Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Curso de Graduação em Engenharia Elétrica

Implementação da Técnica PWM Vetorial por meio da Conexão DSP-FPGA

Nayara Brandão de Freitas

Campina Grande - PB
Julho de 2015

NAYARA BRANDÃO DE FREITAS

IMPLEMENTAÇÃO DA TÉCNICA PWM VETORIAL POR MEIO DA CONEXÃO DSP-FPGA

Relatório de Estágio Supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Orientador: Prof. Dr. Alexandre Cunha Oliveira

Campina Grande - PB

Julho de 2015

NAYARA BRANDÃO DE FREITAS

IMPLEMENTAÇÃO DA TÉCNICA PWM VETORIAL POR MEIO DA CONEXÃO DSP-FPGA

Relatório de Estágio Supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Aprovada em Julho de 2015.

BANCA EXAMINADORA

_____

Prof. Dr. Alexandre Cunha Oliveira

UFCG

_____

Prof. Convidado

UFCG

Campina Grande - PB

Julho de 2015

Dedicado a minha família.

Agradecimentos

Agradeço ao professor Alexandre pela orientação durante esse estágio e ao professor Cursino por todos os ensinamentos.

Agradeço aos colegas de curso e de laboratório que de alguma forma contribuíram com esse trabalho.

Agradeço ao meu namorado Louelson pelo apoio em todos os momentos.

"Todas as flores do futuro estão contidas nas sementes de hoje."

Provérbio Chinês

Resumo

O presente relatório descreve as atividades realizadas durante o Estágio Supervisionado no LEIAM (Laboratório de Eletrônica Industrial e Acionamento de Máquinas), que pertence ao Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande (UFCG). O estágio teve por objetivo desenvolver um sistema que possibilite a implementação da técnica PWM vetorial com a finalidade de dar suporte aos experimentos que são realizados pela equipe do LEIAM nas áreas de conversores e acionamento de máquinas.

Palavras-chave: PWM vetorial, DSP, FPGA, conversores.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Objetivos

Várias técnicas de modulação por largura de pulso (*Pulse Width Modulation* - PWM) podem ser utilizadas na geração das tensões pelos conversores. Nos últimos anos, a técnica PWM vetorial (*Space-Vector PWM*) vem ganhando espaço em detrimento da convencional modulação PWM por portadora. Visando aumentar as ferramentas de trabalho do laboratório, objetiva-se desenvolver um sistema capaz de implementar esta técnica de maneira simples e efetiva.

1.2 Justificativa

É possível implementar facilmente a modulação por portadora utilizando um Processador Digital de Sinais (*Digital Signal Processor* - DSP). Tendo em vista isso, quando deseja-se implementar a técnica PWM vetorial, geralmente busca-se utilizar sua equivalência com a modulação por portadora. Entretanto não é possível encontrar a equivalência direta em alguns casos. Com isso, surge a necessidade de desenvolver um sistema que possibilite a implementação direta da técnica PWM vetorial.

Geralmente a implementação de uma técnica PWM por portadora no DSP é realizada da seguinte forma: a tensão de polo de referência é convertida para um valor entre 0 e 7500, o DSP gera uma portadora triangular por meio de um contador que começa em zero, vai até 7500 e depois decresce até voltar a ser zero. O sinal de comando das chaves é dado pela comparação entre a referência e a portadora triangular. Existem duas lógicas

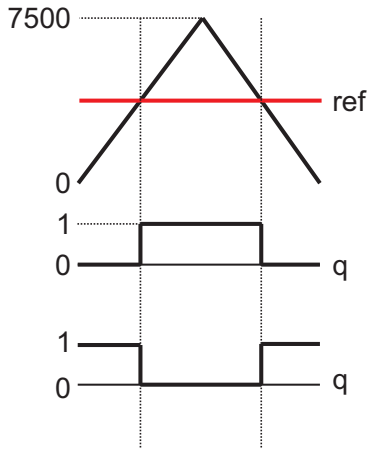

Figura 1.1: Implementação de uma técnica PWM utilizando DSP.

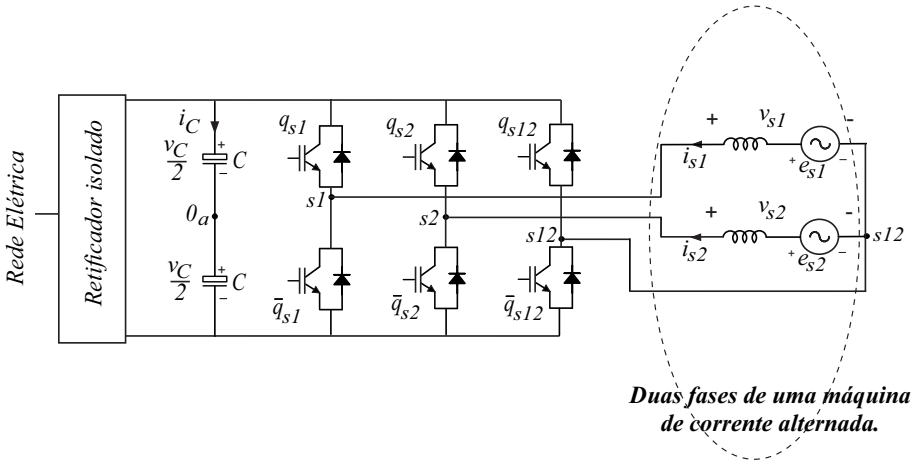

Figura 1.2: Sistema de acionamento estudado.

possíveis para a comparação e elas são apresentas na Figura 1.1 (a) e (b). Como pode ser visto, essa implementação só permite que as chaves mudem de estado uma vez durante o período T da portadora e, como irá ser mostrado, algumas vez a técnica vetorial exige que as chaves mudem de estado mais de uma vez durante o período.

Para ilustrar a dificuldade de geração de sinais PWM usando a técnica vetorial, apresenta-se o estudo do inversor bifásico de três braços utilizado para alimentar duas fases de uma máquina (Figura 1.2). Uma sequência específica de aplicação dos vetores de tensão, encontrada por meio do estudo da técnica LS-PWM, não pode ser implementada utilizando portadoras. Entretanto, com a utilização dessa técnica, as distorções harmônicas das tensões estatóricas são reduzidas e equalizadas, tornando-a interessante de ser implementada.

De forma a realizar essa implementação, um sistema composto por um DSP e um

FPGA (*Field Programmable Gate Array*) é utilizado. O primeiro é responsável por gerar as tensões de referência nas fases da máquina e enviar essas informações serialmente. O segundo, por sua vez, recebe os dados, implementa a técnica vetorial e gera os sinais de comando para os braços do inversor. O sistema desenvolvido pode ser utilizado como base para implementação de qualquer técnica PWM vetorial e seu diagrama da ligação entre os componentes do sistema é apresentado na Figura 1.3.

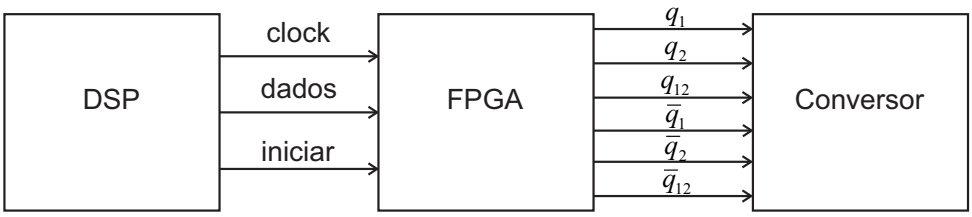

Figura 1.3: Diagrama da ligação DSP-FPGA-Conversor.

1.3 Cronograma de atividades

Foram dedicadas 4h diárias, totalizando 20h semanais, no intervalo de 04/05/2015 a 10/07/2015 (10 semanas). As tarefas realizadas foram divididas da seguinte forma:

- Estudo do conversor e da técnica PWM: 3 semanas.
- Estudo do DSP e da configuração do módulo SPI: 3 semanas.
- Estudo do FPGA e elaboração dos módulos utilizando Verilog: 3 semanas.
- Elaboração do relatório: 1 semana.

Capítulo 2

Laboratório de Eletrônica Industrial e Acionamento de Máquinas - LEIAM

O Laboratório de Eletrônica Industrial e Acionamento de Máquinas (LEIAM) é um dos laboratórios do Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande (UFCG), dispõe de uma infraestrutura laboratorial distribuída numa área ed 330 m^2. A infraestrutura é dividida em quatro ambientes: laboratório de simulação digital, acionamento de máquinas, eletrônica de potência e de qualidade de energia.

O laboratório de simulação digital é equipado com 20 (vinte) computadores. O projeto e desenvolvimento de novas topologias de conversores estáticos é uma das áreas de pesquisa do LEIAM. O de acionamento de máquinas possui bancadas de estrutura flexível para ensaios experimentais e desenvolvimento de sistemas de acionamento de alto desempenho, cada bancada dispõe de um conversor de potência conectado a uma máquina elétrica e um computador com placas de controle e aquisição de dados. O laboratório de eletrônica de potência possui quatro bancadas dotadas de plataformas experimentais para conversores de potência. Os três laboratório de ensaios experimentais são servidos por diversos equipamento de medição: osciloscópios digitais, sondas de tensão e corrente, frequencímetros, multímetros e medidores de potência real.

