage Drives”, *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, VOL. 54, NO. 6, Dezembro 2007.

[3] Gupta, A.K.; Khambadkone, A.M., "A Space Vector PWM Scheme for Multilevel Inverters Based on Two-Level Space Vector PWM," *Industrial Electronics, IEEE Transactions on*, vol.53, no.5, pp.1631-1639, Outubro. 2006.

[4] Pereira Filho, N.; Pinto, J. O. P.; Silva, L. E. B. ; Bose, B. K. . “Simplified Space Vector PWM Algorithm for Multilevel Inverters Using Non-Orthogonal Moving Reference Frame.” *The 2008 IEEE Industry Applications Society Annual Meeting, 2008, Edmonton.*

[5] Haibing Hu; Wenxi Yao; Zhengyu Lu, "Design and Implementation of Three-Level Space Vector PWM IP Core for FPGAs," *Power Electronics, IEEE Transactions on* , vol.22, no.6, pp.2234-2244, Novembro. 2007.

[6] Lopez, O.; Alvarez, J.; Doval-Gandoy, J.; Freijedo, F.; Nogueiras, A.; Penalver, C.M., "Multilevel multiphase space vector PWM algorithm applied to three-phase converters," *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE* , vol., no., pp.3290-3295, 10-13 Novembro. 2008.

[7] Baker, Richard H. Patente: Bridge converter circuit. Diponível em: < http://www.freepatentsonline.com/4270163.html> Acesso em: 03 jun. 2009.

[8] Akira Nabae, Isao Takahashi, Hirofumi Akagi “A New Neutral-Point Clamped PWM Inverter”, *IEEE Transactions on Industry Aplications.* Vol. IA-17, No.5, Setembro 1981, pp. 518-523.

[9] M. Marchesoni, M. Mazzucchelli, S. Tenconi, “A non conventional power converter for plasma stabilization,” *Proc. Power Electron. Spec.Conf.*, 1988, pp. 122–129.

[10] Ying-Yu Tzou; Hau-Jean Hsu; Tien-Sung Kuo, "FPGA-based SVPWM control IC for 3-phase PWM inverters," Industrial Electronics, Control, and Instrumentation, 1996., Proceedings of the 1996 IEEE IECON 22nd International Conference on , vol.1, no., pp.138-143 vol.1, 5-10 Agosto 1996.

[11] Lopez, O.; Alvarez, J.; Doval-Gandoy, J.; Freijedo, F.D.; Nogueiras, A.; Lago, A.; Penalver, C.M., “Comparison of the FPGA Implementation of Two Multilevel Space Vector PWM Algorithms,” *Industrial Electronics, IEEE Transactions on* , vol.55, no.4, pp.1537-1547, Abril 2008.

[12] Yu, Zhenyu, “Space-vector PWM with TMS320C24x/F24x using hardware and software determined switching patterns”, Application Report Texas Instrument, Março 1999.

[13] N. Pereira Filho, J. O. P. Pinto, B. K. Bose, L. E. B. Silva – “A Simple and Ultra-Fast DSP-Based Space Vector PWM Algorithm and its Implementation on a Two-Level Inverter Covering Undermodulation and Overmodulation” In: IECON 2004 - *The 30th Annual Conference of IEEE Industrial Electronics Society*, 2004, Busan, Coréia.

[14] N. Celanovic e D. Boroyevich, “A Fast Space Vector Modulation Algorithm for Multilevel Three-phase Converters”; *IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS*, VOL. 37, NO. 2, Março 2001.

[15] N. Pereira Filho – “Técnicas de modulação vetorial por largura de pulso para inversores fonte de tensão” – Tese de doutorado apresentada à UNIFEI, Minas Gerais, 2007.

[16] N. Pereira Filho, J. O. P. Pinto, B. K. Bose, L. E. B. Silva “A Neural-Network-Based Space Vector PWM of a Five-Level Voltage-Fed Inverter”, IAS, 2004.

[17] Zhaoyong Zhou; Guijie Yang; Tiecai Li, "Design and implementation of an FPGA-based 3-phase sinusoidal PWM VVVF controller," *Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE* , vol.3, no., pp. 1703-1708 Vol.3, 2004.

[18] Cyclone II Device Handbook, Volume 1, Altera. Disponível em: <http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf> Acesso em: 03 jun. 2009.

[19] Quartus II Handbook Version 9.0 Volume 1: Design and Synthesis. Disponível em:< http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf> Acesso em: 03 jun. 2009.

Anexo A – Geração do Vetor Referência no MatLab®

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Mestrado em Engenharia Elétrica UFMS              %%
%% Edvaldo F. Freitas Lima                           %%
%% Algoritmo para geração dos vetores de referência  %%
%% 20/11/2007                                        %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear;

load tbl_sen;

theta    =  0 %% O ângulo começa em zero
Watual   =  376.9911; %% 2*pi*60
T_Sample =  9.9100e-005; %% 1/10090.81

delta = Watual * T_Sample; %% valor da variação do ângulo
fator_index = (450/(pi/2));
i=1;

while theta < 2*pi
    thetam(i,1) = delta + theta   %% ângulo mais a sua variação
    theta = thetam(i,1);

        if ( theta > 0 && theta <= pi/2)
            quadrante = 1;
            alfa = theta;

            ss = 1;
            sc = 1;

        elseif (theta > pi/2 && theta <= pi)
            quadrante = 2;
            alfa = pi - theta;

            ss =  1;
            sc = -1;

        elseif (theta > pi && theta <= (3*pi)/2)
            quadrante = 3;
            alfa = (3*pi/2) - theta;

            ss = -1;
            sc = -1;

        elseif (theta > (3*pi)/2 && theta <= 2*pi)
            quadrante = 4;
            alfa = 2*pi - theta;

            ss = -1;
            sc =  1;
        end
        mat_erro(i,1) = quadrante;
        mat_erro(i,2) = ss;
        mat_erro(i,3) = sc;

%%%Encontrando o valor do Indice para o Seno

        indexm(i,1) = (alfa * fator_index)
        index = indexm(i,1);
        mat_erro(i,4) = index;

       if (index < 1)
            index = 1
       end

        index_int  = floor(index)
        index_frac = index-index_int;

        v1_s = tbl_sen(index_int);
        v2_s = tbl_sen(index_int + 1);

        Vsen(i,1) = (((v2_s - v1_s)*index_frac) + v1_s)* ss;

%%%Encontrando o valor do indice para o Coseno
```

```
        index_cos = 451-index_int
        mat_erro(i,5) = index_cos;
        mat_erro(i,6) = theta;

        v1_c = tbl_sen(index_cos)

        if(index_cos == 1)
             index_cos = 2
        end
        v2_c = tbl_sen(index_cos - 1);
        Vcos(i,1) = (((v2_c - v1_c)*index_frac) + v1_c)* sc;
        i=i+1;

end
```

Anexo B – Desenvolvimento do Algoritomo da Modulação Vetorial no MatLab®.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%           INVERSOR MULTINIVEL - 3 NIVEIS                                  %%
%%                                                                           %%
%%        Mestrado em Engenharia Elétrica - UFMS                             %%
%%        Edvaldo F. Freitas Lima                                            %%
%%     Algoritmo para geração dos valores dos comparadores                   %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear

load ks1ag;
load ks1ah;
load ks1agh;
load ks2ag;
load ks2ah;
load ks2agh;
%%load VdVqVHDL
load VdVq

%%%%Matriz de Normalização
A11 = [   3           3           0          -3          -3            0      ];
A12 = [-sqrt(3)    sqrt(3)     2*sqrt(3)   sqrt(3)    -sqrt(3)     -2*sqrt(3) ];
A21 = [  0           -3          -3           0           3            3      ];
A22 = [2*sqrt(3)   sqrt(3)     -sqrt(3)   -2*sqrt(3)  -sqrt(3)       sqrt(3)  ];


sext = [ 2 6 1 4 3 5 ]; %%Numero do sextante indexado pela variavel NS
Ts =2465;

%%%Cálculo dos valores dos Comparadores%%%%

for i=1:170

%%%%Identificação do Sextante
indice = i
A = sign ( -Vq(i,1) )

% B = sign ( -sqrt(3) * Vd(i,1) + Vq(i,1) );
% C = sign ( sqrt(3)* Vd(i,1) + Vq(i,1) );

produto = sqrt(3) * Vd(i,1);
B = sign (Vq(i,1) - produto)
C = sign (Vq(i,1) + produto)

if A > 0
    A = 0
else
    A = 1
end;

if B > 0
    B = 0
else
    B = 1
end;

if C > 0
    C = 0
else
    C = 1
end;
%%Número do Sextante
Ns = A + 2*B + 4*C
Nsm(i,1) = Ns;

%%%Normalizaçao de Vd e Vq
sextante = sext(Ns)
sextm(i,1) = sextante;

aux_sext = sextante / 2;
aux_int = floor(aux_sext);
ip = aux_sext - aux_int;

%%Normalização
Vg=( A11(sextante)*Vd(i,1) + A12(sextante)*Vq(i,1) );
```

```
Vh=( A21(sextante)*Vd(i,1) + A22(sextante)*Vq(i,1) );
Vgm(i,1) = Vg;
Vhm(i,1) = Vh;

%%%Identificação do triângulo
%%Parte Inteira
Vgu = floor (Vg);
Vhu = floor (Vh);

%%Parte Fracionária
Vgf = Vg - Vgu;
Vhf = Vh - Vhu;

%%Modo de Operação
Md = floor (Vg + Vh)
Mdm(i,1) = Md;

%%Localização do triangulo
Ls = Md^2 + Md + 1 + Vhu - Vgu
Lsm(i,1)=Ls;

%%%%Cálculo das Razões Cíclicas
%%Definir tipo de triângulo

type = Ls + Md
type_f = mod(type,2)

if type_f == 0
    triang_type = 1
else
    triang_type = 0
end

%%Razões Cíclicas

tg  = abs (triang_type - Vgf);
th  = abs (triang_type - Vhf);
tgh = 1 - tg - th;

tgm  (i,1) = tg;
thm  (i,1) = th;
tghm (i,1) = tgh;

%%Número do triângulo dentro do Hexagono (Lh)
Lh = ((sext(Ns))-1)*4+Ls
Lhm(i,1) = Lh;


%%Constantes para comparar com a onda triangular simetrica

%%      ::Fase A::
%%Chave 1 e sua complementar
temp_tso1a = ks1ag(Lh,1)*tg + ks1ah(Lh,1)*th + ks1agh(Lh,1)*tgh;
Tso1A(i,1)= temp_tso1a * Ts;

ksgm(i,1)=ks1ag(Lh,1);
kshm(i,1)=ks1ah(Lh,1);
ksghm(i,1)=ks1agh(Lh,1);

%%Chave 2 e sua complementar

temp_ts02a = ks2ag(Lh,1)*tg + ks2ah(Lh,1)*th + ks2agh(Lh,1)*tgh;
Tso2A(i,1) = temp_ts02a*Ts;

%% Utilizando apenas a parte inteira para comparar.
if(ip == 0)
   Tso1A(i,1) = Ts - floor(Tso1A(i,1));
   Tso2A(i,1) = Ts - floor(Tso2A(i,1));
else
   Tso1A(i,1) = floor(Tso1A(i,1));
   Tso2A(i,1) = floor(Tso2A(i,1));
end;

end; %%%final do for
```

