UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA

PROJETO EM ENGENHARIA ELÉTRICA:

GERADOR DE SINAL PWM UTILIZANDO MÚLTIPLOS RECORTES

ALUNO: Luciano de Macedo Barros

MATRÍCULA: 20621509

PROFESSOR ORIENTADOR: Prof. Alexandre Cunha Oliveira

CAMPINA GRANDE, JUNHO DE 2009

AGRADECIMENTOS

Agradeço primeiro a Deus, pela oportunidade de viver, buscando um maior crescimento moral e espiritual.

À minha mãe (Hermelinda) e a meu pai (Luzimar) que me deram todo o apoio, estrutura, confiança, carinho e amor, ao longo da minha vida. Aprendi com eles a ser uma pessoa boa. Aprendi o significado da palavra amar. Aprendi a ser um homem íntegro e honesto. Aprendi e aprendo muito com eles, todos os dias, que apesar da distância, sempre estiveram presentes, mesmo através de uma ligação ou em uma visita surpresa. Tenho neles a minha referência.

À minha segunda mãe (Nilza) por todo amor dedicado a mim nesses 25 anos de vida. Às minhas irmãs (Luzianne e Luciana) e a meu irmão (Luzemberg), por toda paciência, ajuda e amor nessa minha atual existência.

A meu avô (Lucemar Barros). Homem visionário, com quem aprendi ensinamentos para toda vida. Em quem me espelho a cada dia para conquistar as virtudes e a sabedoria desse homem.

Ao amigo/professor Alexandre Cunha, com o qual em quase 4 anos de convívio, pude usufruir de uma excelente companhia. Tenho nele exemplo de obstinação, esforço e responsabilidade. Agradeço pelos puxões de orelhas, paciência, enfim por toda a ajuda.

A obreira de última hora, minha namorada (Tarsila Lívia), por ter chegado no momento exato, me dando carinho, amor, amizade, estando sempre presente. Com quem eu encontrei e encontro, por diversas vezes, a calma e o sorriso de que tanto preciso.

A todos os meus amigos e amigas, que tanto me ajudaram e me acolheram aqui em Campina Grande. Após uma saída tão precoce de casa, estando longe da família, fiz aqui nesta cidade, irmãos(ãs).

Agradeço em especial aos amigos Lucas Vinícius Hartman e a Antonio Agripino, pela importantíssima ajuda prestada ao longo do desenvolvimento do projeto.

Agradeço a todos os professores pelos ensinamentos, dados dentro ou fora da sala de aula. As secretárias do departamento que tem para conosco, carinho e atenção de mães.

SUMÁRIO

1. INTRODUÇÃO

1.1 Considerações Gerais

Sinais modulados em largura de pulso, comumente denominados sinais PWM (do inglês, *Pulse Width Modulation*), são amplamente utilizados em aplicações de eletrônica de potência. De forma geral, a amplitude de um sinal de tensão/corrente é representada pela largura do sinal PWM. A associação da largura do sinal, ao estado ligado ou desligado de chaves de potência de um circuito de potência, representa a essência da operação de conversores no âmbito da eletrônica de potência.

A geração de sinais PWM de forma digital tem sido o método preferencialmente utilizado, dado à disponibilidade e baixo custo de dispositivos eletrônicos, dedicados ou não, com capacidade de geração de sinais PWM. Os sinais PWM podem ser gerados digitalmente através da modulação escalar (Jacobina et al., 1997) ou através da modulação vetorial (Jacobina et al., 1999). Na modulação vetorial, as tensões a serem aplicadas a carga polifásica são geradas a partir de um conjunto de vetores de tensão definidos em função do estado das chaves do inversor.

Assim, no caso de sistemas trifásicos há um conjunto de 8 vetores e no caso de sistemas hexafásicos há um conjunto de 64 vetores.

A utilização de sistemas hexafásicos tem se tornado um tema de grande interesse em função do aumento da demanda por sistemas de acionamento para máquinas de elevada potência, alimentadas com alta tensão de modo a reduzir a corrente dos enrolamentos. Várias estratégias de modulação para inversores de 6 braços têm sido apresentadas na literatura técnica(Julian et al., 1996)(Bakhshai et al., 1998). As variações de implementação entre as estratégias estão relacionadas à aplicação de técnicas que visam minimizar flutuações na corrente nas fases da máquina, eliminação de harmônicos e minimização/eliminação da tensão de modo comum (Manjrekar and Lipo, 1999)(Oriti et al., 1997)(Correa et al., 2003a)(Correa et al., 2003b).

Como os vetores de tensão de referência são gerados a partir dos vetores de tensão associados ao estado das chaves do inversor, e como no sistema hexafásico há um aumento no número desses vetores, passa a existir a necessidade de geração de padrões de chaveamento onde

podem ocorrer mais de duas transições no estado da chave durante um período do sinal PWM. No caso de sistemas trifásicos e utilizando a modulação PWM com pulsos centrados, ocorrem apenas duas transições durante um período do sinal PWM. Além disso, todos os hardwares dedicados à geração de sinais PWM, encontrados em DSP's e microcontroladores, são preparados para atender, no máximo, a esse número de transições durante um período PWM.

A falta de dispositivos de hardware dedicados e com capacidade de atender a demanda de sinais PWM com múltiplos pulsos durante um período do sinal PWM, tem levado ao uso de dispositivos FPGA que implementem essa tarefa (Mondal et al., 2007)(Shu et al., 2007)(Jian et al., 2007).