Capítulo 3

O Inversor de Três Braços e a Técnica PWM Vetorial

3.1 Apresentação do Modelo

O sistema de acionamento estudado (Figura 1.2) é formado por um barramento capacitivo e um inversor de três braços alimentando duas fases de uma máquina. O circuito equivalente simplificado é apresentado na Figura 3.1. Assumindo que as variáveis v_{s1}, v_{s2}, i_{s1} e i_{s2} representam as tensões e correntes estatóricas, respectivamente, e que a máquina é representada por um modelo RLE, as seguintes equações podem ser derivadas para o modelo do sistema utilizando as leis de Kirchhoff

$$v_{s1} = e_{s1} + l_s \frac{di_{s1}}{dt} + r_s i_{s1} \tag{3.1}$$

$$v_{s2} = e_{s2} + l_s \frac{di_{s2}}{dt} + r_s i_{s2} \tag{3.2}$$

Observando a Figura 3.1, as seguintes equações podem ser escritas

$$v_{s1} = v_{s10_a} - v_{s120_a} \tag{3.3}$$

$$v_{s2} = v_{s20_a} - v_{s120_a} \tag{3.4}$$

onde v_{s10_a}, v_{s20_a} e v_{s120_a} são as tensões de polo.

3.2 Técnica PWM Vetorial

As tensões fornecidas pelos inversores podem ser representadas no plano vetorial v_{s1} x v_{s2} (Figura 3.2). Esses planos vetoriais são definidos de forma que as tensões v_{s1}

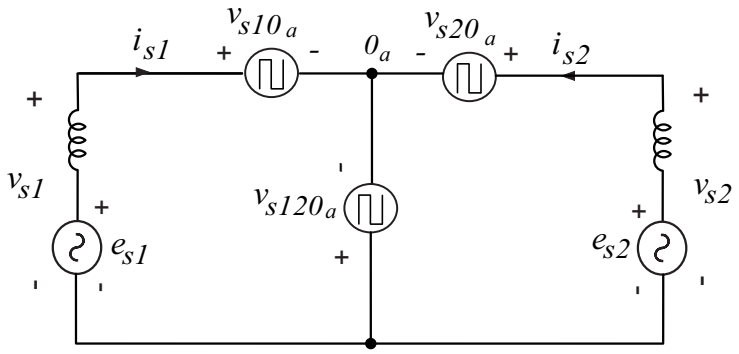

Figura 3.1: Circuito simplificado do conversor estudado.

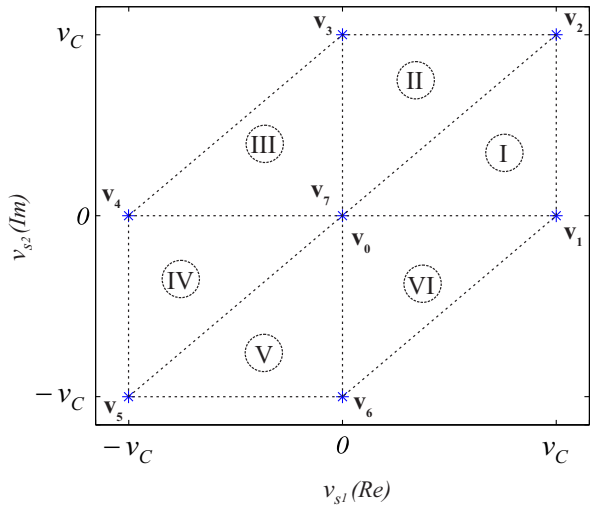

Figura 3.2: Plano vetorial v_{s1} x v_{s2}.

e v_{s2} coincidam com os eixos real e imaginário, respectivamente. Um vetor de tensão no plano pode ser representado por

$$\mathbf{v_n} = v_{s1} + iv_{s2} \tag{3.5}$$

com $n = 0, 1, 2, ..., 7$, como mostrado na Figura 3.2.

Cada triângulo é um setor e cada vértice representa um vetor de tensão. Os vetores de tensão podem ser representados a partir de funções, cujos parâmetros são os estados binários das chaves $[q_{s1}, q_{s2}, q_{s12}]$ são mostrados na Tabela 3.1.

As chaves q_{s1}, q_{s2} and q_{s12} geram oito possíveis combinações de vetores. Dois vetores nulos ($\mathbf{v_0}$ and $\mathbf{v_7}$), quatro vetores com amplitude v_C ($\mathbf{v_1}$, $\mathbf{v_3}$, $\mathbf{v_4}$ e $\mathbf{v_6}$) e dois vetores com amplitude $\sqrt{2}v_C$ ($\mathbf{v_2}$ e $\mathbf{v_5}$). Esses vetores definem os setores I, II, III, IV, V, and VI como mostrado na Figura 3.2. As tensões v_{s1} and v_{s2} podem tomar apenas três valores: v_C, 0, or $-v_C$.

Tabela 3.1: Vetores e tensões geradas no plano v_{s1} x v_{s2}

q_{s1}	q_{s2}	q_{s12}	vector	v_{s1}	v_{s2}
0	0	0	$\mathbf{v_0}$	0	0
1	0	0	$\mathbf{v_1}$	v_C	0
1	1	0	$\mathbf{v_2}$	v_C	v_C
0	1	0	$\mathbf{v_3}$	0	v_C
0	1	1	$\mathbf{v_4}$	-v_C	0
0	0	1	$\mathbf{v_5}$	-v_C	-v_C
1	0	1	$\mathbf{v_6}$	0	-v_C
1	1	1	$\mathbf{v_7}$	0	0

A quantidade de tempo que cada vetor de tensão deve ser aplicado pode ser obtida analiticamente. Considere que $\mathbf{v}^* = v_{s1}^* + iv_{s2}^*$ represente a tensão de referência a ser sintetizada pelo inversor durante um período de amostragem de tamanho T. Como consequência da geometria, o vetor de referência localizado em um setor pode ser sintetizado utilizando os três vetores mais próximos que definem aquele setor. Sejam v_x, v_y e v_z esses vetores.

Para cada setor pode ser escrito que

$$\mathbf{v}^* = \mathbf{v_x}\frac{t_x}{T} + \mathbf{v_y}\frac{t_y}{T} + \mathbf{v_z}\frac{t_z}{T}. \tag{3.6}$$

onde os períodos de aplicação de cada vetor são t_x, t_y e t_z e restritos a $T = t_x + t_y + t_z$.

Utilizando a forma matricial, temos

$$\begin{bmatrix} Re(\mathbf{v_x}) & Re(\mathbf{v_y}) & Re(\mathbf{v_z}) \\ Im(\mathbf{v_x}) & Im(\mathbf{v_y}) & Im(\mathbf{v_z}) \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} v_{s1}^* \\ v_{s2}^* \\ T \end{bmatrix} \tag{3.7}$$

Note que a sequência de aplicação dos vetores pode ser escolhida de diversas formas. Entretanto, existe uma sequência de aplicação dos três vetores de tensão em cada setor que otimiza as tensões aplicadas nas fases da máquina (v_{s1} and v_{s2}) reduzindo as distorções harmônicas. Para qualquer setor mostrado na Figura 3.2, apenas um dos vetores nulos será utilizado.

O vetor de tensão nulo e sequência correta de aplicação dos vetores de tensão para cada setor são mostrados na Figura 3.3 e na Tabela 3.2. Por exemplo, quando o vetor

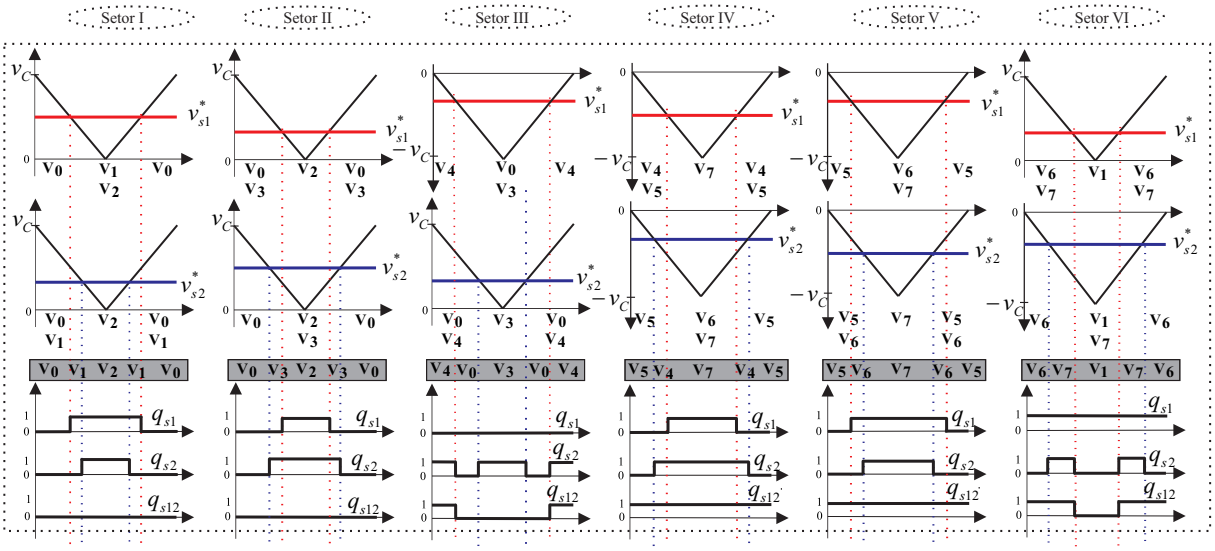

Figura 3.3: Sequência de aplicação dos vetores de tensão.

de referência $\mathbf{v}^*$ está localizado no setor I, os vetores $\mathbf{v_0}$, $\mathbf{v_1}$ e $\mathbf{v_2}$ devem ser utilizados de acordo com a sequência seguinte: $\mathbf{v_0} \rightarrow \mathbf{v_1} \rightarrow \mathbf{v_2} \rightarrow \mathbf{v_1} \rightarrow \mathbf{v_0}$. Quando $\mathbf{v}^*$ é localizado no setor IV, os vetores $\mathbf{v_4}$, $\mathbf{v_5}$ and $\mathbf{v_7}$ devem ser aplicados de acordo com a sequência seguinte: $\mathbf{v_5} \rightarrow \mathbf{v_4} \rightarrow \mathbf{v_7} \rightarrow \mathbf{v_4} \rightarrow \mathbf{v_5}$.