Anexo C – Verificação do Projeto Utilizando a Ferramenta ModelSim®

```
--VERIFICAÇÃO DO PROJETO UTILIZANDO A FERRAMENTA MODELSIM®
--GERAR AS MATRIZES PARA SEREM CONFERIDAS COM O MATLAB®

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
Use ieee.math_real.all;
Use ieee.std_logic_textio.all;
Use std.textio.all;

---Essas são as saidas do meu teste
Entity tGeraVdVq32 is
    PORT (
                            index   :  out  integer range 0 to 200
         );
end;

architecture only of tGeraVdVq32 is


----BLOCO DE GERAÇÃO DOS VETORES DE REFERENCIA E TEMPOS DAS CHAVES
COMPONENT GeraVdVq32
      PORT (
         index_mod  :  in   signed (15 downto 0); -- valor do indice de modulação
                       rst        :  in   std_logic;
                       start      :  in   std_logic;
                       fim_calc   :  in   std_logic;
                       clk        :  in   std_logic;
                       index      :  out  integer range 0 to 200; -- iteração do algoritmo
                       calc       :  out  std_logic;
                       Lh         :  out  integer range 0 to 45;
                       Sextante   :  out  integer range 0 to 7;
                       tg         :  out  signed (15 downto 0); -- Q2.14
                       th         :  out  signed (15 downto 0); -- Q2.14
                       tgh        :  out  signed (15 downto 0); -- Q2.14
                       Vd         :  out  signed (15 downto 0); -- Q1.15
                       Vq         :  out  signed (15 downto 0); -- Q1.15
                       Vg         :  out  signed (15 downto 0); -- Q2.14
                       Vh         :  out  signed (15 downto 0); -- Q2.14

                       freq       :  out  std_logic; -- geração da onda quadrada;
                       err_time   :  out  std_logic; -- erro nos cálculos de tg th e tgh
                       sext_ip    :  out  std_logic  -- sextante é impar ou par


        );
END COMPONENT ;


--TEMPO DE CHAVEAMENTO PARA CADA CHAVE
COMPONENT time_switch1a
    PORT(
              Ts     :     in  std_logic_vector (15 downto 0); -- Valor do TS
              timer  :     in  std_logic; -- Habilitar bloco apos os calculos
              Lh     :     in  integer range 0 to 100; -- Inteiro
              tg     :     in  signed  (15 downto 0);  -- Q11.5
              th     :     in  signed  (15 downto 0);  -- Q11.5
              tgh    :     in  signed  (15 downto 0);  -- Q11.5
              clk    :     in  std_logic; -- Sinal de clock
             comparador :  out std_logic;  -- Habilitar bloco para gerar PWM
              TSO1A      :  out std_logic_vector (15 downto 0)
         );
     END COMPONENT ;


COMPONENT time_switch2a
    PORT(
            Ts     :     in  std_logic_vector (15 downto 0); -- Valor do TS13.3
            timer  :     in std_logic; -- Habilitar bloco apos os calculos
            Lh     :     in  integer range 0 to 100; -- Inteiro
            tg     :     in  signed (15 downto 0);  -- Q11.5
            th     :     in  signed (15 downto 0);  -- Q11.5
            tgh    :     in  signed (15 downto 0);  -- Q11.5
            clk    :     in  std_logic; -- Sinal de clock
```

```
              comparador :  out std_logic;  -- Habilitar bloco para gerar PWM
              TSO2A      :  out std_logic_vector (15 downto 0)
          );
     END COMPONENT ;


COMPONENT time_switch1b
    PORT(
           Ts     :     in  std_logic_vector (15 downto 0); Q13.3
           timer  :     in std_logic; -- Habilitar bloco apos os calculos
           Lh     :     in  integer range 0 to 100; -- Inteiro
           tg     :     in  signed (15 downto 0);  -- Q11.5
           th     :     in  signed (15 downto 0);  -- Q11.5
           tgh    :     in  signed (15 downto 0);  -- Q11.5
           clk    :     in  std_logic; -- Sinal de clock
           comparador :  out std_logic;  -- Habilitar bloco para gerar PWM
           TSO1B      :  out std_logic_vector (15 downto 0) --
          );
     END COMPONENT ;


COMPONENT time_switch2b
    PORT(
           Ts     :     in  std_logic_vector (15 downto 0); -- Valor do TSQ13.3
           timer  :     in  std_logic; -- Habilitar bloco apos os calculos
           Lh     :     in  integer range 0 to 100; -- Inteiro
           tg     :     in  signed (15 downto 0);  -- Q11.5
           th     :     in  signed (15 downto 0);  -- Q11.5
           tgh    :     in  signed (15 downto 0);  -- Q11.5
           clk    :     in  std_logic; -- Sinal de clock
           comparador  :  out std_logic;  -- Habilitar bloco para gerar PWM
           TSO2B       :  out std_logic_vector (15 downto 0) -- Valor do Comparador
          );
     END COMPONENT ;


 COMPONENT time_switch1c
    PORT(
           Ts     :     in  std_logic_vector (15 downto 0); -- Valor do TS,
           timer  :     in  std_logic; -- Habilitar bloco apos os calculos
           Lh     :     in  integer range 0 to 100; -- Inteiro
           tg     :     in  signed (15 downto 0);  -- Q11.5
           th     :     in  signed (15 downto 0);  -- Q11.5
           tgh    :     in  signed (15 downto 0);  -- Q11.5
           clk    :     in  std_logic; -- Sinal de clock
           comparador  :  out std_logic;  -- Habilitar bloco para gerar PWM
           TSO1C       :  out std_logic_vector (15 downto 0) -- Valor do Comparador
          );
     END COMPONENT ;


COMPONENT time_switch2c
    PORT(
           Ts     :     in  std_logic_vector (15 downto 0); -- Valor do TS Q13.3
           timer  :     in  std_logic; -- Habilitar bloco apos os calculos
           Lh     :     in  integer range 0 to 100; -- Inteiro
           tg     :     in  signed (15 downto 0);  -- Q11.5
           th     :     in  signed (15 downto 0);  -- Q11.5
           tgh    :     in  signed (15 downto 0);  -- Q11.5
           clk    :     in  std_logic; -- Sinal de clock
           comparador :  out std_logic;  -- Habilitar bloco para gerar PWM
           TSO2C      :  out std_logic_vector (15 downto 0) -- Valor do Comparador
          );
     END COMPONENT ;

--CONTROLE DO PWM E DE TODA A ESTRUTURA DO ALGORITMO
COMPONENT ctrl_pwm
   PORT (
         max_delay    :    in   std_logic_vector (15 downto 0); -- valor do tempo morto
         cont_max     :    in   std_logic_vector (15 downto 0); -- valor maximo do contador
         comparador   :    in   std_logic; -- sinal de entrado do bloco comparador
         TSO1A        :    in   std_logic_vector (15 downto 0); -- valor de Comparação
         clk          :    in   std_logic; -- sinal do clock
         rst          :    in   std_logic; -- sinal do reset
```

```
        up_dw        :    out  std_logic; -- sinal de subida e descida do contador
        pwm_dead     :    out  std_logic; -- Sinal de PWM com tempo Morto
        pwm_dead_not :    out  std_logic; -- Sinal complementar do PWM com tempo Morto
        new_calc     :    out  std_logic  -- sinal para habilitar um novo cálculo
    );

END COMPONENT;


--- BLOCO PARA SIMPLES CONFIGURAÇÃO DO SISTEMA
COMPONENT config
   PORT (
        clk          : in   std_logic;
        max_delay    : out  std_logic_vector (15 downto 0); -- Valor do tempo morto
        cont_max     : out  std_logic_vector (15 downto 0); -- Maximo valor do contador
        index_mod    : out  std_logic_vector (15 downto 0); --
        Ts           : out  std_logic_vector (15 downto 0)  -- Q11,5
    );

END COMPONENT;

--Sinais do GeraVdVq
SIGNAL     clk        :     std_logic :='0';
SIGNAL     rst        :     std_logic :='0';
SIGNAL     start      :     std_logic :='0';
SIGNAL     index_mod  :     signed (15 downto 0); -- valor do indice de modulação
SIGNAL     index_for  :     std_logic_vector (15 downto 0); -- valor do indice de modulação
SIGNAL     Ts         :     std_logic_vector (15 downto 0); -- Valor do TS
SIGNAL     fim_calc   :     std_logic;
SIGNAL     index      :     integer range 0 to 200; -- iteração do algoritmo
SIGNAL     calc       :     std_logic;
SIGNAL     Lh         :     integer range 0 to 45;
SIGNAL     tg         :     signed (15 downto 0); --Q11.5
SIGNAL     th         :     signed (15 downto 0); --Q11.5
SIGNAL     tgh        :     signed (15 downto 0); --Q11.5
SIGNAL     freq       :     std_logic; -- geração da onda quadrada;
SIGNAL     err_time   :     std_logic; -- erro nos cálculos de tg th e tgh
SIGNAL     Sextante   :     integer range 0 to 7;
SIGNAL     Vd         :     signed (15 downto 0); -- Q1.15
SIGNAL     Vq         :     signed (15 downto 0); -- Q1.15
SIGNAL     Vg         :     signed (15 downto 0); -- Q2.14
SIGNAL     Vh         :     signed (15 downto 0); -- Q2.14
SIGNAL     sext_ip    :     std_logic; -- determina se o sextante é impar ou par


--Sinais do TimeSwitch

SIGNAL     timer      :   std_logic; -- Habilitar bloco apos os calculos
SIGNAL     TSO1A      :   std_logic_vector (15 downto 0); -- Chave 1 da fase A
SIGNAL     TSO2A      :   std_logic_vector (15 downto 0); --
SIGNAL     TSO1B      :   std_logic_vector (15 downto 0);
SIGNAL     TSO2B      :   std_logic_vector (15 downto 0);
SIGNAL     TSO1C      :   std_logic_vector (15 downto 0);
SIGNAL     TSO2C      :   std_logic_vector (15 downto 0);
SIGNAL     comparador1a :   std_logic;  -- Habilitar bloco para gerar PWM
SIGNAL     comparador2a :   std_logic;  -- Habilitar bloco para gerar PWM
SIGNAL     comparador1b :   std_logic;  -- Habilitar bloco para gerar PWM
SIGNAL     comparador2b :   std_logic;  -- Habilitar bloco para gerar PWM