Neste trabalho será apresentado um gerador de sinais PWM implementado a partir de um dispositivo FPGA contido na placa DE2 da ALTERA, utilizando a linguagem Verilog. Através da técnica desenvolvida, é possível gerar sinais PWM com alta resolução e também gerar padrões PWM com múltiplos pulsos em um mesmo período do sinal PWM. Com esta técnica é possível atender aos requisitos de geração de sinais de sistemas hexafásicos, sem restrições de sequência de chaveamento, bem como gerar sinais com rebatimento de pulsos, sem que seja necessário diminuir a frequência de chaveamento.

1.2 Modulação por Largura de Pulso

A modulação por largura de pulso de um sinal ou em fontes de alimentação envolve a modulação de sua razão cíclica (*duty cycl*e) para transportar qualquer informação sobre o canal de comunicação ou controlar o valor da alimentação entregue a carga. Tomaremos como base um interruptor (qualquer dispositivo com abertura e fechamento muito rápido, como por exemplo FET de potência) que controla o fluxo de energia para uma carga, logo, de acordo com o valor da potência que se deseja fornecer a carga, este interruptor receberá sinais de abertura e fechamento.

Ocorre com esta abertura e fechamento a geração de pulsos (pulsos de largura variável). A largura de cada pulso é definida pelo tempo em que a chave ficar fechada. Obtemos com isto, o

chamado "ciclo ativo", que é a relação entre o tempo em que ocorre o pulso e a duração do ciclo completo.

Em qualquer estratégia de PWM, o objetivo primário é determinar o tempo em que a chave deve permanecer fechada (ou o seu estado complementar, chave aberta) para se ter a tensão ou corrente de saída desejadas. Como objetivos secundários, deve-se buscar a obtenção de um processo de chaveamento que minimize as perdas por chaveamentos e que não introduza harmônicas de baixa freqüência as tensões/correntes geradas.

Para os estudos que serão desenvolvidos ao longo do trabalho, será usado o inversor ponte trifásica mostrado na Fig. 1. Utilizando a implementação digital do PWM, o inversor gera tensões instantâneas cujo valor médio, em um intervalo de tempo T (onde T é o período de chaveamento), é igual a tensão de referência. O ponto intermediário "O" será um dos referênciais de tensão.

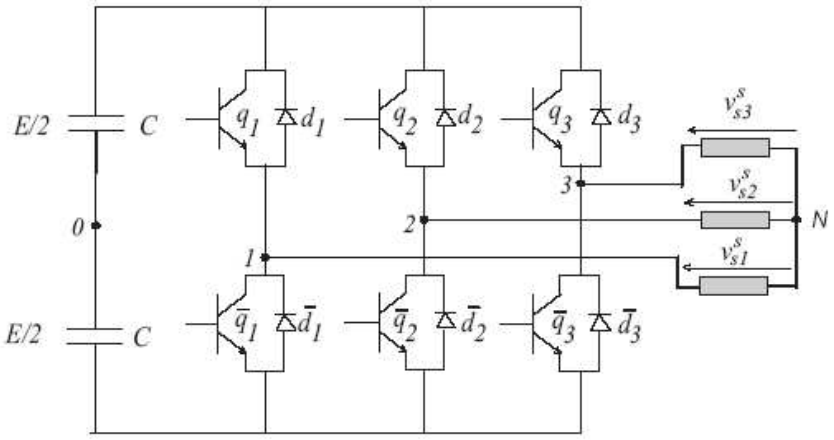

Fig. 1. Inversor trifásico.

1.3 Modulação PWM a partir de Circuitos Eletrônicos

A partir de circuitos eletrônicos é possível gerar pulsos através da comparação entre duas formas de onda (senoidal e triangular). Neste caso, a onda triangular será a portadora e o sinal que se deseja modular será o sinal modulante. Ao aplicar os dois sinais nos terminais do

comparador, como indicado na Fig. 2, a saída do mesmo oscilará entre 0 e 1 lógico, conforme a regra indicada abaixo:

$$V_t > V_r,\ s = 0;$$

$$V_t < V_r,\ s = 1;$$

Quando a modulante tiver maior valor que a portadora, a chave vai estar conduzindo, caso contrário, a chave se encontrará aberta.

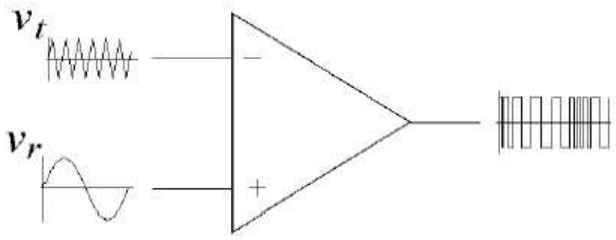

Fig. 2. Modulação por Largura de Pulso.

A seguir, descreveremos o circuito que gera os sinais de comando das chaves. Na eletrônica analógica existe um componente chamado amplificador operacional que é muito utilizado e bastante versátil, com diversas aplicações em eletrônica e que possui ainda uma função muito útil para a implementação da lógica de controle da estratégia PWM. Na entrada negativa do amplificador operacional é inserido a forma de onda triangular e na entrada positiva uma senóide. Ocorre então uma comparação entre estas formas de onda, a fim de definir o instante do pulso de sinal de controle.

Quando a senóide tiver maior amplitude que o valor da triangular ocorre a saturação. Quando isso ocorre, algumas das interseções entre a portadora e o sinal modulante são perdidas,

como resultado, alguns pulsos deixam de existir na forma de onda de saída, como indicado na Fig. 3.