Essa solução foi encontrada observando o comportamento dos vetores durante a utilização da técnica Level-Shifted PWM (LS-PWM). Essa técnica consiste em definir os estados das chaves comparando as tensões de referência com portadoras triangulares de alta frequência com mesma fase e diferentes níveis. A comparação de v_{s1} e v_{s2} com as portadoras seleciona os vetores de tensão que possivelmente poderão ser utilizados. A intersecção dessas possibilidades detecta o vetor de tensão que deve ser aplicado e, por conseguinte, define os estados das chaves. Por isso, chamaremos a técnica apresentada de Vetorial baseada no LS-PWM.

Tabela 3.2: Sequência de aplicação dos vetores de tensão.

Setor	Sequência de aplicação dos vetores de tensão
I	$\mathbf{v_0} \rightarrow \mathbf{v_1} \rightarrow \mathbf{v_2} \rightarrow \mathbf{v_1} \rightarrow \mathbf{v_0}$
II	$\mathbf{v_0} \rightarrow \mathbf{v_3} \rightarrow \mathbf{v_2} \rightarrow \mathbf{v_3} \rightarrow \mathbf{v_0}$
III	$\mathbf{v_4} \rightarrow \mathbf{v_0} \rightarrow \mathbf{v_3} \rightarrow \mathbf{v_0} \rightarrow \mathbf{v_4}$
IV	$\mathbf{v_5} \rightarrow \mathbf{v_4} \rightarrow \mathbf{v_7} \rightarrow \mathbf{v_4} \rightarrow \mathbf{v_5}$
V	$\mathbf{v_5} \rightarrow \mathbf{v_6} \rightarrow \mathbf{v_7} \rightarrow \mathbf{v_6} \rightarrow \mathbf{v_5}$
VI	$\mathbf{v_6} \rightarrow \mathbf{v_7} \rightarrow \mathbf{v_1} \rightarrow \mathbf{v_7} \rightarrow \mathbf{v_6}$

3.3 Resultados de Simulação

O artigo [1] apresenta a técnica PWM Vetorial chamada *Variable Shared-Leg Voltage* e sua modulação equivalente por portadora. Nesta técnica, para cada setor, o tempo de aplicação do vetor nulo é dividido entre os vetores v_0 e v_7 por meio do parâmetro μ. A sequência de aplicação dos vetores é estabelecida de forma que nenhuma chave mude de estado mais de uma vez durante o período T.

As técnicas *Variable Shared-Leg Voltage* (utilizando $\mu = 0$) e Vetorial baseada no LS-PWM foram comparadas em termos da distorção harmônica das tensões nas fases da máquina e da frequência média de chaveamento. Os resultados encontram-se nas tabelas 3.3, 3.4, 3.5 e 3.6 para α igual a 0°, 30°, 60° e 90°. α é o defasamento angular entre os enrolamentos das fases 1 e 2 da máquina. Quando $\alpha = 90°$, o barramento capacitivo é mantido com tensão igual a $1,41pu$, enquanto que, uma tensão igual a $1pu$ é suficiente para $alpha = 0°, 30°, 60°$.

A distorção harmônica foi calculada utilizando a seguinte equação 3.8:

$$WTHD_{(\%)} = \frac{100}{\gamma_1}\sqrt{\sum_{u=2}^{N_u}\left(\frac{\gamma_u}{u}\right)^2} \tag{3.8}$$

onde γ_1 é a componente fundamental da tensão, γ_u é a amplitude do componente harmônico de ordem u e N_u é o número de harmônicos considerado ($N_u = 1000$).

Como resultados tivemos que a técnica Vetorial baseada no LS-PWM tem frequência média de chaveamento um pouco maior do que a *Variable Shared-Leg Voltage*. Isso se dá por conta do chaveamento extra da chave q_{s2} nos setores III e VI. Entretanto, temos que as tensões geradas nas fases da máquina possuem distorções harmônicas menores e mais equalizadas quando comparadas com as geradas pela técnica *Variable Shared-Leg Voltage*. Para α igual a 0°, 30° e 90°, as duas fases possuem mesma distorção harmônica enquanto que, para $alpha = 60°$, a diferença de distorção harmônica nas duas fases é de menos de 6%.

A Figura 3.4 mostra as formas de onda geradas pelo inversor de três braços quando a técnica PWM vetorial é utilizada. Independentemente da sequência de aplicação dos vetores, três níveis de tensão são gerados.

Tabela 3.3: WTHDs das tensões geradas utilizando a técnica *Variable Shared-Leg Voltage* com $\mu = 0$.

α	v_{s1} WTHD (%)	v_{s2} WTHD (%)
0°	0,4152	0,4152
30°	0,3958	0,3359
60°	0,3421	0,3826
90°	0,6758	0,5119

Tabela 3.4: WTHDs das tensões geradas utilizando a técnica Vetorial baseada no LS-PWM.

α	v_{s1} WTHD (%)	v_{s2} WTHD (%)
0°	0,2759	0,2759
30°	0,2759	0,2777
60°	0,2759	0,2920
90°	0,4892	0,4856

Tabela 3.5: Frequência de chaveamento (kHz) dos IGBTs utilizando a técnica *Variable Shared-Leg Voltage* com $\mu = 0$.

α	$f_{q_{s1}}$	$f_{q_{s2}}$	$f_{q_{s12}}$	f_{med}
0°	5,04	5,04	4,98	5,02
30°	7,14	7,05	5,82	6,67
60°	6,72	6,66	6,66	6,68
90°	6,30	6,24	7,47	6,67

Tabela 3.6: Frequência de chaveamento (kHz) dos IGBTs utilizando a técnica Vetorial baseada no LS-PWM.

α	$f_{q_{s1}}$	$f_{q_{s2}}$	$f_{q_{s12}}$	f_{med}
0°	9,99	9,99	0,06	6,68
30°	8,43	11,67	1,68	7,26
60°	6,75	13,35	3,36	7,82
90°	5,10	15,06	5,07	8,41

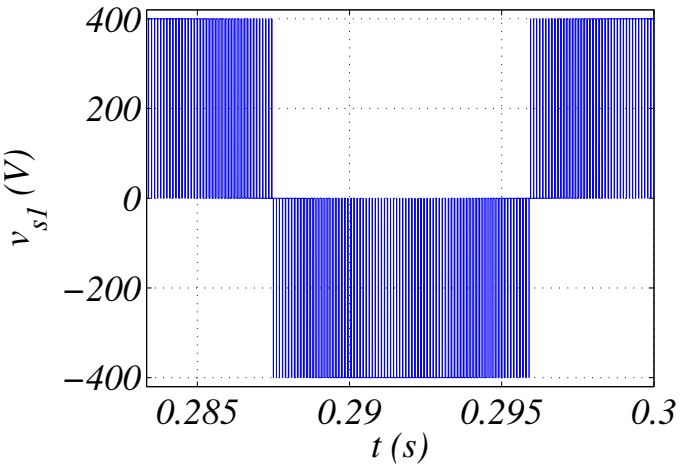

Figura 3.4: Forma de onda gerada pelo inversor de três braços ao utilizar a técnica vetorial.

Capítulo 4

DSP

4.1 TMS320C28x

O TMS320C28x (aqui chamado de C28x) é um microcontrolador de 32 bits com ponto fixo produzido pela *Texas Instruments* que é especializado em aplicações de controle de alta performance como robótica, automação industrial, dispositivos de armazenamento em massa, iluminação, redes ópticas, conversores e outras aplicações de controle que precisam de um único processador para solucionar aplicações de alta performance. O TMS320F28335, utilizado na parte experimental desse estágio, pertence a família C28x. Esta versão de DSP da família C28x possui uma unidade de Ponto Flutuante em Hardware, que lhe possibilita realizar operações aritméticas de ponto flutuante com grande velocidade.

A arquitetura do C28x pode ser dividida nos seguintes blocos funcionais:

- CPU e barramentos;
- Memória;
- Periféricos.

Múltiplos barramentos são usados para mover dados entre memórias e periféricos e a CPU. Sua memória contem:

- Barramento de leitura de programa (22 bits de endereço e 32 bits de dados);
- Barramento de leitura de dados (32 bits de endereço e 32 bits de dados);

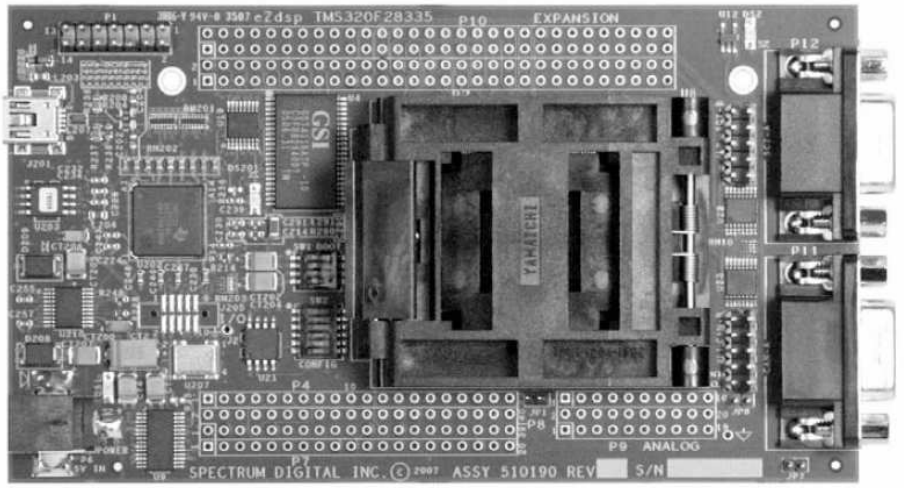

Figura 4.1: Módulo eZdsp^{TM}.