SIGNAL     comparador1c :   std_logic;  -- Habilitar bloco para gerar PWM
SIGNAL     comparador2c :   std_logic;  -- Habilitar bloco para gerar PWM

--Sinais do Controle do PWM

SIGNAL         max_delay    :   std_logic_vector (15 downto 0); -- valor do tempo morto
SIGNAL         cont_max     :   std_logic_vector (15 downto 0); -- valor maximo do contador
SIGNAL         cont         :   std_logic_vector (15 downto 0); -- valor do contador
SIGNAL         new_calc     :   std_logic; -- sinal para habilitar um novo cálculo
SIGNAL         up_dw        :   std_logic; -- sinal de subida e descida do contador
SIGNAL         pwm_dead     :   std_logic; -- Sinal de PWM com tempo Morto
SIGNAL         pwm_dead_not :   std_logic; -- Sinal complementar do PWM com tempo Morto


begin
```

```
dut : GeraVdVq32
   PORT MAP (
                            clk       => clk,
                            rst       => rst,
                            start     => start,
                            index_mod => index_mod,
                            fim_calc  => new_calc,
                            calc      => calc,
                            index     => index,
                            Lh        => Lh,
                            tg        => tg,
                            th        => th,
                            tgh       => tgh,
                            err_time  => err_time,
                            freq      => freq,
                            Sextante  => Sextante,
                            Vd        => Vd,
                            Vq        => Vq,
                            Vg        => Vg,
                            Vh        => Vh,
                            sext_ip   => sext_ip
           );


dut3 : ctrl_pwm
   PORT MAP (
              TSO1A        =>   TSO1A,
              clk          =>   clk,
              rst          =>   rst,
              comparador   =>   comparador1a,
              cont_max     =>   cont_max,
              max_delay    =>   max_delay,
              new_calc     =>   new_calc,
              up_dw        =>   up_dw,
              pwm_dead     =>   pwm_dead,
              pwm_dead_not =>   pwm_dead_not
           );

dut4 : config
   PORT MAP (
               clk          =>   clk,
               index_mod    =>   index_for,
               max_delay    =>   max_delay,
               cont_max     =>   cont_max,
               Ts           =>   Ts
          );

dut2 : time_switch1a
   PORT MAP (
                Ts         => Ts,
                tg         => tg,
                th         => th,
                tgh        => tgh,
                Lh         => Lh,
                timer      => calc,
                clk        => clk,
                TSO1A      => TSO1A,
                comparador => comparador1a
           );

dut5 : time_switch2a
   PORT MAP (
                Ts         => Ts,
                tg         => tg,
                th         => th,
                tgh        => tgh,
                Lh         => Lh,
                timer      => calc,
                clk        => clk,
                TSO2A      => TSO2A,
                comparador => comparador2a
           );

dut6 : time_switch1b
   PORT MAP (
```

```
                Ts          => Ts,
                tg          => tg,
                th          => th,
                tgh         => tgh,
                Lh          => Lh,
                timer       => calc,
                clk         => clk,
                TSO1B       => TSO1B,
                comparador => comparador1b
            );


dut7 : time_switch2b
   PORT MAP (
                Ts          => Ts,
                tg          => tg,
                th          => th,
                tgh         => tgh,
                Lh          => Lh,
                timer       => calc,
                clk         => clk,
                TSO2B       => TSO2B,
                comparador => comparador2b
            );


dut8 : time_switch1c
   PORT MAP (
                Ts          => Ts,
                tg          => tg,
                th          => th,
                tgh         => tgh,
                Lh          => Lh,
                timer       => calc,
                clk         => clk,
                TSO1C       => TSO1C,
                comparador => comparador1c
            );


dut9 : time_switch2c
   PORT MAP (
                Ts          => Ts,
                tg          => tg,
                th          => th,
                tgh         => tgh,
                Lh          => Lh,
                timer       => calc,
                clk         => clk,
                TSO2C       => TSO2C,
                comparador => comparador2c
            );

clock : PROCESS
   begin

     wait for 10 ns;
     clk  <= not clk;

   end PROCESS clock;

stimulus : PROCESS
   begin

              rst   <= '1';
              start <= '0';
              index_mod <= "0010110011001101"; --0.70
              wait for 4 ns;
              rst   <= '0';
              wait for 6 ns;
              start <= '1';
              wait;

   end PROCESS stimulus;
```

```
--Salvando os valores no arquivo texto
file_io: PROCESS IS
   file     out_file1  : text OPEN WRITE_MODE IS "tso1a_values.m";
   file     out_file2  : text OPEN WRITE_MODE IS "tso2a_values.m";
   file     out_file3  : text OPEN WRITE_MODE IS "tso1b_values.m";
   file     out_file4  : text OPEN WRITE_MODE IS "tso2b_values.m";
   file     out_file5  : text OPEN WRITE_MODE IS "tso1c_values.m";
   file     out_file6  : text OPEN WRITE_MODE IS "tso2c_values.m";
   file     out_file7  : text OPEN WRITE_MODE IS "vd_values.m";
   file     out_file8  : text OPEN WRITE_MODE IS "vq_values.m";
   file     out_file9  : text OPEN WRITE_MODE IS "vg_values.m";
   file     out_file10 : text OPEN WRITE_MODE IS "vh_values.m";
   file     out_file11 : text OPEN WRITE_MODE IS "lh_values.m";
   file     out_file12 : text OPEN WRITE_MODE IS "sext_values.m";
   file     out_file13 : text OPEN WRITE_MODE IS "tg_values.m";
   file     out_file14 : text OPEN WRITE_MODE IS "th_values.m";
   file     out_file15 : text OPEN WRITE_MODE IS "tgh_values.m";
   file     out_file16 : text OPEN WRITE_MODE IS "sext_ip.m";

   variable out_line1  : line; -- tso1a
   variable out_line2  : line; -- tso2a
   variable out_line3  : line; -- tso1b
   variable out_line4  : line; -- tso2b
   variable out_line5  : line; -- tso1c
   variable out_line6  : line; -- tso2c
   variable out_line7  : line; -- vd
   variable out_line8  : line; -- vq
   variable out_line9  : line; -- vg
   variable out_line10 : line; -- vh
   variable out_line11 : line; -- lh
   variable out_line12 : line; -- sextante
   variable out_line13 : line; -- tg
   variable out_line14 : line; -- th
   variable out_line15 : line; -- tgh
   variable out_line16 : line; -- sextante_ip

   variable tso1aInt   : integer;
   variable tso2aInt   : integer;
   variable tso1bInt   : integer;
   variable tso2bInt   : integer;
   variable tso1cInt   : integer;
   variable tso2cInt   : integer;
   variable vdInt      : integer;
   variable vqInt      : integer;
   variable vgInt      : integer;
   variable vhInt      : integer;
   variable lhInt      : integer;
   variable sextInt    : integer;
   variable tgInt      : integer;
   variable thInt      : integer;
   variable tghInt     : integer;
   variable sextipInt  : integer;

   variable auxvd      : std_logic_vector(15 downto 0);
   variable auxvq      : std_logic_vector(15 downto 0);
   variable auxvg      : std_logic_vector(15 downto 0);
   variable auxvh      : std_logic_vector(15 downto 0);
   variable auxmx      : integer range 0 to 3000;
   variable auxip      : integer range 0 to 3;

   constant um         : integer range 0 to 200 := 1 ;
   constant csn        : integer range 0 to 200 := 170 ;
   constant maxi       : integer range 0 to 3000 := 2465;
   variable flag       : std_logic;

   begin
       wait for 10 ns;
       if(calc = '1' and flag = '0')
       then

        --TSO1A
        tso1aInt := CONV_INTEGER(tso1a);

        --TSO2A
        tso2aInt := CONV_INTEGER(tso2a);
```

```
--TSO1B
tso1bInt := CONV_INTEGER(tso1b);

--TSO2B
tso2bInt := CONV_INTEGER(tso2b);

--TSO1C
tso1cInt := CONV_INTEGER(tso1c);

--TSO2C
tso2cInt := CONV_INTEGER(tso2c);

 --Vd
 auxvd := CONV_STD_LOGIC_VECTOR(Vd,16);
 vdInt := CONV_INTEGER(auxvd);

 --Vq
 auxvq := CONV_STD_LOGIC_VECTOR(Vq,16);
 vqInt := CONV_INTEGER(auxvq);

 --Vg
 auxvg := CONV_STD_LOGIC_VECTOR(Vg,16);
 vgInt := CONV_INTEGER(auxvg);

 --Vh
 auxvh := CONV_STD_LOGIC_VECTOR(Vh,16);
 vhInt := CONV_INTEGER(auxvh);

 --Lh
 lhInt := CONV_INTEGER(Lh);

 --Sext
 sextInt := CONV_INTEGER(Sextante);

 --TG
 tgInt   := CONV_INTEGER(tg);

 --TH
 thInt   := CONV_INTEGER(th);

 --TGH
 tghInt  := CONV_INTEGER(tgh);

 --Sextante Impar ou Par
 sextipInt := CONV_INTEGER(Sext_ip);

 if (index = um) then

    --TSO1A
    write(out_line1,string'("mat1 = ["));
    writeline(out_file1, out_line1); --grava na primeira linha mat =[
    write(out_line1, tso1aInt);      --save results to line
    writeline(out_file1, out_line1); --write line to file

    --TSO2A
    write(out_line2,string'("mat2 = ["));
    writeline(out_file2, out_line2); --grava na primeira linha mat =[
    write(out_line2, tso2aInt);      --save results to line
    writeline(out_file2, out_line2); --write line to file

    --TSO1B
    write(out_line3,string'("mat3 = ["));
    writeline(out_file3, out_line3); --grava na primeira linha mat =[
    write(out_line3, tso1bInt);      --save results to line