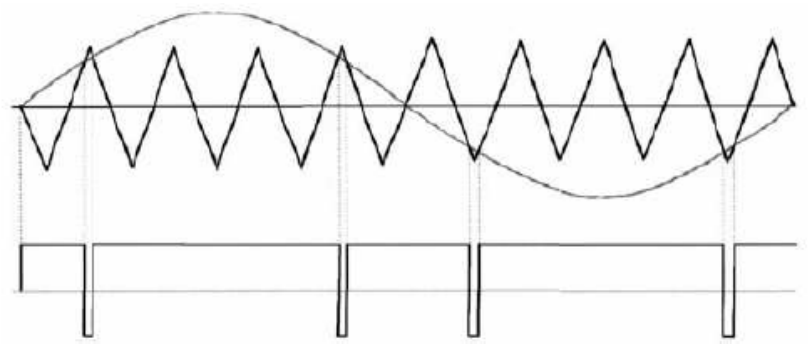

Fig. 3. Modulação por Largura de Pulso.

1.4 Modulação Escalar

Pode-se calcular os tempos de operação das chaves diretamente com as tensões trifásicas. Através dos tempos de condução das chaves, pode-se impor uma tensão média em cada fase, durante um período τ. Sendo $V_{S1}^{S*} + V_{S2}^{S*} + V_{S3}^{S*}$ as tensões de referência que se quer obter, para se calcular os tempos da modulação escalar, utiliza-se as tensões de referência de pólo V_{S10}^{S*}, V_{S20}^{S*} e V_{S30}^{S*}, dadas por:

$$V_{S10}^{S*} = V_{S1}^{S*} + V_n \quad (1)$$

$$V_{S20}^{S*} = V_{S2}^{S*} + V_n \quad (2)$$

$$V_{S30}^{S*} = V_{S3}^{S*} + V_h \quad (3)$$

onde V_h é uma parcela homopolar comum a todas as fases.

Como o vetor V_{h1} é comum à todas as tensões, então, o vetor tensão resultante independe dele, dependendo apenas de V_{S1}^{S*}, V_{S2}^{S*} e V_{S3}^{S*}. Os tempos em que as chaves ficam ON e OFF são determinados pela igualdade entre as tensões V_{S10}^{S*}, V_{S20}^{S*} e V_{S30}^{S*}, sendo constantes no intervalo τ, e os valores médios para as tensões instantâneas de pólo correspondentes $\bar{V}_{S10}^{S}$, $\bar{V}_{S20}^{S}$ e $\bar{V}_{S30}^{S}$.

Definindo-se os períodos (larguras de pulso) τ_1, τ_2 e $\tau_2$como sendo o intervalo de tempo em que as chaves q_1, q_2 e q_3, respectivamente, estão fechadas, e $\tau_1 - \tau, \tau_2 - \tau$ e $\tau_3 - \tau$ como sendo o intervalo de tempo em que as chaves q_1, q_2 e q_3 , respectivamente, estão abertas. Abaixo temos as igualdades para os valores médios:

$$\frac{1}{\tau}\int_0^{\tau} V_{S10}^{S*}\, dt = V_{S10}^{S*} = \frac{1}{\tau}\int_0^{\tau} V_{S10}^{S}\, dt = \bar{V}_{S10}^{S} = \left[\frac{E}{2}\tau_1 - \frac{E}{2}(\tau - \tau_1)\right]\frac{1}{\tau} \quad (4)$$

$$\frac{1}{\tau}\int_0^{\tau} V_{S20}^{S*}\, dt = V_{S20}^{S*} = \frac{1}{\tau}\int_0^{\tau} V_{S20}^{S}\, dt = \bar{V}_{S20}^{S} = \left[\frac{E}{2}\tau_2 - \frac{E}{2}(\tau - \tau_2)\right]\frac{1}{\tau} \quad (5)$$

$$\frac{1}{\tau}\int_0^{\tau} V_{S30}^{S*}\, dt = V_{S30}^{S*} = \frac{1}{\tau}\int_0^{\tau} V_{S30}^{S}\, dt = \bar{V}_{S30}^{S} = \left[\frac{E}{2}\tau_3 - \frac{E}{2}(\tau - \tau_2)\right]\frac{1}{\tau} \quad (6)$$

as variáveis com uma barra correspondem aos seus respectivos valores médios.

A partir das expressões de (4)-(6), chega-se as relações para os tempos T_1, T_2 e T_3 :

$$T_1 = \left(\frac{V_{S10}^{S*}}{E} + \frac{1}{2}\right) T \tag{7}$$

$$T_2 = \left(\frac{V_{S20}^{S*}}{E} + \frac{1}{2}\right) T \tag{8}$$

$$T_3 = \left(\frac{V_{S30}^{S*}}{E} + \frac{1}{2}\right) T \tag{9}$$

1.5 Modulação Vetorial

Outra maneira de determinarmos a largura do pulso é pela forma vetorial, que tem como base o modelo fasorial no plano a,b,c.

A técnica PWM baseada na teoria de vetores espaciais, também denominada modulação vetorial, é especificada para aplicação no caso de conversores trifásicos. Ela toma como ponto de partida a definição de vetores de tensão associados a cada uma das oito configurações possíveis das chaves do inversor.

As oito (2^3) possíveis configurações do inversor referem-se às combinações de abertura e fechamento das três chaves da metade superior do inversor. Os estados 111 (três chaves superiores fechadas) ou 000 (três chaves inferiores fechadas) representam o vetor nulo na origem do plano, os outros seis vetores são denominados de vetores ativos.

Outro ponto distinto da modulação vetorial está em sua característica operacional, onde um padrão de chaveamento é definido apenas pelos tempos de aplicação de seus vetores ativos. Assim, a liberdade para estabelecer os intervalos de aplicação dos vetores nulos permite a geração de padrões adequados a diversas aplicações.