- Barramento de escrita de dados (32 bits de endereço e 32 bits de dados).

Como os barramentos de dados possuem largura igual a 32 bits, é possível realizar operações de 32 bits em um único ciclo. Essa arquitetura de barramentos é conhecida como Harvard Bus Architecture e possibilita o DSP a buscar uma instrução, ler e gravar um dado durante um único ciclo.

O C28x vem com vários periféricos otimizados para apoiar aplicações de controle. Foi utilizado nesse trabalho o periférico Interface Serial Periférica (Serial Peripheral Interface - SPI) para realizar a conexão do DSP a uma placa FPGA (DE2 - ALTERA).

4.2 eZdsp^{TM} F28335

O eZdsp^{TM} (Figura 4.1), produzido pela Spectrum Digital, Inc., é um módulo que permite a avaliação do TMS320F28335. O mesmo é uma excelente plataforma para desenvolver e executar software para o processador TMS320F28335. Para simplificar o desenvolvimento de código e diminuir o tempo de depuração, o C2000 Code Composer Studio^{TM} é fornecido.

O eZdsp^{TM} possui as seguintes características de hardware:

- TMS320F28335 Controlador Digital de Sinais (*Digital Signal Controller* - DSC);
- velocidade de operação de 150 MHz;
- unidade de ponto flutuante de 32-bits;

- 68K bytes de RAM no chip;
- 512K bytes de memória flash no chip;
- 256K bytes de memória SRAM fora do chip;
- conversor analógico-digital (A/D) de 12 bits no chip com 16 canais de entrada;
- clock de entrada de 30 MHz.;
- conector RS-232;
- interface CAN 2.0;
- conectores para múltipla expansão;
- adaptador CA para operação com 5 V;
- conector IEEE 1149.1 JTAG.

Possui as seguintes características de software:

- TI F28xx Code Composer Studio^{TM}, versão 3.3;
- Flash APIs da Texas Instruments;
- arquivos de cabeçalho para o F28335 e exemplos.

4.3 Interface Serial Periférica

A Interface Serial Periférica (Serial Peripheral Interface - SPI) é uma porta serial síncrona de Entrada/Saída que permite a transmissão/recepção serial de palavras de tamanho programado entre o C28x e outro dispositivo periférico. Durante a transmissão, um dispositivo SPI deve ser configurado como mestre e os outros como escravos.

Uma dado a ser transmitido deve ser justificado à esquerda por software enquanto que os dados recebidos são justificados à direita. A SPI contém filas do tipo FIFO (*First In, First Out*) de 16 níveis para transmissão e recepção. O mestre pode começar a transmissão de dados a qualquer instante por controlar o sinal de clock.

A figura 4.2 mostra o diagrama de blocos da SPI. O escravo escreve a palavra a ser enviada em seu registrador de deslocamento SPIDAT. Quando o mestre escreve o dado

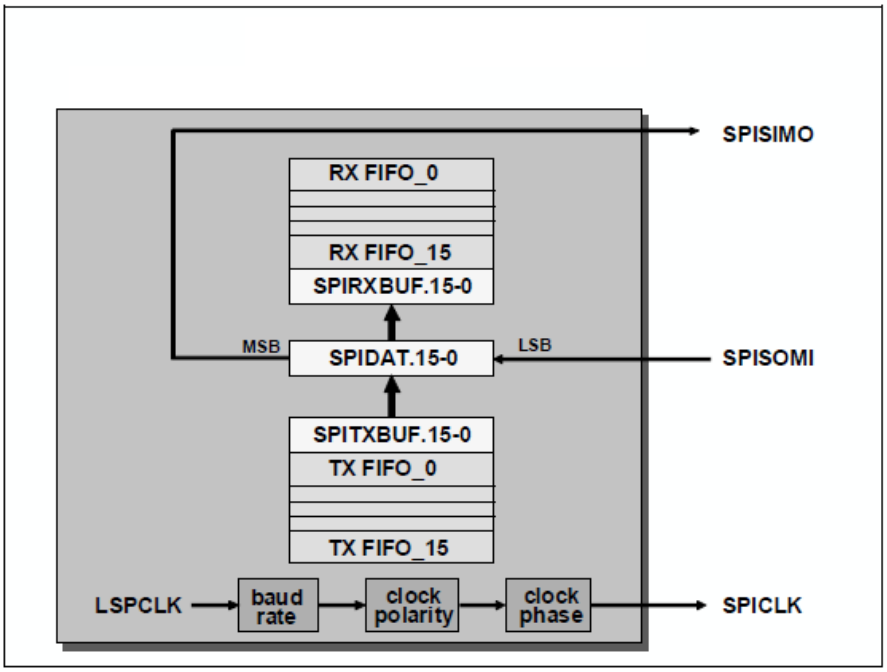

Figura 4.2: Diagrama de blocos da SPI.

a ser transmitido em seu registrador SPIDAT ou SPITXBUF, o sinal de clock SPICLK é iniciado. O bit mais significativo (Most Significant Bit - MSB) do registrador SPIDAT do mestre é deslocado para fora enquanto o bit menos significativo (Least Significant Bit - LSB) da palavra gravada no SPIDAT do escravo é carregado. Isso se repete até que todos os bits da palavra sejam transmitidos. O registrador SPIDAT é copiado para o SPIRXBUF. O bit SPI INT Flag torna-se 1 e, se o SPI INT ENA for 1, uma interrupção acontece. Se ainda existem palavras no SPITXBUF (tanto no do mestre quanto no do escravo), a palavra é carregada no SPIDAT e a transmissão continua assim que o registrador SPIDAT do mestre for carregado.

Existem 125 velocidades de transferência programáveis. A velocidade é determinada pelas seguintes equações (SPIBRR é o registrador):

- Para SPIBRR = 3 a 127: Velocidade de transmissão = $\frac{LSPCLK}{SPIBRR+1}$ bits/segundo
- Para SPIBRR = 0, 1 ou 2: Velocidade de transmissão = $\frac{LSPCLK}{4}$ bits/segundo

O tamanho da palavra é programável entre 1 e 16 bits. Isso é definido por meio do registrador de configuração SPICCR.3-0. O tamanho da palavra é igual ao número binário formado por esse quatros bits somado com 1.

4.4 Configuração da SPI

Abaixo é apresentado o código em C que realiza a configuração do DSP e de alguns periféricos, entre os quais a porta SPI. As operações realizas pelas funções apresentadas no código são descritas por meio de comentários.

```
// Inicializar arquivos de cabeçalho e exemplos
#include "DSP28x_Project.h"

// Protótipo das funções contidas nesse arquivo.
void delay_loop(void);
void spi_xmit(Uint16 a);
void spi_fifo_tx_init(void);
void spi_init(void);

void main(void)
{

   Uint16 vs1;  // vs2
   Uint16 vs2;  // vs1

/* Inicialização do sistema de controle.
Essa função é encontrada no arquivo DSP2833x_SysCtrl.c.*/
   InitSysCtrl();

/* Inicialização dos pinos da GPIO utilizados durante a transmissão serial.
Essa função é encontrada no arquivo DSP2833x_Spi.c*/
   InitSpiaGpio();

// Desabilitar interrupções da CPU
   DINT;

/* Inicializar os registradores de controle da PIE em seus estados default
(todas interrupções desabilitadas e os flags limpos).
Essa função é encontrada no arquivo DSP2833x_PieCtrl.c*/
   InitPieCtrl();

/* Desabilitar as interrupções da CPU e limpar todos os flags
```

```
 das interrupções da CPU*/
   IER = 0x0000;
   IFR = 0x0000;

/* Inicializar a tabela de vetores PIE
Essa função é encontrada no arquivo DSP2833x_PieVect.c.*/
   InitPieVectTable();

// Inicializações:
   spi_fifo_init();   // Inicializar o FIFO da Spi
   spi_init();        // Inicializar SPI

/* Inicializar o pino GPIO12 como saída (chamaremos de pino "iniciar").
Ele será usado para sinalizar o momento no qual o FPGA deve começar
a aplicar os vetores*/
    EALLOW;
   GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;
   //Configura o pino como propósito geral
   GpioCtrlRegs.GPADIR.bit.GPIO12  = 1;
   //Configura o pino como saída
   EDIS;

// Utilizar código específico ==============================

    //Atribuir valores a vs1 e vs2
   vs1 = 0x4E20;
   vs2 = 0x2710;

   /*Gravar os dados no FIFO de transmissão
   e iniciar a transferência*/
   spi_xmit(vs1);
   spi_xmit(vs2);

   delay_loop();

   //Fazer o pino "iniciar" ser igual a 1
   GpioDataRegs.GPASET.bit.GPIO12 = 1;

   delay_loop();
```

```
   //Fazer o pino "iniciar" ser igual a 0
   GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;

// ===========================================================

}


// Apresentação de todas as funções e interrupções:

//Função delay
void delay_loop()
{
    long      i;
    for (i = 0; i < 1000000; i++) {}
}

//Função que configura a SPI
void spi_init()
{
    SpiaRegs.SPICCR.all =0x006F;
    // reset ligado, dado é enviado na borda de descida
    // modo loop desabilitado, envia dados de 16 bits
    SpiaRegs.SPICTL.all =0x00E4;
    // desabilitar interrupções de overrun do registrador de dados recebidos.
    // dado é enviado na borda de descida do clock
    //spi configurada como mestre
    //desabilitar recebimento de dados
    //desabilitar interrupções da spi
    SpiaRegs.SPIBRR =0x007F;
    //velocidade de transmissão igual a máxima do dsp dividida por 8
    SpiaRegs.SPICCR.all =0x00EF;
    //tirar spi do resest
    SpiaRegs.SPIPRI.bit.FREE = 1;
    //impede que a operação da spi seja suspensa por um breakpoint
}

/*Função que grava o dado "a" no FIFO de transmissão.
```

```
Como a SPI está configurada como mestre,
os dados serão transmitidos até que o FIFO de transmissão seja esvaziado.*/
void spi_xmit(Uint16 a)
{
    SpiaRegs.SPITXBUF=a;
}


/*Função que configura o FIFO de transmissão*/
void spi_fifo_tx_init()
{
    SpiaRegs.SPIFFTX.all=0xFFC0;
    //limpar interrupções do FIFO de transmissão
    //desabilitar interrupções do FIFO de transmissão
    //habilitar FIFO da SPI
    SpiaRegs.SPIFFCT.all=0x0;
    /*não há delay na transferência de palavras do buffer FIFO de
    transmissão para registrador de transmissão de deslocamento*/
}
```

Capítulo 5

FPGA

5.1 A família de FPGAs Cyclone™ II e placa DE2

Os dispositivos Altera® Cyclone II possuem baixo custo e características otimizadas tornando-os boas soluções para aplicações automotivas, comunicação, processamento de vídeo, teste e medições e outras.