    writeline(out_file3, out_line3); --write line to file

    --TSO2B
    write(out_line4,string'("mat4 = ["));
    writeline(out_file4, out_line4); --grava na primeira linha mat =[
    write(out_line4, tso2bInt);      --save results to line
    writeline(out_file4, out_line4); --write line to file

    --TSO1C
    write(out_line5,string'("mat5 = ["));
```

```
        writeline(out_file5, out_line5);  --grava na primeira linha mat =[
        write(out_line5, tso1cInt);      --save results to line
        writeline(out_file5, out_line5); --write line to file

        --TSO2C
        write(out_line6,string'("mat6 = ["));
        writeline(out_file6, out_line6); --grava na primeira linha mat =[
        write(out_line6, tso2cInt);      --save results to line
        writeline(out_file6, out_line6); --write line to file

        --VD
        write(out_line7,string'("mat7 = ["));
        writeline(out_file7, out_line7); --grava na primeira linha mat =[
        write(out_line7, vdInt);         --save results to line
        writeline(out_file7, out_line7); --write line to file

        --VQ
        write(out_line8,string'("mat8 = ["));
        writeline(out_file8, out_line8); --grava na primeira linha mat =[
        write(out_line8, vqInt);         --save results to line
        writeline(out_file8, out_line8); --write line to file

        --VG
        write(out_line9,string'("mat9 = ["));
        writeline(out_file9, out_line9); --grava na primeira linha mat =[
        write(out_line9, vgInt);         --save results to line
        writeline(out_file9, out_line9); --write line to file

        --VH
        write(out_line10,string'("mat10 = ["));
        writeline(out_file10, out_line10); --grava na primeira linha mat =[
        write(out_line10, vhInt);          --save results to line
        writeline(out_file10, out_line10); --write line to file

        --LH
        write(out_line11,string'("mat11 = ["));
        writeline(out_file11, out_line11); --grava na primeira linha mat =[
        write(out_line11, lhInt);          --save results to line
        writeline(out_file11, out_line11); --write line to file

        --Sextante
        write(out_line12,string'("mat12 = ["));
        writeline(out_file12, out_line12); --grava na primeira linha mat =[
        write(out_line12, sextInt);        --save results to line
        writeline(out_file12, out_line12); --write line to file

        --tg
        write(out_line13,string'("mat13 = ["));
        writeline(out_file13, out_line13); --grava na primeira linha mat =[
        write(out_line13, tgInt);         --save results to line
        writeline(out_file13, out_line13); --write line to file

        --th
        write(out_line14,string'("mat14 = ["));
        writeline(out_file14, out_line14); --grava na primeira linha mat =[
        write(out_line14, thInt);          --save results to line
        writeline(out_file14, out_line14); --write line to file

        --tgh
        write(out_line15,string'("mat15 = ["));
        writeline(out_file15, out_line15); --grava na primeira linha mat =[
        write(out_line15, tghInt);         --save results to line
        writeline(out_file15, out_line15); --write line to file

        --tgh
        write(out_line16,string'("mat16 = ["));
        writeline(out_file16, out_line16); --grava na primeira linha mat =[
        write(out_line16, sextipInt);         --save results to line
        writeline(out_file16, out_line16); --write line to file

        flag := '1';

    elsif (index = csn) then

        --TSO1A
```

```
write(out_line1, tso1aInt);      --save results to line
writeline(out_file1, out_line1); --write line to file
write(out_line1, string'("];"));
writeline(out_file1, out_line1); --write line to file


--TSO2A
write(out_line2, tso2aInt);      --save results to line
writeline(out_file2, out_line2); --write line to file
write(out_line2, string'("];"));
writeline(out_file2, out_line2); --write line to file


--TSO1B
write(out_line3, tso1bInt);      --save results to line
writeline(out_file3, out_line3); --write line to file
write(out_line3, string'("];"));
writeline(out_file3, out_line3); --write line to file

--TSO2B
write(out_line4, tso2bInt);      --save results to line
writeline(out_file4, out_line4); --write line to file
write(out_line4, string'("];"));
writeline(out_file4, out_line4); --write line to file

--TSO1C
write(out_line5, tso1cInt);      --save results to line
writeline(out_file5, out_line5); --write line to file
write(out_line5, string'("];"));
writeline(out_file5, out_line5); --write line to file

--TSO2C
write(out_line6, tso2cInt);      --save results to line
writeline(out_file6, out_line6); --write line to file
write(out_line6, string'("];"));
writeline(out_file6, out_line6); --write line to file

--VD
write(out_line7,vdInt);
writeline(out_file7, out_line7);  --grava na primeira linha mat =[
write(out_line7, string'("];"));  --save results to line
writeline(out_file7, out_line7);  --write line to file

--VQ
write(out_line8,vqInt);
writeline(out_file8, out_line8);  --grava na primeira linha mat =[
write(out_line8, string'("];"));  --save results to line
writeline(out_file8, out_line8);  --write line to file

--VG
write(out_line9,vgInt);
writeline(out_file9, out_line9);  --grava na primeira linha mat =[
write(out_line9, string'("];"));  --save results to line
writeline(out_file9, out_line9);  --write line to file

--VH
write(out_line10,vhInt);
writeline(out_file10, out_line10);  --grava na primeira linha mat =[
write(out_line10, string'("];"));   --save results to line
writeline(out_file10, out_line10);  --write line to file

--LH
write(out_line11,LhInt);
writeline(out_file11, out_line11); --grava na primeira linha mat =[
write(out_line11, string'("];"));  --save results to line
writeline(out_file11, out_line11); --write line to file

--SEXTANTE
write(out_line12,sextInt);
writeline(out_file12, out_line12); --grava na primeira linha mat =[
write(out_line12, string'("];"));  --save results to line
writeline(out_file12, out_line12); --write line to file

--TG
write(out_line13,tgInt);
```

```
    writeline(out_file13, out_line13); --grava na primeira linha mat =[
    write(out_line13, string'("];"));  --save results to line
    writeline(out_file13, out_line13); --write line to file

    --TH
    write(out_line14,thInt);
    writeline(out_file14, out_line14); --grava na primeira linha mat =[
    write(out_line14, string'("];"));  --save results to line
    writeline(out_file14, out_line14); --write line to file

    --TGH
    write(out_line15,tghInt);
    writeline(out_file15, out_line15); --grava na primeira linha mat =[
    write(out_line15, string'("];"));  --save results to line
    writeline(out_file15, out_line15); --write line to file

    --Sextante impar ou par
    write(out_line16,sextipInt);
    writeline(out_file16, out_line16); --grava na primeira linha mat =[
    write(out_line16, string'("];"));  --save results to line
    writeline(out_file16, out_line16); --write line to file

    wait;
else

    --TSO1A
    write(out_line1, tso1aInt);       --save results to line
    writeline(out_file1, out_line1);  --write line to file

    --TSO2A
    write(out_line2, tso2aInt);       --save results to line
    writeline(out_file2, out_line2);  --write line to file

    --TSO1B
    write(out_line3, tso1bInt);       --save results to line
    writeline(out_file3, out_line3);  --write line to file

    --TSO1A
    write(out_line4, tso2bInt);       --save results to line
    writeline(out_file4, out_line4);  --write line to file

    --TSO1C
    write(out_line5, tso1cInt);       --save results to line
    writeline(out_file5, out_line5);  --write line to file

    --TSO1A
    write(out_line6, tso2cInt);       --save results to line
    writeline(out_file6, out_line6);  --write line to file

    --VD
    write(out_line7, vdInt);         --save results to line
    writeline(out_file7, out_line7); --write line to file

    --VQ
    write(out_line8, vqInt);         --save results to line
    writeline(out_file8, out_line8); --write line to file

    --VG
    write(out_line9, vgInt);         --save results to line
    writeline(out_file9, out_line9); --write line to file

    --VH
    write(out_line10, vhInt);          --save results to line
    writeline(out_file10, out_line10); --write line to file

    --LH
    write(out_line11, lhInt);          --save results to line
    writeline(out_file11, out_line11); --write line to file

    --SEXTANTE
    write(out_line12, sextInt);          --save results to line
    writeline(out_file12, out_line12); --write line to file

    --TG
    write(out_line13, tgInt);          --save results to line
```

```
            writeline(out_file13, out_line13); --write line to file

            --TH
            write(out_line14, thInt);          --save results to line
            writeline(out_file14, out_line14); --write line to file

            --TGH
            write(out_line15, tghInt);          --save results to line
            writeline(out_file15, out_line15);  --write line to file

            --SEXTANTE IMPAR OU PAR
            write(out_line16, sextipInt);      --save results to line
            writeline(out_file16, out_line16); --write line to file

            flag := '1';
         end if;


       end if;

       if(calc = '0')
       then
         flag := '0';
       end if;

   end process;

---Ligando algumas saidas do Chip na arquitetura de teste

         indexa   <= index;

end only;
```