O inversor pode produzir três tensões de saída independentes V_1, V_2 e V_3. Cada tensão pode apresentar dois níveis, dependendo de qual chave esteja na condução. Sua variação de tensão assume valores médios entre o pólo e o ponto médio entre os capacitores de -E/2 e +E/2, sendo E o valor da tensão no barramento CC. Tendo-se três tensões, pode ser representado um vetor no plano definido pelos eixos abc, deslocados 120^o um do outro como mostra a Fig. 4.

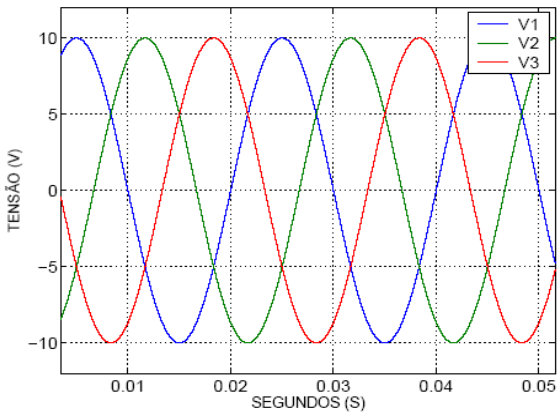

Fig. 4. Tensões de entrada.

Pode-se também representar os estados do inversor por vetores. Na Fig. 5 temos para o estado 110.

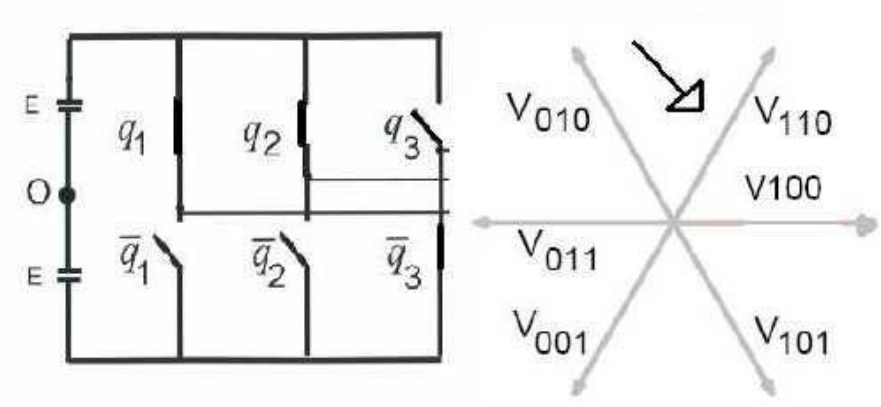

Fig. 5. Representação dos estados do inversor.

Então, em cada período de comutação, ocorre a modulação vetorial, gerando uma sequência de diferentes estados do inversor (é comum a sequência ser formada por três vetores, sendo um destes o vetor nulo).

A soma das larguras de pulso relativas a cada estado deve satisfazer à restrição:

$$t_0 + t_x + t_{x\text{-}60} = T \tag{10}$$

Sendo T o período total. Para se ter a saída desejada de tensões médias no inversor, é necessário obter o vetor resultante V*. Faz-se então uma análise dos vetores ativos adjacentes ao vetor V*, sendo estes vetores e o vetor nulo, aqueles que deverão ser aplicados para produzir as tensões desejadas.

As projeções de V* nos vetores ativos adjacentes determinam os tempos de aplicação de cada vetor ativo, e por conseguinte as respectivas razões cíclicas, enquanto a duração do vetor nulo é determinada pela expressão:

$$t_0 = T - t_x - t_{x\text{-}60} \tag{11}$$

Pode-se utilizar diferentes estratégias para criar os vetores necessários como mostram as Fig.7 a Fig. 9, ambas para obtenção do vetor V* da Fig. 6. Na estratégia 1, o estado de V_1 é mantido constante (V_1 = 1, grampeamento de fase) e comum aos dois vetores, durante todo o período de comutação. A única alteração ocorre em V_2 e V_3.

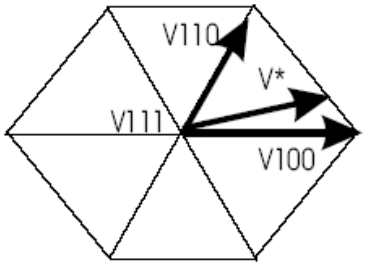

Fig. 6. Estratégia de obter o vetor V*.

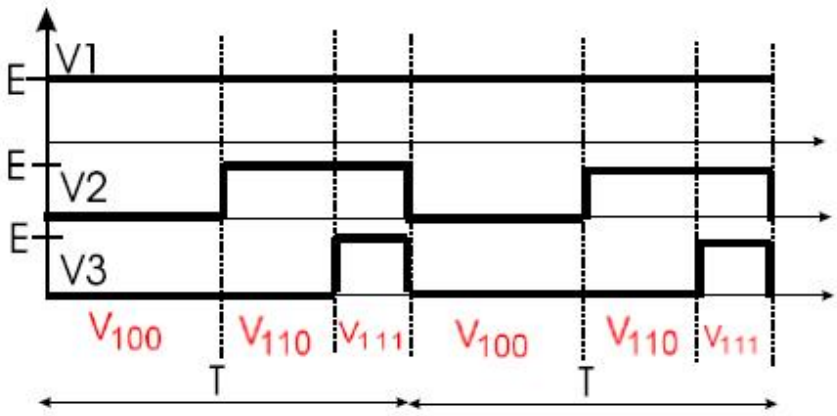

Fig. 7. Estratégia 1 para obtenção do vetor V*.

Na estratégia 2, temos uma minimização das comutações, reduzindo as perdas do conversor. Vemos que V_1 está sempre em 1 (grampeamento de fase), como na estratégia 1, mas cada período adjacente é repetido, não alterando o estado anterior das chaves.