A placa DE2 da Altera® (Figura 5.1) é um kit de desenvolvimento para avaliação da família de FPGAs Cyclone™ II. Essa placa utiliza um EP2C35F672C6N em conjunto com uma série de periféricos permitindo o desenvolvimento de uma série de aplicações. Alguns desses periféricos são: conjuntos de diodos emissores de luz, chaves de apertar e de seleção liga/desliga, display LCD de 16x2 linhas, displays de 7 segmentos, porta de comunicação infra-vermelho, conector Ethernet RJ45 e RS232, entre outros.

O projeto apresentado neste trabalho foi programado no FPGA EP2C35F672C6N por meio da ferramenta de síntese lógica Quartus II. Essa ferramenta programa a placa, permite a depuração de erros na descrição Verilog, a simulação do projeto, entre outras funcionalidades. O projeto encapsula informações sobre hierarquia de design, bibliotecas, restrições e configurações de projeto.

5.2 Implementação do PWM Vetorial

Para a implementação do PWM vetorial, foram criados três módulos. O módulo Serial é responsável por receber e armazenar os valores das tensões de referência. Já o módulo Calculos, é responsável por encontrar o setor e calcular os tempos de aplicação

dos vetores. O módulo Aplicacao_Vetores gera os sinais de controle a serem aplicados nos braços do conversor. O diagrama de blocos mostrando como os módulos são conectados é mostrado na Figura 5.2.

Os fluxogramas que apresentam as operações realizadas por cada módulo são mostradas nas Figuras 5.3, 5.4 e 5.5.

Os códigos em Verilog que implementam cada módulo são apresentados abaixo.

```
ï»¿module Serial (clock_in, reset, bit_recebido, vs1, vs2, recebeu_dados);

input clock_in, reset, bit_recebido;

output signed [15:0] vs1, vs2;
output recebeu_dados;

wire clock_in, reset, bit_recebido;

reg signed [15:0] vs1, vs2;
reg recebeu_dados;

reg nova_palavra;
reg [4:0] contador_bits;
reg [1:0] contador_palavras;
reg signed [15:0] palavra;


//Essa parte do codigo eh executada sempre que ocorrer uma borda de descida
//no clock enviado pelo DSP ou no pino de reset.
always @ (negedge clock_in or negedge reset) begin
    if (!reset) begin
        //Se o reset estiver em 0, todos os contadores sao zerados
        contador_palavras = 0;
        contador_bits = 0;
    end else begin
        //O registrador "palavra" desloca os bits ja recebidos para a
        //direita e armazena o novo bit na posicao mais a esquerda
        palavra = {palavra[14:0],bit_recebido};
        contador_bits = contador_bits + 1;
```

```
        //o numero de bits recebidos eh incrementado
        if (contador_palavras == 2'b10) begin
            //caso duas palavras ja tenham sido recebidas, o contador de
            //palavras eh resetado
            contador_palavras = 2'b00;
        end
        if (contador_bits == 5'b10000) begin
            //caso o registrador "palavra" ja tenha recebido 16 bits, o
            //contador de palavras eh incrementado, o contador de bits
            // eh zerado e o flag nova_palavra muda de estado
            contador_palavras = contador_palavras + 1;
            contador_bits = 0;
            nova_palavra = !nova_palavra;
        end
    end
end

    //esse always eh executado sempre que o flag nova_palavra muda de estado
    //(o que significa que uma nova palavra foi recebida)
always @ (nova_palavra) begin
    //se o contador de palavras for igual a 1, a palavra recebida foi vs1
    //se o contador de palavras for igual a 2, a palavra recebida foi vs2
    case (contador_palavras)
    2'b01: vs1 = palavra;
    2'b10: begin
        vs2 = palavra;
        recebeu_dados = !recebeu_dados;
        //recebeu_dados muda de estado para avisar ao modulo calculos que vs1
        //e vs2 ja foram recebidos e que novos calculos devem ser realizados
    end
    endcase
end


endmodule
```

```
module Calculos (recebeu_dados, vs1, vs2, setor, t0, t1, t2, t3, t4, t5, t6,
 t7);
```

```
    input signed [15:0] vs1, vs2;
    input recebeu_dados;

    output [2:0] setor;
    output [30:0] t0, t1, t2, t3, t4, t5, t6, t7;

    wire signed [15:0] vs1, vs2;
    wire recebeu_dados;

    reg [2:0] setor;
    reg signed [30:0] t0, t1, t2, t3, t4, t5, t6, t7;

    reg signed [14:0] constante = 15'b010011100010001;


    //esse always eh executado sempre que recebeu_dados dados muda de estado
    //ou seja, quando vs1 e vs2 acabaram de ser recebidos
always @ (recebeu_dados) begin
     t0 = 0;
      t1 = 0;
      t2 = 0;
      t3 = 0;
      t4 = 0;
      t5 = 0;
      t6 = 0;
      t7 = 0;
      //os setores e tempo de aplicacao dos vetores sao encontados
      //e serao utilizados posteriormente pelo modulo Aplicacao_Vetores
      if (vs1 > 0 && vs2 > 0 && vs1 > vs2) begin
      setor = 1;
      t1 = (vs1-vs2)*constante >>> 16;
      t2 = vs2*constante >>> 16;
      t0 = 5000 - t1 - t2;
      end
      else if (vs1 > 0 && vs2 >= 0 && vs1 <= vs2) begin
      t2 = vs1*constante >>> 16;
      t3 = (vs2-vs1)*constante >>> 16;
      t0 = 5000 - t2 - t3;
```

```
      setor = 2;
      end
      else if (vs1 <= 0 && vs2 > 0) begin
      t3 = vs2*constante >>> 16;
      t4 = -vs1*constante >>> 16;
      t0 = 5000 - t3 - t4;
      setor = 3;
      end
      else if (vs1 < 0 && vs2 <= 0 && vs1 < vs2) begin
      t4 = (vs2-vs1)*constante >>> 16;
      t5 = -vs2*constante >>> 16;
      t7 = 5000 - t4 - t5;
      setor = 4;
      end
      else if (vs1 < 0 && vs2 < 0 && vs1 >= vs2) begin
      t5 = -vs1*constante >>> 16;
      t6 = (vs1-vs2)*constante >>> 16;
      t7 = 5000 - t5 - t6;
      setor = 5;
      end
      else if (vs1 >= 0 && vs2 < 0) begin
      t6 = -vs2*constante >>> 16;
      t1 = vs1*constante >>> 16;
      t7 = 5000 - t6 - t1;
      setor = 6;
      end
end

endmodule
```

```
module Aplicacao_Vetores (clk, iniciar, setor, t0, t1, t2,
 t3, t4, t5, t6, t7, q_out, q_out_);

input clk;
input iniciar;
input [2:0] setor;
input signed [30:0] t0, t1, t2, t3, t4, t5, t6, t7;
```

```
output [2:0] q_out;
output [2:0] q_out_;

wire clk;
wire iniciar;
wire [2:0] setor;
wire signed [30:0] t0, t1, t2, t3, t4, t5, t6, t7; //13:0

reg [2:0] q_out;
reg [2:0] q_out_;

reg signed [30:0] tx1, tx2, tx3, tx4;
reg signed [30:0] cont;

reg [2:0]  q_setor1_x1, q_setor1_x2, q_setor1_x3, q_setor1_x4, q_setor1_x5;
reg [2:0]  q_setor2_x1, q_setor2_x2, q_setor2_x3, q_setor2_x4, q_setor2_x5;
reg [2:0]  q_setor3_x1, q_setor3_x2, q_setor3_x3, q_setor3_x4, q_setor3_x5;
reg [2:0]  q_setor4_x1, q_setor4_x2, q_setor4_x3, q_setor4_x4, q_setor4_x5;
reg [2:0]  q_setor5_x1, q_setor5_x2, q_setor5_x3, q_setor5_x4, q_setor5_x5;
reg [2:0]  q_setor6_x1, q_setor6_x2, q_setor6_x3, q_setor6_x4, q_setor6_x5;

reg [2:0]  qx1, qx2, qx3, qx4, qx5;

    // nesse always os registradores recebem a sequencia de estados
    //das chaves para cada setor
    always @ * begin

      q_setor1_x1 = 3'b000;
      q_setor1_x2 = 3'b100;
      q_setor1_x3 = 3'b110;
      q_setor1_x4 = 3'b100;
      q_setor1_x5 = 3'b000;

      q_setor2_x1 = 3'b000;
      q_setor2_x2 = 3'b010;
      q_setor2_x3 = 3'b110;
      q_setor2_x4 = 3'b010;
      q_setor2_x5 = 3'b000;
```

```
      q_setor3_x1 = 3'b011;
      q_setor3_x2 = 3'b000;
      q_setor3_x3 = 3'b010;
      q_setor3_x4 = 3'b000;
      q_setor3_x5 = 3'b011;

      q_setor4_x1 = 3'b001;
      q_setor4_x2 = 3'b011;
      q_setor4_x3 = 3'b111;
      q_setor4_x4 = 3'b011;
      q_setor4_x5 = 3'b001;

      q_setor5_x1 = 3'b001;
      q_setor5_x2 = 3'b101;
      q_setor5_x3 = 3'b111;
      q_setor5_x4 = 3'b101;
      q_setor5_x5 = 3'b001;

      q_setor6_x1 = 3'b101;
      q_setor6_x2 = 3'b111;
      q_setor6_x3 = 3'b100;
      q_setor6_x4 = 3'b111;
      q_setor6_x5 = 3'b101;

    end

    //esse always eh executado na borda de descida do clock de 50 MHz
always @ (negedge clk) begin
        //quando o DSP envia iniciar = 1, cont Ã© zerado e o FPFA guarda nos
        //registradores o tempo e a sequencia de aplicacao dos vetores
      if (iniciar) begin
      cont = 0;
      case (setor)
      1: begin
          qx1 = q_setor1_x1;
          qx2 = q_setor1_x2;
          qx3 = q_setor1_x3;
          qx4 = q_setor1_x4;
          qx5 = q_setor1_x5;
```

```
            tx1 = t0 >>> 1;
            tx2 = tx1 + (t1 >>> 1);
            tx3 = tx2 + t2;
            tx4 = tx3 + (t1 >>> 1);
        end
        2: begin
            qx1 = q_setor2_x1;
            qx2 = q_setor2_x2;
            qx3 = q_setor2_x3;
            qx4 = q_setor2_x4;
            qx5 = q_setor2_x5;
            tx1 = t0 >>> 1;
            tx2 = tx1 + (t3 >>> 1);
            tx3 = tx2 + t2;
            tx4 = tx3 + (t3 >>> 1);
        end
        3: begin
            qx1 = q_setor3_x1;
            qx2 = q_setor3_x2;
            qx3 = q_setor3_x3;
            qx4 = q_setor3_x4;
            qx5 = q_setor3_x5;
            tx1 = t4 >>> 1;
            tx2 = tx1 + (t0 >>> 1);
            tx3 = tx2 + t3;
            tx4 = tx3 + (t0 >>> 1);
        end
        4: begin
            qx1 = q_setor4_x1;
            qx2 = q_setor4_x2;
            qx3 = q_setor4_x3;
            qx4 = q_setor4_x4;
            qx5 = q_setor4_x5;
            tx1 = t5 >>> 1;
            tx2 = tx1 + (t4 >>> 1);
            tx3 = tx2 + t7;
            tx4 = tx3 + (t4 >>> 1);
        end
        5: begin
```

```
        qx1 = q_setor5_x1;
        qx2 = q_setor5_x2;
        qx3 = q_setor5_x3;
        qx4 = q_setor5_x4;
        qx5 = q_setor5_x5;
        tx1 = t5 >>> 1;
        tx2 = tx1 + (t6 >>> 1);
        tx3 = tx2 + t7;
        tx4 = tx3 + (t6 >>> 1);
    end
    6: begin
        qx1 = q_setor6_x1;
        qx2 = q_setor6_x2;
        qx3 = q_setor6_x3;
        qx4 = q_setor6_x4;
        qx5 = q_setor6_x5;
        tx1 = t6 >>> 1;
        tx2 = tx1 + (t7 >>> 1);
        tx3 = tx2 + t1;
        tx4 = tx3 + (t7 >>> 1);
    end
    endcase
end
//cont eh comparado com os registradores que armazenam o momento no qual
//cada vetor deve comeÃ§ar a ser aplicado
if (cont <= tx1) begin
q_out <= qx1;
q_out_ <= ~qx1;
end
else if (cont <= tx2) begin
q_out <= qx2;
q_out_ <= ~qx2;
end
else if (cont <= tx3) begin
q_out <= qx3;
q_out_ <= ~qx3;
end
else if (cont <= tx4) begin
q_out <= qx4;
```

```
    q_out_ <= ~qx4;
    end
    else begin
    q_out <= qx5;
    q_out_ <= ~qx5;
    end

    cont = cont + 1; //cont Ã© incrementado

    //caso cont seja maior que 5000, ele deve deve ser resetado
    if (cont > 5000)
        cont = 0;

    end

    endmodule
```