Anexo D – Desenvolvimento do Algoritmo da Modulação Vetorial em VHDL.

```
-----Edvaldo F. Freitas Lima
---DESENVOLVIMENTO DO PROJETO EM VHDL
Library ieee;
Use ieee.std_logic_1164.all;


Entity GeraVdVq32 is

   Port(
        ---Saidas e entradas comuns a todos os blocos
          index_mod  :  in   signed (15 downto 0); -- valor do indice de modulação
          rst        :  in   std_logic;
          start  :  in   std_logic;
          fim_calc   :  in   std_logic;
          clk        :  in   std_logic;
          index      :  out  integer range 0 to 200
          calc       :  out  std_logic;
          Lh         :  out  integer range 0 to 45;
          tg         :  out  signed (15 downto 0); -- Q2.14
          th         :  out  signed (15 downto 0); -- Q2.14
          tgh        :  out  signed (15 downto 0); -- Q2.14
          freq       :  out  std_logic; -- geração da onda quadrada;
          err_time   :  out  std_logic; -- erro nos cálculos de tg th e tgh
          sext_ip    :  out  std_logic  -- determina se o sextante é impar ou par

               );

end GeraVdVq32;

Architecture  ArchGeraVdVq of GeraVdVq32 is
Type       tipo_estado   is (e0,e1,e2,e3,e4,e5);
Type       Tvetor        is array (0 to 451) of integer;
Type       Tvetor_Sext   is array (0 to 6)   of integer;

Signal     px_estado   :  tipo_estado;
Signal     Sigmod      :  signed ( 15 downto 0 ); -- Vetor de Referencia


Begin

--- Processo para inicio dos cálculos, cálculo do Vetor de Referência
process (clk,start)
 ---------------------------------------------------------------------
 ---BLOCO DE VARIAVEIS PARA GERAÇÃO DO VETOR DE REFERENCIA VD E VQ---
 ---------------------------------------------------------------------

  --Array com os valores dos senos, após sua indexação utilizar a função CONV_SIGNED
  Constant  vetsen :
Tvetor:=(0,57,114,172,229,286,343,400,457,515,572,629,686,743,800,857,915,972,1029,1086,114
3,1200,1257,1314,1371,1428,1485,1542,1599,1656,1713,1769,1826,1883,1940,1997,2053,2110,2167
,2224,2280,2337,2393,2450,2507,2563,2619,2676,2732,2789,2845,2901,2958,3014,30703126,3182,3
238,32943350,3406,3462,3518,3574,3630,3686,3741,3797,3853,39083964,4019,4075,4130,4185,4240
,4296,4351,4406,4461,4516,571,4626,4681,4735,4790,4845,4899,4954,5009,5063,5117,5172,5226,5
280,5334,5388,5442,5496,5550,5604,5657,5711,765,5818,5872,5925,5978,6031,6084,6138,6191,624
3,6296,349,6402,6454,6507,6559,6612,6664,6716,6768,6820,6872,6924,6976,7028,7079,7131,7182,
7234,7285,7336,7387,7438,489,7540,9584,9630,9676,9723,9769,9814,9860,9906,9951,9997,10042,1
0087,0132,10177,10222,10266,10311,10355,10399,10444,10488,10531,10575,10619,10662,10706,107
49,10792,10835,10878,0920,10963,11005,11048,11090,11132,11174,11216,11257,1299,11340,11381,
11422,11463,11504,11545,11585,11626,11666,11706,11746,11786,11825,11865,11904,11943,11982,1
2021,12060,12099,12137,12176,12214,12252,12290,12328,12365,12403,12440,12477,12514,12551,12
588,12624,12660,12697,12733,12769,12804,12840,12875,12911,12946,12981,13016,13050,130857591
,7641,7692,7742,7793,7843,7893,7943,7993,043,8093,8142,8192,8241,8291,8340,8389,8438,8487,8
536,585,8634,8682,8731,8779,8827,8875,8923,8971,9019,9067,114,9162,9209,9256,9304,9351,9397
,9444,9491,9538,9584,9630,9676,9723,9769,9814,9860,9906,9951,9997,10042,10087,0132,10177,10
222,10266,10311,10355,10399,10444,10488,10531,10575,10619,10662,10706,10749,10792,10835,108
78,0920,10963,11005,11048,11090,11132,11174,11216,11257,1299,11340,11381,11422,11463,11504,
11545,11585,11626,11666,11706,11746,11786,11825,11865,11904,11943,11982,12021,12060,12099,1
2137,12176,12214,12252,12290,12328,12365,12403,12440,12477,12514,12551,12588,12624,12660,12
697,12733,12769,12804,12840,12875,12911,12946,12981,13016,13050,13085,13119,13153,13187,132
21,13255,13288,13322,13355,13388,13421,13454,13486,13519,13551,13583,13615,13647,13678,1371
0,13741,13772,13803,13833,13864,13894,13925,13955,13985,14014,14044,14073,14102,14131,14160
,14189,14217,14246,14274,14302,14330,14357,14385,14412,14439,14466,14493,14520,14546,14572,
14598,14624,6358,16362,16364,16367,16370,16372,16374,16376,16378,16379,16380,16382,16382,16
383,16384,16384,16384,16384);
```

```
  Variable  aux_Vref_32 :   signed ( 31 downto 0 ) := "00000000000000000000000000000000";
  Variable  aux_Vref_16 :   signed ( 15 downto 0 ) := "0000000000000000";
  Variable  var_Vref    :   signed ( 31 downto 0 ) ;
  Variable  theta       :   signed ( 15 downto 0 ) := "111100101100111";
  Variable  alfa        :   signed ( 15 downto 0 ) := "0000000000100000";
  Variable  index_cos   :   std_logic_vector (  8 downto 0 ) := "000000000";
  Variable  index48     :   signed ( 47 downto 0 ) := 000000000000000000000000000000000000";
Variable  Iv1s, Iv2s    :   integer range 0 to 500 := 0;
  Variable  Iv1c, Iv2c  :   integer range 0 to 500 := 0; --
  Variable  quadrante   :   integer range 0 to 6;
  Variable  index_fra   :   signed ( 31 downto 0 ) := "00000000000000000000000000000000";
  Variable  index_int   :   std_logic_vector (  8 downto 0 ) := "000000000";
  Variable  v1_s,v2_s   :   signed ( 15 downto 0 ) := "0000000000000000";
  Variable  v1_c,v2_c   :   signed ( 15 downto 0 ) := "0000000000000000
  Variable  ss, sc      :   std_logic; -- Sinal do seno e coseno 1 é '+' e 0 é '-'
  Variable  Vsen, Vcos  :   signed ( 15 downto 0 );
  Variable  difV        :   signed ( 15 downto 0 ); --  V1
  Variable  mul_idx_48  :   signed ( 47 downto 0 ); -- modulação vezes a diferença
  Variable  vsc         :   signed ( 15 downto 0 ); -- varaivel auxiliar
  Variable  Var_Vd, Var_Vq   :   signed ( 47 downto 0 );
  Variable  var_index   :   integer range 0 to 200;

 ------------------------------------------------------
 ---BLOCO DE VARIAVEIS PARA IDENTIFICAÇÃO DO SEXTANTE---
 ------------------------------------------------------

   Variable    A,B,C            : std_logic_vector (3 downto  0);
   Variable    Ns               : std_logic_vector (3 downto 0);
   Variable    Ns_int,Sext      : integer range 0 to 6;
   Variable    Aux_sext         : std_logic_vector (2 downto 0);
   Variable    sigA,sigB,sigC   : std_logic;
   Variable    vd_16,vq_16      : signed (15 downto 0);

   Variable    produto          : signed (31 downto 0);
   Variable    prod_q3_12       : signed (15 downto 0);
   Variable    temp_calc        : signed (15 downto 0);
   Variable    vq_sft           : signed (15 downto 0);
   Constant    sqrt3            : signed (15 downto 0) := "0110111011011010";
   Constant    vetsext          : Tvetor_Sext :=(0,2,6,1,4,3,5);


-----------------------------------------------------------------
---BLOCO DE VARIAVEIS PARA NORMALIZAÇÃO DO VETOR DE REFERENCIA---
-----------------------------------------------------------------
   Variable vg_part1, vg_part2, vh_part1, vh_part2 : signed (31 downto 0);
   Variable var_vg, var_vh : signed (31 downto 0);
   Variable AuxA11 : signed (15 downto 0);
   Variable AuxA12 : signed (15 downto 0);
   Variable AuxA21 : signed (15 downto 0);
   Variable AuxA22 : signed (15 downto 0);
   A11  : Tvetor_Sext :=(0, 40960,40960, 24576,24576,0,0);
   A12  : Tvetor_Sext :=(0,51347, ,14189,51347, 14189,28378,37158);
   A21  : Tvetor_Sext :=(0,0,40960,40960,0,24576,24576);
   A22  : Tvetor_Sext :=(0,28378,14189,51347,37158,51347,14189);



-------------------------------------------------------------
---BLOCO DE VARIAVEIS PARA CÁLCULO DAS RAZOES CÍCLICAS E LH---
-------------------------------------------------------------
  variable  md          :  signed (2 downto 0); -- modo
  variable  md_int      :  integer range 0 to 10;
  variable  vgmaisvh    :  signed (15 downto 0); -- variavel intermediaria
  variable  Ls          :  integer; -- localizaçao do triangulo
  variable  Vhu,Vgu     :  signed (1 downto 0); -- Q2.14 --> XX,000000000000
  variable  Vh_int,Vg_int : integer range 0 to 10;
  variable  Vhf,Vgf     :  signed (15 downto 0); -- Q3.13 --> 000,XXXXXXXXXXX
  variable  tipo        :  integer;
  variable  tipo_vec    :  signed (15 downto 0);
  variable  var_tg,var_th,var_tgh : signed (15 downto 0);
  variable  var_tg32,var_th32     : signed (31 downto 0);
  variable  triang_type :  signed (15 downto 0) := "0000000000000000";
  variable  sext_int    :  integer range 0 to 10;
  variable  Lhsig       :  integer range 0 to 45;
  variable  vg_16,vh_16 :  signed (15 downto 0);
```

```
  variable  prim_calc   :  std_logic := '1';

  constant  umQ214      :  signed (15 downto 0) := "0100000000000000";
  constant  umb         :  integer := 1;
  constant  dezesseis   :  integer := 4;


begin
----------------------RESET--------------------------------
       if (start = '0' or rst = '1') then
              --Vref  <= "0000000000000000";
              calc  <= '0';
              var_index := 0;
              theta := "1111111101100111";
              sext_ip <= '0';
              px_estado <= e0;


       elsif (clk'event and clk = '1') then
              if ( start = '1' and rst = '0') then
               case px_estado is
-----------------Estado E0-------------------------------
----------------Controle do Fluxo de dados--------------
             when e0 =>
                              calc  <= '0';
                           --Vref = (2*m)/pi ;
                           aux_Vref_16 := Sigmod + Sigmod; -- Q2,14 + Q2,14 = Q2,14

                           aux_Vref_32 := aux_Vref_16 * um_div_pi;-- Q2,14*Q1,15 = Q3,29
                           Var_Vref := aux_Vref_32; -- Q3,29 --aux_Vref_32(29 downto 14);
                           index <= 0; -- identifica o numero da iteração
                           px_estado <= e1;
-----------------Estado E1-------------------------------
----------------Geração do Vetor de Referencia----------
                             when e1 =>
                               theta := delta + theta;    -- Primeiro é o delta
                               calc  <= '0';
                               var_index := var_index + 1;
                               ---Definindo os Quadrantes,o angulo passa ser agora o alfa
                               if (theta <= pi_div2) then
                              --Quadrante 1
                              quadrante := 1;
                                           alfa := theta;
                                           ss := '1';
                                           sc := '1';
                                           freq <= '1'; -- determinar a onda quadrada

                                    elsif ( theta > pi_div2 and theta <= pi) then
                              --Quadrante 2
                                           quadrante := 2;
                                           alfa := pi - theta;  -- Q4,12 - Q4,12
                                           ss := '1';