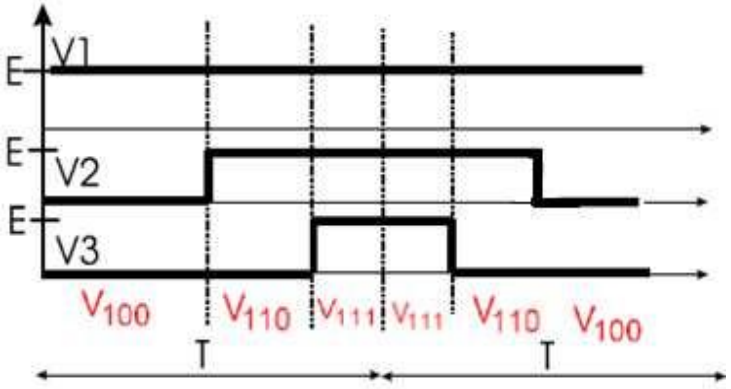

Fig. 8. Estratégia 2 para obtenção do vetor V*.

Na estratégia 3, os vetores 111 e 000 formam o estado nulo. Sua principal característica está no fato de os pulsos de cada fase estarem centrados exatamente na passagem de um ciclo de chaveamento para o outro. Nesta estratégia, fazendo-se a amostragem das correntes elétricas nos pólos do inversor (fases do inversor), no momento de transição de um ciclo para o próximo, obtém-se os respectivos valores médios das correntes.

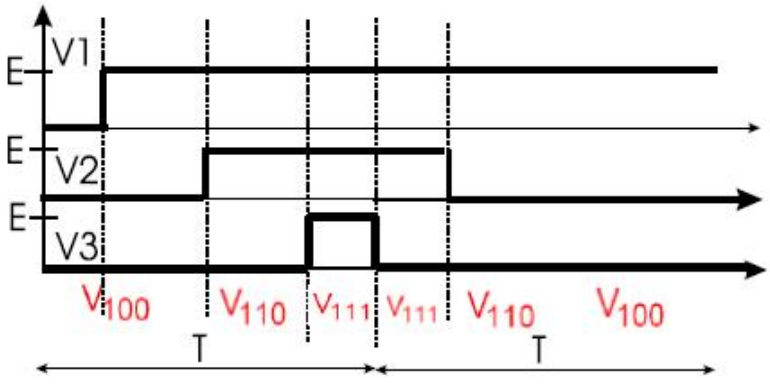

Fig. 9. Estratégia 3 de obtenção do vetor V*.

2. DISTRIBUIÇÃO DOS PULSOS PWM EM UM CICLO PWM

Pesquisas recentes em acionamentos de elevada potência, tais como tração ferroviária ou propulsão de navios, por exemplo, através de conversores eletrônicos, vêm apontando algumas limitações das chaves de potência quanto à capacidade de potência e frequência de chaveamento.

Nos acionamentos de baixa e média potência, onde a frequência de chaveamento pode ser maior, fenômenos não desejáveis do acionamento trifásico, como a tensão de modo comum e a oscilação do conjugado aparecem, como causa do aumento do número de falhas nos mancais. A tensão de modo comum (vh) produz efeitos danosos nos mancais do motor. Outra conseqüência deste problema é o aparecimento de ruídos audíveis nos motores que são bastante indesejáveis em ambientes como "cleanrooms". Outro problema, que acontece nas máquinas multifases, é o surgimento de correntes harmônicas de circulação entre os enrolamentos estatóricos quando alimentadas por inversores fonte de tensão (VSI - Voltage Source Inverter), provocando o surgimento de perdas suplementares, sobredimensionamento dos dispositivos de potência. Esses efeitos colaterais, observados no uso de inversores de freqüência indicam a necessidade de desenvolver-se estratégias de controle PWM que contornem de maneira satisfatória estes efeitos e suas conseqüências.

Com uma modulação PWM adequada, reduz-se ou até mesmo se elimina a tensão de modo comum que atua nas capacitâncias parasitas dos mancais, evitando seu desgaste prematuro (o desgaste mecânico dos mancais é atribuído às correntes que fluem nos próprios mancais e à tensão no eixo da máquina).

As soluções para o desgaste dos mancais têm se concentrado na diminuição da tensão de modo comum pelo fato de não precisarem de equipamento adicional nem tampouco de mudanças na estrutura da máquina.

Isto se torna possível com os recortes em um único período (Gerador de Sinal PWM Utilizando Múltiplos Recortes), diminuindo ou eliminando totalmente a tensão de modo comum e a oscilação do conjugado. A dificuldade de se fazer essa distribuição dos pulsos com o sistema atual está, basicamente, na questão de tempo para programação dos *timers* que impedem a programação de todos os pulsos recortados em um período PWM.

Serão apresentados os casos de geração de sinais PWM que apresentam múltiplos pulsos por ciclo PWM, bem como as restrições dos circuitos disponíveis para geração desses sinais PWM. Os casos que serão apresentados, baseiam-se nos padrões PWM definidos em (Correa et al., 2003a), os quais visam minimizar/eliminar a geração da tensão de modo comum. A eliminação da tensão de modo comum, no acionamento de máquinas hexafásicas é alcançado através da aplicação de vetores que apresentem tensão de modo comum nula.