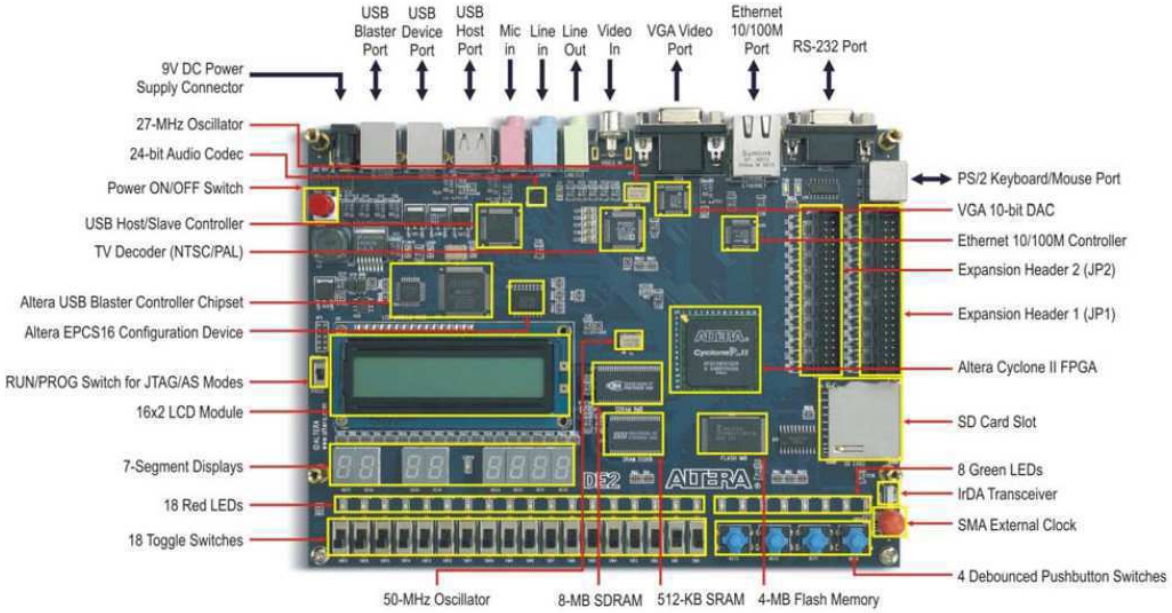

Figura 5.1: Placa DE2.

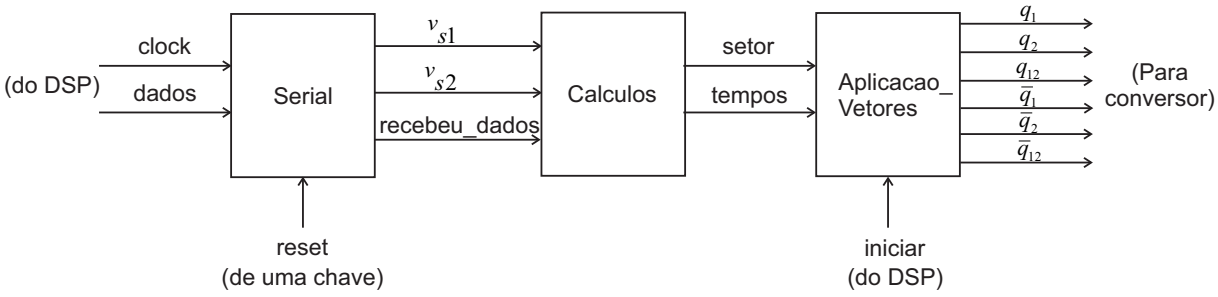

Figura 5.2: Diagrama de blocos dos módulos feitos no FPGA.