                                           sc := '0';
                                           freq <= '1'; -- determinar a onda quadrada

                                    elsif ( theta > pi and theta <= piX3_div2 ) then
                              --Quadrante 3
                                           quadrante := 3;
                                           alfa := theta - pi;  -- Q4,12
                                           ss := '0';
                                           sc := '0';
                                           freq <= '0'; -- determinar a onda quadrada

                                    elsif ( theta > piX3_div2 and theta <= X2_pi) then
                              --Quadrante 4
                                           quadrante := 4;
                                           alfa := X2_pi - theta; -- Q4,12 - Q4,12
                                           ss := '0';
                                           sc := '1';
                                           freq <= '0'; -- determinar a onda quadrada

                                  end if;

                                           --Definindo o index
                                           Index48 := alfa * fat_index;-- Q4,12 * Q10,22
```

```
                                        index_int := "000000000";   -- Q9,0

                                        index_int := std_logic_vector (index48(42
downto 34));
                                        index_fra :=
"00000000000000000000000000000000";
index_fra(30 downto 0) := index48(33 downto 3);


                        --Cálculo dos indices de Seno e Cosseno
                                        Iv1s := CONV_INTEGER(index_int);
                                        Iv2s := CONV_INTEGER(Iv1s + um);

                                        index_cos := val_451 - Iv1s;
                                        Iv1c := CONV_INTEGER(index_cos);
                                        Iv2c := CONV_INTEGER(Iv1c + um);


                                        --Buscando os valores do Seno
                                        v1_s := CONV_SIGNED(vetsen(Iv1s),16);
                                        v2_s := CONV_SIGNED(vetsen(Iv2s),16);

                                        --Buscando os valores do Cosseno
                                        v1_c := CONV_SIGNED(vetsen(Iv1c),16);
                                        v2_c := CONV_SIGNED(vetsen(Iv2c),16);

                                    difV := (v2_s - v1_s);       -- Q2,14 - Q2,14

                                    mul_idx_48 := difV*index_fra;-- Q2,14 * Q1,31
                                    vsc := mul_idx_48(46 downto 31) + v1_s; -- Q2,14

                                    if(ss = '0') then
                                         Vsen := -vsc;
                                    else
                                      Vsen := vsc;
                                    end if;


                                    difV := (v2_c - v1_c);       -- Q2,14 - Q2,14
                                    mul_idx_48 := difV*index_fra;-- Q2,14 * Q1,31
                                    vsc := v2_c - mul_idx_48(46 downto 31); -- Q2,14

                                    if(sc = '0') then
                                         Vcos := -vsc;
                                    else
                                      Vcos :=  vsc;
                                    end if;

                                        Var_Vd := Var_Vref * Vcos; -- Q3,29 * Q2,14
                                        Var_Vq := Var_Vref * Vsen; -- Q3,29 * Q2,14
                                        vd_16  := Var_Vd(43 downto 28); -- Q1,15
                                        vq_16  := Var_Vq(43 downto 28); -- Q1,15


                                        px_estado <= e2;
-----------------Estado E2--------------------------------
-----------------Identificação do Sextante---------------
                                when e2  =>
                                        ------Cálculo do Sextante
                                        ------ A ------
                                        A := "0000";
                                        sigA := Vq_16(15) xor '1';
                                        A(0) := sigA;
                                        ---------------------------------------------
                                        ---------------------------------------------
                                        produto    := (sqrt3 * vd_16); -- Q2.14 * Q1.15
                                        prod_q3_12 := produto(31 downto 16); -- Q3,29

                                        vq_sft(12 downto 0) := vq_16(14 downto 2);

                                        if ( vq_16(15) = '1') then
                                               vq_sft(14 downto 13) := "11";
                                        else
                                               vq_sft(14 downto 13) := "00";
```

```
                                        end if;
                                               vq_sft(15) := vq_16(15);

                                        ------- B -------
                                        B := "0000";
                                        temp_calc  :=  Vq_sft - prod_q3_12;
                                        sigB := temp_calc(15);
                                        B(1) := sigB;

                                        ------- C --------
                                        C := "0000";
                                        temp_calc  :=  Vq_sft + prod_q3_12 ;
                                        sigC := temp_calc(15);
                                        C(2) := sigC;

                                        ------ Numero do Sextante -----
                                        Ns := ((A or B) or C);
                                        Ns_int := CONV_INTEGER(Ns);
                                        Sext := vetsext(Ns_int);
                                        Aux_Sext := CONV_STD_LOGIC_VECTOR(Sext,3);


                                        px_estado <= e3;           ---  Sempre um
estado a mais apos a edição e adicionamento de mais um bloco

-----------------Estado E3-------------------------------
-----------------Normalização Vg e Vh--------------------
                            when e3  =>
                                   AuxA11  := CONV_SIGNED(A11(Sext),16);
                                        vg_part1:= AuxA11* Vd_16;  --Q3,13 * Q1,15

                                        AuxA12  := CONV_SIGNED(A12(Sext),16);
                                        vg_part2:= AuxA12* Vq_16;  --Q3,13 * Q1,15

                                        var_vg  := vg_part1 + vg_part2;--Q4,28
                                        vg_16   := var_vg(29 downto 14);

                                        AuxA21  := CONV_SIGNED(A21(Sext),16);
                                        vh_part1:= AuxA21*Vd_16;  --Q3,13 * Q1,15

                                        AuxA22  := CONV_SIGNED(A22(Sext),16);
                                        vh_part2:= AuxA22*Vq_16;  --Q3,13 * Q1,15
                                        var_vh  := vh_part1 + vh_part2;--Q4,28
                                        vh_16   := var_vh(29 downto 14);

                                        px_estado <= e4;

-----------------Estado E4-------------------------------
-----------------Cálculo das razões cíclicas e Lh--------
                             when e4  =>
                                      Vhf := "0000000000000000";  -- Q2.14
                                      Vhf (13 downto  0) := vh_16(13 downto  0); -- Q2.14
                                      Vhu := vh_16(15 downto 14); -- Q2.14
                                      Vh_int := CONV_INTEGER(Vhu);

                                      Vgf := "0000000000000000";  -- Q2.14
                                      Vgf (13 downto  0) := vg_16(13 downto  0); -- Q2.14
                                      Vgu := Vg_16(15 downto 14); --Q2.14
                                      Vg_int := CONV_INTEGER(Vgu);

                                      vgmaisvh := vg_16 + vh_16 ; --Q2.14 + Q2.14 = Q2.14
                                      md :="000";
                                      md(1 downto 0) := vgmaisvh(15 downto 14);
                                      Ls := (md_int*md_int)+md_int+umb + Vh_int - Vg_int;
                                      tipo     := Ls + md_int;
                                      tipo_vec := CONV_SIGNED (tipo,16);
                                      triang_type := "0000000000000000";
                                      triang_type(14) := not(tipo_vec(0));
                                      --------------Variavel Tg-----------------
                                      var_tg := ( triang_type - Vgf ); -- Q2.14 - Q2.14
                                      if (var_tg(15) = '1') then
                                                 var_tg := -var_tg;
                                      end if;
```

```
                                    ---------------Variavel Th-------------------------
--
                                    var_th := ( triang_type - Vhf ); -- Q3.13 - Q3.13
                                    if (var_th(15) = '1') then
                                              var_th := -var_th;
                                    end if;









                                    ----------------Variavel Tgh-----------------------
--
                                    var_tgh := umQ214 - var_tg - var_th; -- Q13.3

                                    sext_int := CONV_INTEGER (sext);
                                    Lhsig  := (sext_int - umb)*dezesseis + Ls; -- apenas
inteiros                                   px_estado <= e5;

-----------------Estado E5-------------------------------
                              when e5  =>
                              --Assegura os valores na saida do Bloco

                                   --Saidas Tg Th Tgh e Lh
                                   Lh  <= Lhsig;
                                   tg  <= var_tg;  -- Q2.14
                                   th  <= var_th;  -- Q2.14
                                   tgh <= var_tgh; -- Q2.14
                                   err_time <= var_tg(15) or var_th(15) or
var_tgh(15);
                                   Sext_ip <= Aux_Sext(0);
                                   calc  <= '1';
                                   index <= var_index;
                                  px_estado <= e5;

                                 end if;
           end if;
end process;
end  ArchGeraVdVq;
```