A aplicação dos mesmos pode ser realizada de modo que o valor médio da tensão de modo comum seja nulo em um período do sinal PWM, ou que a mesma seja instantaneamente nula. Na tabela 1 são mostradas as sequências de vetores para cada um dos 12 setores em que a tensão de referência pode ser posicionada. Na tabela 2 é apresentada a codificação que identifica o estado das chaves q1, q3, q5, q2, q4 e q6 do inversor hexafásico. Esta codificação é utilizada na indicação dos vetores mostrados na tabela 1, no formato **v***xy*, onde *x* representa o estado das chaves q1, q3 e q5, e *y* representa o estado das chaves q2, q4 e q6, respectivamente. Na Fig. 10 são apresentadas as curvas do sinal PWM aplicadas as chaves superiores do inversor, quando uma tensão de referência está localizada no setor 1.

Tabela 1. Sequência de vetores para posicionamento da tensão de referencia.

S30	*t0/2*	t´m/2	t2m/2	tm	t2m/2	t´m/2	t0/2
1	V07	V25	V26	V16	V11	V61	V70
2	V07	V36	V26	V21	V11	V12	V70
3	V07	V12	V22	V21	V31	V36	V70
4	V07	V23	V22	V32	V31	V41	V70
5	V07	V41	V42	V32	V33	V23	V70
6	V07	V52	V42	V43	V33	V34	V70
7	V07	V34	V44	V43	V53	V63	V70
8	V07	V45	V44	V54	V53	V63	V70
9	V07	V63	V64	V54	V55	V45	V70
10	V07	V14	V64	V65	V55	V56	V70
11	V07	V56	V66	V65	V15	V14	V70
12	V07	V61	V66	V16	V15	V25	V70

Tabela 2. Valores máximos e mínimos das indutâncias por fase.

x/y	*0*	1	2	3	4	5	6	7
q1/q2	0	1	1	0	0	0	1	1
q3/q4	0	0	1	1	1	0	0	1
q5/q6	0	0	0	0	1	1	1	1

Observa-se nos sinais de comando das chaves q3 e q2 três transições em um mesmo período do sinal PWM. Este número de transições não é realizável se for utilizado o módulo gerador de sinal PWM encontrado nos DSP's dedicados a aplicações de acionamento, pois os módulos realizam no máximo duas comparações, quando são configurados para gerar sinais PWM simétricos ou no modo "Space Vector" (Timer de propósito geral programado para gerar contagens "Up/Down").

Em sistemas não dedicados, é padrão o uso de contadores e comparadores para geração de sinais PWM, cujo princípio de funcionamento é similar ao implementado pelo bloco PWM dos DSP's. Nesses circuitos, é possível implementar mais de duas comparações por ciclo PWM, no entanto, será necessário reprogramar os valores de comparação durante o período do sinal PWM.

A reprogramação envolve acesso a dispositivos de I/O e portanto consomem tempo, que em algumas rotinas de controle não existe sobrando. Assim, a implementação de um circuito gerador PWM com a capacidade de gerar mais do que duas transições por ciclo do sinal PWM geralmente faz uso de dispositivos de hardware periféricos, sendo atualmente comum o uso de dispositivos FPGA (Mondal et al., 2007)(Shu et al., 2007)(Jian et al., 2007). Na implementação da eliminação da tensão de modo comum de modo instantâneo em (Correa et al., 2003a), também são usados padrões que apresentam sinais PWM com mais do que três transições por ciclo do sinal PWM. Na Fig. 11 é mostrada a sequência de sinais PWM aplicados as chaves, quando deseja-se gerar uma tensão de referência localizada no setor 1, utilizando o caso 4, cujos vetores são apresentados nas tabelas 5 e 6 em (Correa et al., 2003a).

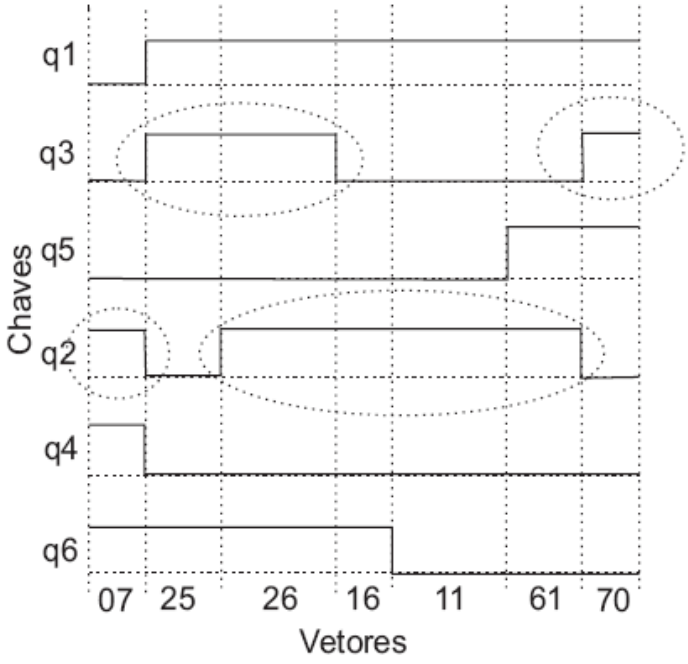

Figura 10. Sinais PWM das chaves para o setor 1, caso 2.

Pode ser observado em todos os sinais PWM mais do que duas transições, além disso, pode-se chamar a atenção para um outro aspecto dos sinais PWM, há dois padrões, aplicados em períodos consecutivos do sinal PWM. Com a implementação utilizando o FPGA da DE2 da Altera, todos os pulsos poderiam ser gerados em um mesmo período do sinal PWM, aumentando a frequência de chaveamento, que pode minimizar as componentes harmônicas de alta frequência da corrente.