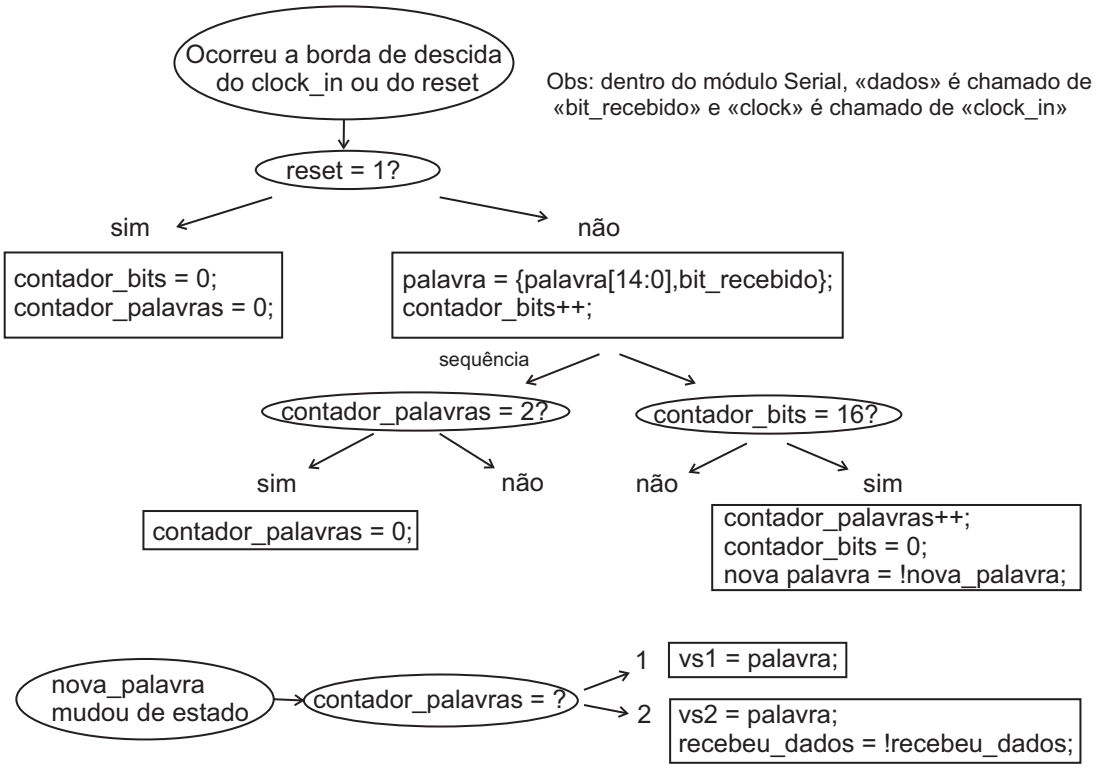

Figura 5.3: Esquema do módulo "Serial".

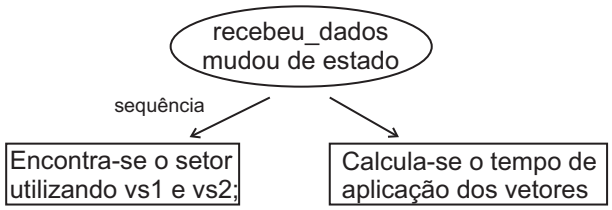

Figura 5.4: Esquema do módulo "Calculos".

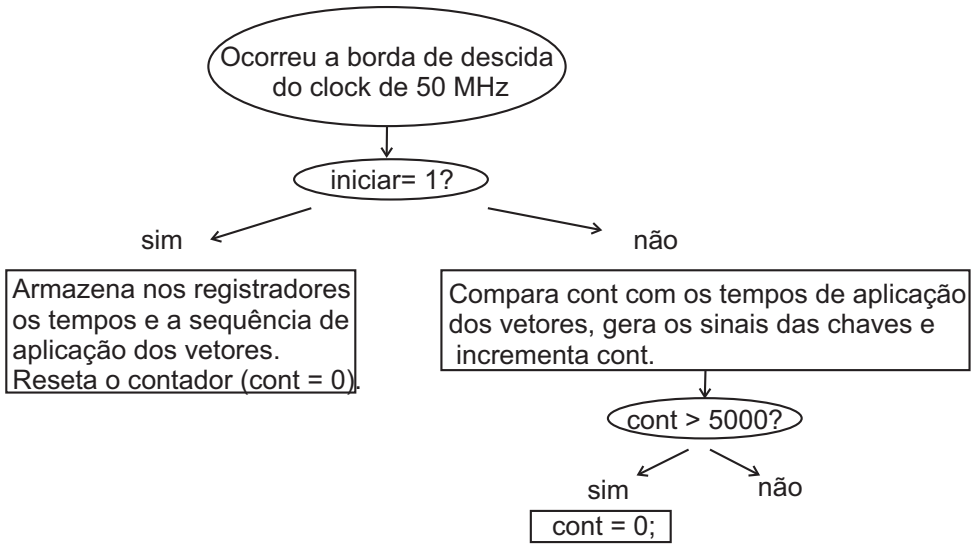

Figura 5.5: Esquema do módulo "Aplicacao_Vetores".

Capítulo 6

Resultados

A Tabela 6.1 mostra as tensões de referência e o tempo de aplicação dos vetores para cada setor durante os experimentos. t_m é o tempo de aplicação do vetor v_m onde $m = 0, 1, 2, 3, 4, 5, 6, 7$ e o período T é igual a $100\mu s$. As Figuras 6.1-6.6 mostram de cima para baixo os sinais de controle das chaves q_1, q_2 e q_{12}. Comparando essas figuras com a Figura 3.3, vemos que a sequência de aplicação de vetores e as larguras de pulso estão corretas.

Tabela 6.1: Tensões de referência e tempo de aplicação dos vetores ($T = 100\mu s$).

	Setor I	Setor II	Setor III	Setor IV	Setor V	Setor VI
v^*_{s1}	0,61 pu	0,305 pu	-0,549 pu	-0,61 pu	-0,305 pu	0,549 pu
v^*_{s2}	0,305 pu	0,763 pu	0,153 pu	-0,305 pu	-0,763 pu	-0,153 pu
Tempos	$t_1 = 30,5\mu s$	$t_2 = 30,5\mu s$	$t_3 = 15,3\mu s$	$t_4 = 30,5\mu s$	$t_5 = 30,5\mu s$	$t_6 = 15,3\mu s$
	$t_2 = 30,5\mu s$	$t_3 = 45,8\mu s$	$t_4 = 54,9\mu s$	$t_5 = 30,5\mu s$	$t_6 = 45,8\mu s$	$t_1 = 54,9\mu s$
	$t_0 = 39,0\mu s$	$t_0 = 23,7\mu s$	$t_0 = 29,8\mu s$	$t_7 = 39,0\mu s$	$t_7 = 23,7\mu s$	$t_7 = 29,8\mu s$

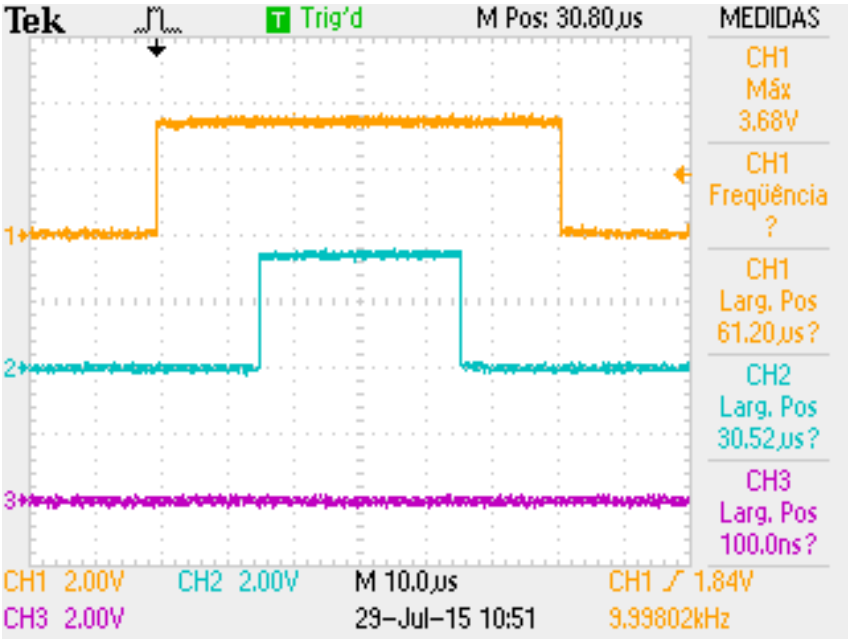

Figura 6.1: Setor I.

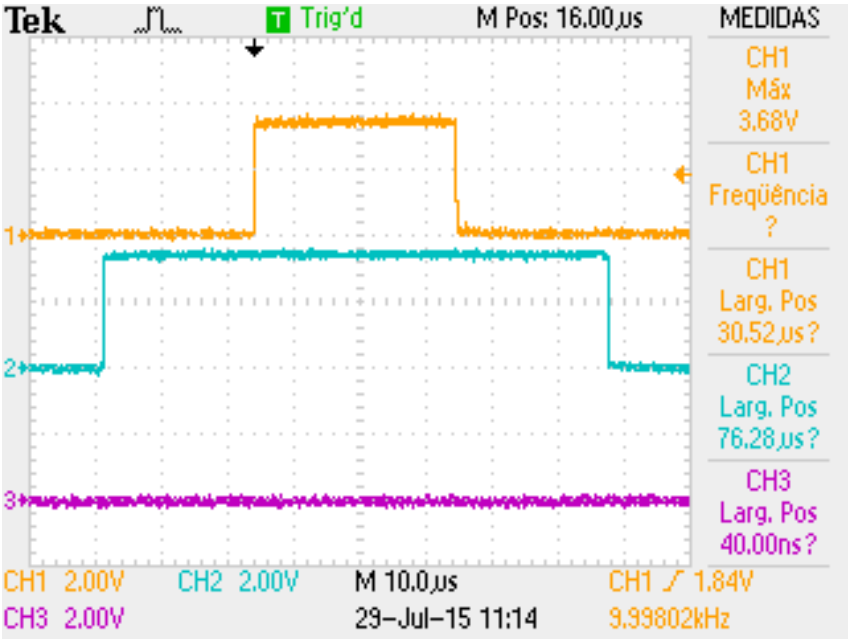

Figura 6.2: Setor II.