Anexo E – Cálculos dos Tempos de Chaveamento em VHDL.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

Entity time_switch1a is
    port (
              Ts      :  in  std_logic_vector (15 downto 0);
              timer   :  in  std_logic; -- Habilitar bloco apos os calculos
              Lh      :  in  integer range 0 to 45; -- Inteiro
              tg      :  in  signed (15 downto 0);  -- Q2.14
              th      :  in  signed (15 downto 0);  -- Q2.14
              tgh     :  in  signed (15 downto 0);  -- Q2.14
              clk     :  in  std_logic; -- Sinal de clock

             comparador  :  out std_logic;  -- Habilitar bloco para gerar PWM
              TSO1A      :  out std_logic_vector (15 downto 0) fase A

         );
end time_switch1a;

Architecture  Archtime_switch of time_switch1a is
Type       Tvetor        is array (1 to 24) of integer;

begin
       process (clk,timer)

    variable  part_tg   : signed (31 downto 0);
    variable  part_th   : signed (31 downto 0);
    variable  part_tgh  : signed (31 downto 0);
    variable  aux_tg    : signed (15 downto 0);
    variable  aux_th    : signed (15 downto 0);
    variable  aux_tgh   : signed (15 downto 0);
    variable  temp_TS0  : std_logic_vector (31 downto 0);
    variable  temp_TSA  : std_logic_vector (47 downto 0);
    variable  temp_TS1  : std_logic_vector (15 downto 0);


--------------Constantes para as chaves 1 da fase A
              Constant Kg1a : Tvetor :=
(4096,0,4096,0,4096,4192,0,0,8192,8192,8192,8192,0,0,0,0,
       4192,8192,4096,8192,8196,8192,4096,8192);

              Constant Kh1a : Tvetor :=
(4096,0,4096,0,0,0,4096,0,8192,8992,8192,8192,0,0,0,0,4096,
                                          8192,8192,0,4096,8194096,8192);

              Constant Kgh1a : Tvetor:=
(6144,4096,0,4096,4048,4096,0,0,2134,8092,8992,8192,204
              0,0,0,6144,812,8192,4096,2048,4096,8192,4096);

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

        aux_tg  := CONV_SIGNED(Kg1a (Lh),16);
        aux_th  := CONV_SIGNED(Kh1a (Lh),16);
        aux_tgh := CONV_SIGNED(Kgh1a(Lh),16);
        part_tg  := aux_tg  * tg;   -- Q3.13*Q2.14 = Q5.27
        part_th  := aux_th  * th;   -- Q3.13*Q2.14 = Q5.27
        part_tgh := aux_tgh * tgh;  -- Q3.13*Q2.14 = Q5.27
 --      Q5.27
        temp_TS0 := part_tg + part_th + part_tgh; -- Q5.27 vetor com 32 bits
        temp_TSA := temp_TS0 * Ts; -- Q5.27 * Q13.3 vetor com Q18.30 vetor com 48 bits
        temp_TS1 := "0000000000000000";
        temp_TS1(12 downto 0) := temp_TSA(42 downto 30);
        TSO1A <= temp_TS1; -- Apenas a parte inteira 13 bits
        comparador <='1';
      else
        comparador <= '0';
      end if;
     end if;
    end process;
end Archtime_switch;
```

Anexo F – Modulador PWM desenvolvido em VHDL

```
---Edvaldo Lima
---Módulo de Controle do processo de gração dos sinais de PWM para controle do Inversor

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

Entity ctrl_pwm is
 port (
         max_delay    : in   std_logic_vector (15 downto 0); -- valor do tempo morto
         cont_max     : in   std_logic_vector (15 downto 0); -- valor maximo do contador
         comparador   : in   std_logic; -- sinal de entrado do bloco comparador
         TSO1A        : in   std_logic_vector (15 downto 0); -- valor de Comparação
         clk          : in   std_logic; -- sinal do clock
         rst          : in   std_logic; -- sinal do reset
         Sext_ip      : in   std_logic; -- sinal que define se o Sextante é impar ou par

         up_dw        : out  std_logic; -- sinal de subida e descida do contador
         pwm_dead     : out  std_logic; -- Sinal de PWM com tempo Morto
         pwm_dead_not : out  std_logic; -- Sinal complementar do PWM com tempo Morto
         new_calc     : out  std_logic;  -- sinal para habilitar um novo cálculo
         pwm_original : out  std_logic; -- Sinal de PWM
         cont         : out  std_logic_vector (15 downto 0)
      );
end ctrl_pwm;

Architecture Arch_pwm of ctrl_pwm is

  Type    tipo_estado is (e0,e1,e2,e3,e4,e5,e6);
  Signal  px_estado     : tipo_estado ;