Na Fig. 12, são apresentadas as curvas dos sinais PWM gerados a partir dos vetores do caso 5, listados na tabela 7 em (Correa et al., 2003a). Os vetores do caso 5 são convenientes quando se trabalha com baixos índices de modulação. A discrepância entre a tensão gerada e a de referência é minimizada pelo fato de serem utilizados nesse caso apenas vetores de pequena amplitude. De modo equivalente aos casos anteriores, observa-se no sinal PWM da chave q6 a existência de quatro transições durante o período do sinal PWM.

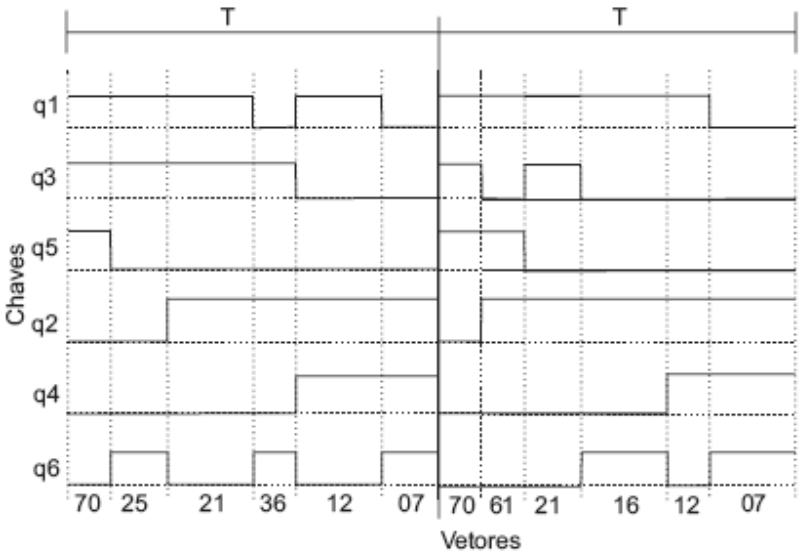

Figura 11. Sinais PWM das chaves para o setor 1.

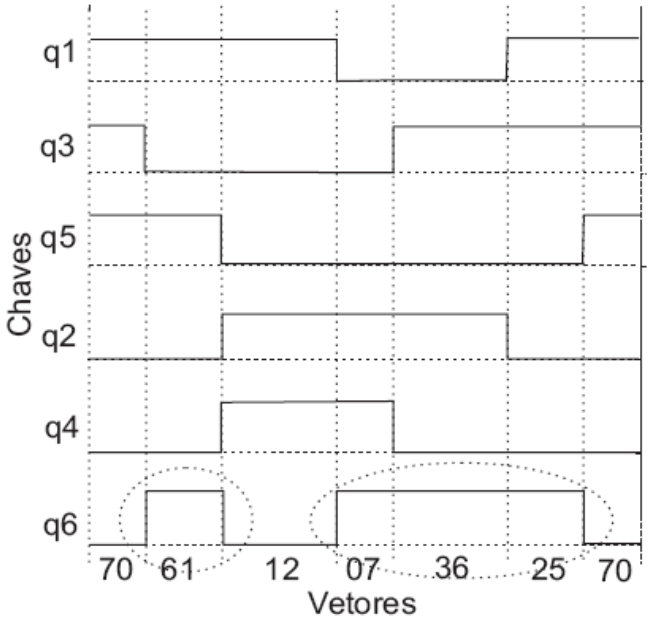

Figura 12. Sinais PWM das chaves para o setor 1, caso 5.

3. A PLACA DE DESENVOLVIMENTO

Nessa etapa do trabalho, forneceremos uma visão geral acerca da placa de desenvolvimentos de projetos digitais com auxílio de um FPGA, Altera DE2 Board, exemplificando sua configuração física, características especiais e possibilidades de uso e implementação.

Também serão conhecidos os passos necessários para se utilizar a ferramenta de síntese lógica Quartus II, para a partir de uma descrição de um projeto digital em Verilog, programar o FPGA.

Para a realização do projeto, será utilizada a placa DE2 – *Development Education Board*, fabricada pela Altera (www.altera.com). Essa placa é versátil no sentido de fornecer uma grande capacidade de desenvolvimento de projetos e um vasto apoio no âmbito educacional, apresentando uma vasta gama de exemplos e materiais de suporte aos alunos e professores, que podem ser encontrados no *website* da Altera ou no CD que acompanha o pacote da placa.

Na parte de hardware, a placa DE2 é equipada com vários dispositivos que a tornam capaz de implementar desde simples projetos digitais, até complexos projetos multimídias. Listemos seus dispositivos:

- Altera Cyclone® II 2C35 FPGA device.
- Dispositivo de configuração serial da Altera – EPCS16
- USB Blaster (on board)
- 512-Kbytes SRAM, 8-Mbytes SDRAM, 4-Mbytes Flahs memory
- SD Card socket
- 8 displays de 7 segmentos
- Display LCD 16x2.
- 4 botões seletores
- 18 chaves digitais
- 18 leds vermelhos
- 9 leds verdes
- Osciladores de 50-MHz e um segundo de 27-MHz, como fontes de clock.

- CODEC de 24-bits, com qualidade de CD, com saídas e entradas de som e microfone.
- VGA DAC (digital analogic conversor) com conexão VGA-vídeo.
- TV Decoder (NTSC/PAL) e conector de entrada para TV.
- Controlador 10/100 Ethernet, com conectores.
- Controlador USB Host/Slave, com conectores tipo A e B.
- RS-232 Transceiver e conector de 9 pinos (DB-9)
- PS/2 mouse/keyboard connector.
- IrDA Transceiver (Transmissor InfraVermelho)
- Dois expansores de 40 pinos, com proteção de diodo.