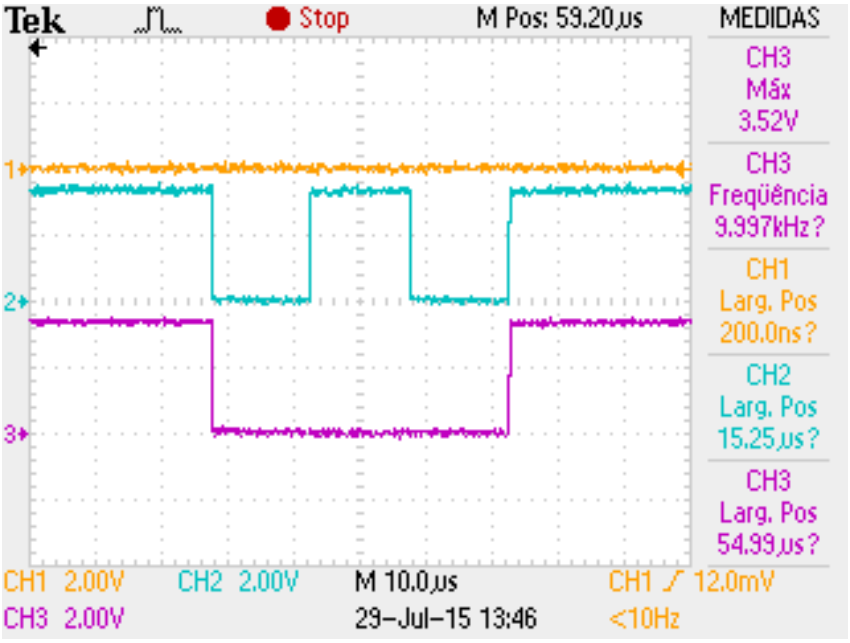

Figura 6.3: Setor III.

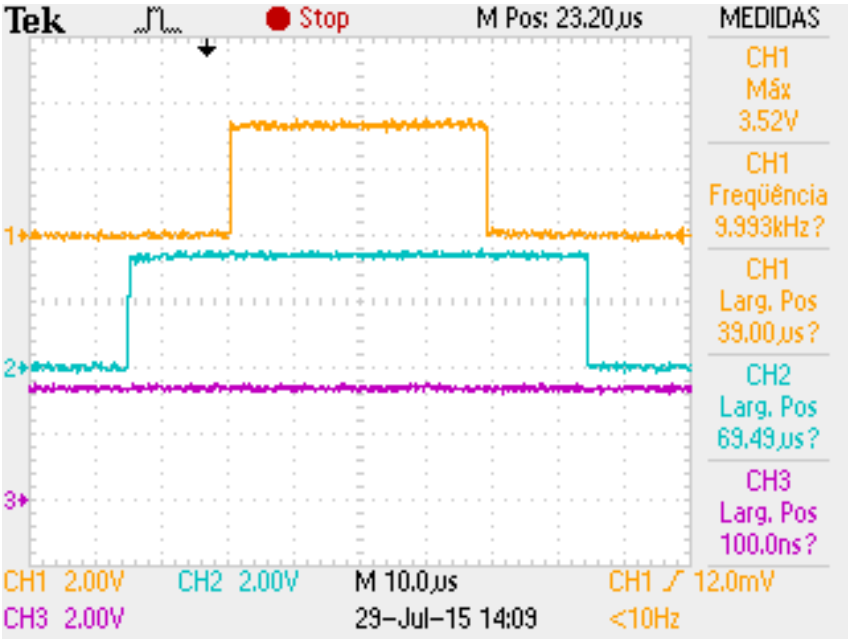

Figura 6.4: Setor IV.

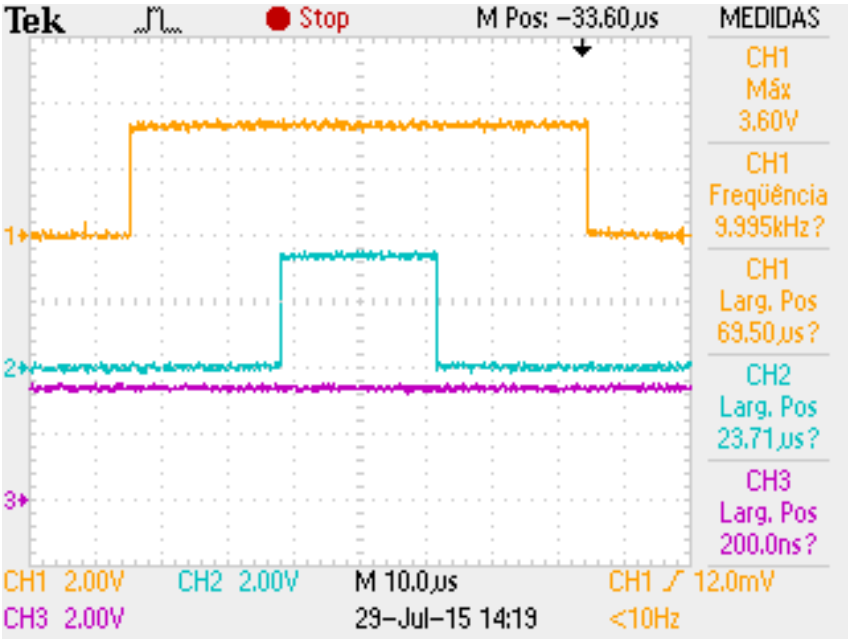

Figura 6.5: Setor V.

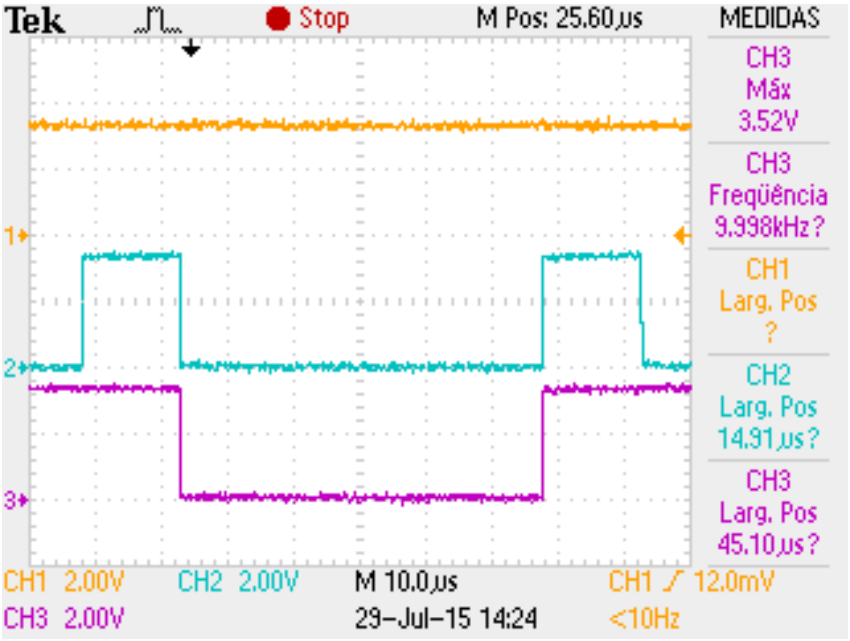

Figura 6.6: Setor VI.

Capítulo 7

Conclusão

O trabalho desenvolvido abordou vários temas estudados durante a graduação, em especial os assuntos das disciplinas Eletrônica de Potência e Arquitetura de Sistemas Digitais. Foram estudadas as ferramentas DSP e FPGA na implementação de técnicas PWM.

Quando a técnica de modulação pode ser realizada por meio de portadoras triangulares, o DSP pode ser utilizado. Entretanto, nem sempre é possível encontrar uma equivalência por portadoras para todas as técnicas desenvolvidas. Como forma de motivação, foi mostrada uma sequência de aplicação de vetores de tensão da técnica PWM vetorial que gera bons resultados de simulação e que não possui modulação por portadora equivalente. Nesse caso, procurou-se implementar a técnica vetorial propriamente dita.

O DSP foi utilizado para gerar as tensões de referência e enviar esses dados para o FPGA por meio de uma comunicação serial. O módulo SPI do DSP foi estudado bem como sua configuração.

O FPGA foi utilizado para receber os valores das tensões de referência, encontrar os setores no plano vetorial, calcular os tempos de aplicação de cada vetor e gerar os comandos para cada braço do conversor de forma a aplicar os vetores de tensão corretos nas fases da máquina.

Foi visto que o sistema é uma solução viável para implementar a configuração estudada e que, utilizando os mesmos princípios, pode-se implementar qualquer outra técnica PWM vetorial. Com isso, outros membros do laboratório poderão utilizar esse sistema para implementar as técnicas que desenvolverem.

Referências Bibliográficas

[1] C.B Jacobina, T.M. Oliveira e E.R.C. da Silva, *Control of the single-phase three-leg AC/AC converter*, Industrial Electronics, IEEE Transactions on , vol.53, no.2, pp.467,476, April 2006 doi: 10.1109/TIE.2006.870655.

[2] TMS320C28^{TM} MCU Workshop. *Workshop Guide and Lab Manual*, Texas Instruments, 2009.

[3] TMS320F28335, TMS320F28334, TMS320F28332 TMS320F28235, TMS320F28234, TMS320F28232 Digital Signal Controllers (DSCs). *Data Manual*, Texas Instruments, 2011.

[4] TMS320x2833x, 2823x System Control and Interrupts. *Reference Guide*, Texas Instruments, 2010.

[5] TMS320x2833x, 2823x Serial Peripheral Interface (SPI). *Reference Guide*, Texas Instruments, 2009.

[6] Code Composer Studio Development Tools v3.3. *Getting Started Guide*, Texas Instruments, 2006.

[7] eZdsp^{TM} F28335. *Technical Reference*, Spectrum Digital, Inc., 2007.

[8] DE2 Development and Education Board. *User Manual*, Altera Corporation, 2006.

[9] Section I. *Cyclone II Device Family Data Sheet*, Altera Corporation, 2005.

[10] Quartus II Handbook Volume 1. *Design and Synthesis*, Altera Corporation, 2015.