  Signal  ref2,ref1     : std_logic_vector (15 downto 0);
  Signal  pwm           : std_logic;
  Signal  fim_pwm       : std_logic;
  Signal  sig_pwm_dead  : std_logic;
  Signal  sig_pwm_dead_not : std_logic;
  Signal  enb_cont      :   std_logic;
  Signal  ref1_ig_ref2  :   std_logic;
  Signal  sig_ip        :   std_logic;


begin

-------------------------------------------------------
-------       Processo gerador de PWM            -----
-------------------------------------------------------

process(clk,rst,enb_cont)
   variable up_down  : std_logic;
   variable contador : std_logic_vector (15 downto 0);

  begin
       if(enb_cont = '0' or rst ='1')then
              contador := "0000000000000000";
              up_down  := '1';
              fim_pwm  <= '0';

       elsif (clk'event and clk='1') then -- Rising edge (gatilhado na subida).
           if (ref1_ig_ref2 = '1') then
                ref2 <= ref1; -- usado por causa do primeiro valor que eh calculado

           end if;

              if(up_down='1') then
                     if(contador = cont_max) then -- Comparação com o valor maximo do
contador
                            up_down  := '0';
                     else
                            contador := contador +1;
                     end if;

                     if (contador = "0000000000000011") then -- pequeno atraso no sinal de
fim_pwm
                  fim_pwm  <= '0';
               end if;
```

```
              else
                     if(contador = "0000000000000000") then
                            up_down := '1';
                            ref2 <= ref1;
                     else
                            contador := contador -1;

                            if(contador = "0000000000000011") then
                                fim_pwm <= '1';
                            end if;
                     end if;

              end if;
-------------------------------------------------------------
---              Geração do Sinal de PWM                ----
-------------------------------------------------------------

          if(clk'event and clk ='1') then
               if ( contador >= ref2) then  -- compara com ref2
                   if(sig_ip = '0') then
                          pwm <= '0'; -- Sextante PAR
                   else
                    pwm <= '1'; -- Sextante IMPAR
              end if;
                 else
                     if(sig_ip = '0') then
                            pwm <= '1'; -- Sextante PAR
                        else
                         pwm <= '0'; -- Sextante IMPAR
                     end if;
                 end if;
          end if;
                Pwm_original <= pwm;
                up_dw <= up_down;
                Cont <= contador;
        end if;

  end process;

-------------------------------------------------------
------      Processo gerador de Tempo Morto     ------
-------------------------------------------------------

process (clk,pwm,enb_cont)
        variable q   : std_logic_vector(15 downto 0);
        constant nnu : std_logic_vector (15 downto 0) := "0000001111011111";
        constant zro : std_logic_vector (15 downto 0) := "0000000000000000";

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

            if (enb_cont = '1') then

                 if (TSO1A = nnu or TSO1A = zro) then

                    pwm_dead <= pwm;
                        pwm_dead_not <=not (pwm);

                 else
                      if (pwm='0') then
                             if(q=0)then
                              pwm_dead <= '0';
                              pwm_dead_not <='1';
                             else
                              q := q-1;
                              pwm_dead <='0';
                              pwm_dead_not <='0';
                             end if;
                      else
                             if(q = max_delay)then
                              pwm_dead <= '1';
                              pwm_dead_not <='0';
                             else
                              q := q+1;
                              pwm_dead <='0';
```

```
                              pwm_dead_not <='0';
                             end if;
                       end if;-- if pwm = 0
                  end if; -- comparação do TSO1A com nnu e zro
               else
                             pwm_dead <= '0';
                             pwm_dead_not <= '0';
               end if; -- if enable contador


       end if;--clock

end process;

-------------------------------------------------------------------------
-------- Maquina de Estado para controle do Gerador do PWM      -------
-------- Sincronismo entre os Cálculos e a Geração de PWM       -------
-------------------------------------------------------------------------

process ( clk, comparador,rst )
   variable  var_ref1, var_ref2 : std_logic_vector (15 downto 0);
begin

   if rst = '1' then
      px_estado <= e0;
      enb_cont  <= '0';
      new_calc  <= '0';

   elsif clk'event and clk = '1' then
               case px_estado is
                      when e0 =>
                        if ( rst = '1') then
                           px_estado <= e0;

                     elsif (comparador = '1') then
                           px_estado  <= e1;  -- ja terminou o primeiro calculo, colocar o
valor de TSO1A para ser comparado com a onda triangular
                           --new_calc  <= '0'; -- ainda nao ha nada calculado

                     else
                                 enb_cont  <= '0'; -- desabilita o contador 1° ciclo ainda
                                 new_calc  <= '0'; -- ainda nao ha nada calculado

                                 px_estado <= e0 ; -- fica aguardando alguma coisa a ser
calculada
                                 ref1_ig_ref2 <= '1'; -- não ha atualizaçao de variavel
ainda, primeiro calculo
                          if;

                       when e1 =>
                          if (rst = '1') then
                            px_estado <= e0;

                       else
                                 var_ref2  := TSO1A;  -- coloca no registrado  o primeiro
valor de TSO1A calculado
                                 ref1_ig_ref2 <= '1'; -- indica que eh o primeiro calculo para
o algoritmo
                                 px_estado    <= e2;     -- muda de estado, apenas para
assegurar o valor TSO1A que sera savo no registrador
                           if;

                       when e2 =>
                          if ( rst = '1') then
                            px_estado <= e0;

                       ---------Primeiro Valor Calculado-----------
                       elsif (comparador = '1') then
                           ref1_ig_ref2 <= '1'; -- sinaliza que ainda eh o primeiro cálculo
                           ref1    <= var_ref2; -- coloca o valor do registrador para a
variavel, para que em seguida ela seja comparada

                           sig_ip    <= Sext_ip; -- Atualiza o Sinal de IMPAR ou PAR do
Sextante.
                           enb_cont  <= '1'; -- habilita a contagem do contador
```

```
                        new_calc  <= '1'; -- habilita o circuito para um novo cálculo,
qdo o comparador passar a zero new_cal tem que ir para zero tmb
                        px_estado <= e2 ; -- fica aguardando ate que o valor de
comparador va pra 0

                    else
                        px_estado <= e3;  -- troca de estado quando comparador passa para
0 novamente, isto significa que o circuito esta fazendo o calculo para o proximo valor de
TSO1A
                        ref1_ig_ref2 <= '1'; -- sinaliza que eh o primeiro calculo

                    end if;

                   when e3 =>
                       if (rst = '1') then
                 px_estado <= e0;

                       elsif ( fim_pwm = '1' and comparador = '1') then -- passa para um
novo estado apenas quando terminar o ciclo de pwm e calculo
                        px_estado <= e4;

                    else
                        new_calc  <= '0'; -- dar um pulso em new_calc
                        px_estado <= e3 ; --
                        ref1_ig_ref2 <= '1'; -- sinaliza que ainda e o primeiro calculo
                    end if;

                when e4 =>
                    if (rst = '1') then
                        px_estado <= e0;

                    else
                        Sig_ip    <= Sext_ip; -- Atualiza o tipo de sextante PAR ou
IMPAR
                        var_ref1  := TSO1A;   -- guarda o valor de tso1a no registrador
                        px_estado <= e5;
                        ref1_ig_ref2 <= '0'; -- sinaliza que NÃO eh mais o primeiro
calculo

                    end if;

                when e5 =>
                    if (rst = '1') then
                        px_estado <= e0;

                    elsif (comparador = '1') then
                       ref1         <= var_ref1;
                       ref1_ig_ref2 <= '0';
                       new_calc     <= '1';
                       px_estado    <= e5;

                    elsif (comparador = '0' and fim_pwm = '0') then
                      px_estado    <= e6;
                      ref1_ig_ref2 <= '0';

                    end if;
                when e6 =>
                   if(rst = '1') then
                      px_estado  <= e0;

                    elsif (comparador = '1' and fim_pwm = '1') then
                      px_estado  <= e4;

                    else
                      new_calc     <= '0';
                      px_estado    <= e6;
                      ref1_ig_ref2 <= '0';

                    end if;
            end case;
   end if;

end process;
end Arch_pwm;
```

Anexo G – Cálculos em Ponto Fixo no FPGA

Tomando como exemplo a multiplicação entre dois números representados no formato Q4,12, ou seja, um bit de sinal, três bits representando a parte inteira e doze bits representando a parte fracionária do número; esta multiplicação resultará em um número no formato Q8,24.

Para a utilização dos números na formatação Q4,12, é preciso desprezar os bits do número que não serão utilizados nos cálculos. Aplicando a metodologia desenvolvida tem-se que os bits utilizados serão: os bits 27 para identificar o sinal, os bits 26 ao 24 para representar da parte inteira e os bits 23 ao 11 para representar a parte fracionária do número. A Tabela G.1 demonstra a metodologia desenvolvida.

Tabela G.1- Operação de multiplicação de números em ponto fixo na linguagem VHDL

Nº do Bit	1º Parcela (Q4,12)	2º Parcela (Q4,12)	Resultado em 32 bits (Q8,24)	Resultado em 16 bits (Q4,12)
0	F	F	F	-
1	F	F	F	-
2	F	F	F	-
3	F	F	F	-
4	F	F	F	-
5	F	F	F	-
6	F	F	F	-
7	F	F	F	-
8	F	F	F	-
9	F	F	F	-
10	F	F	F	-
11	F	F	F	-
12	I	I	F	**F**
13	I	I	F	**F**
14	I	I	F	**F**
15	S	S	F	**F**
16	-	-	F	**F**
17	-	-	F	**F**
18	-	-	F	**F**
19	-	-	F	**F**
20	-	-	F	**F**
21	-	-	F	**F**
22	-	-	F	**F**
23	-	-	F	**F**
24	-	-	I	**I**
25	-	-	I	**I**
26	-	-	I	**I**
27	-	-	I	S
28	-	-	I	-
29	-	-	I	-
30	-	-	I	-
31	-	-	S	-

Na Tabela G.1 “F” representa os bits da parte fracionária dos números, “I” significa os bits que representam a parte inteira do número e “S” corresponde ao bit de sinal do número.