Nas Figs. 13 e 14 podem-se observar uma foto da placa e a distribuição dos dispositivos de hardware na mesma.

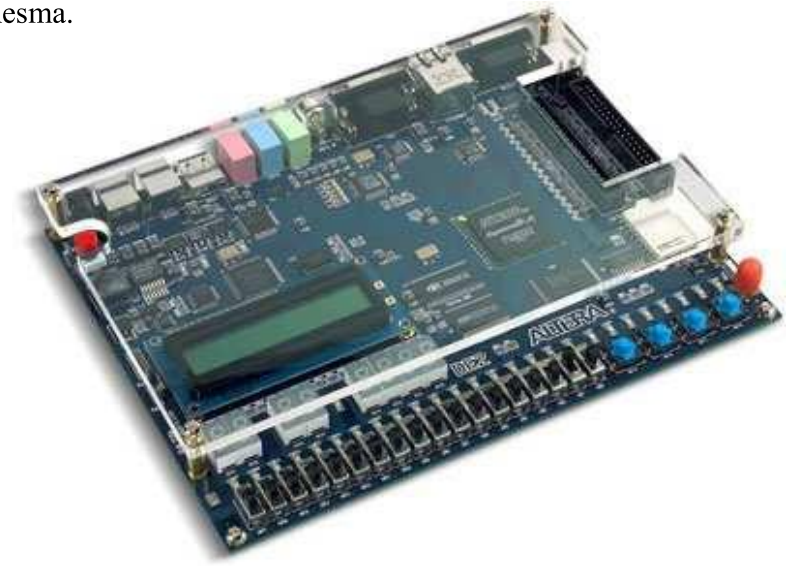

Fig. 13. Vista geral da placa.

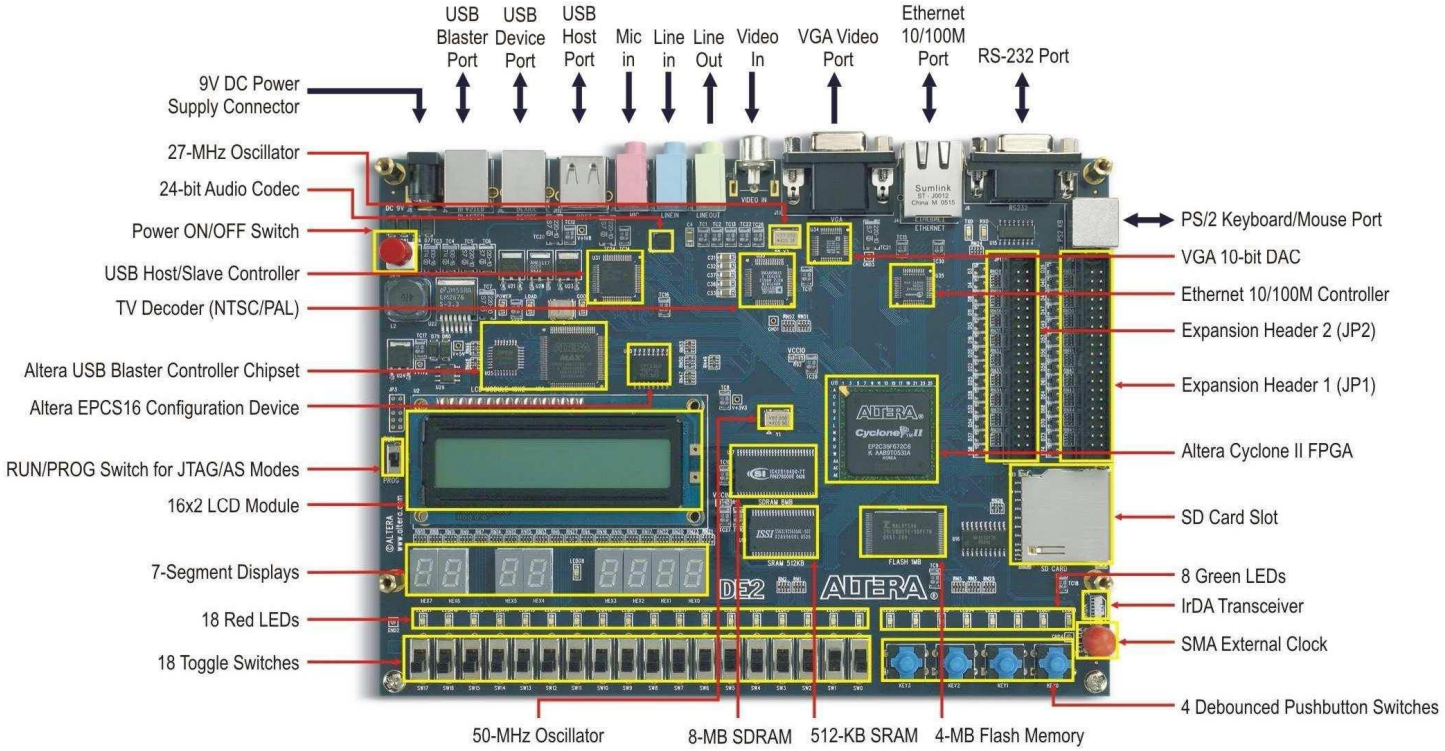

Fig. 14. Disposição dos dispositivos de hardware na DE2 Board.

Além de sua capacidade em hardware, a DE2 Board tem um suporte de software completo. Para editar, compilar, simular e gravar programas em HDL (Hardware Description Language) no FPGA, usaremos o software fornecido junto com o material de suporte, o Quartus II, que vai ser melhor conhecido nesse texto.

Na placa DE2 é possível desenvolver diversas implementações, desde projetos digitais simples: criação de registradores, flip-flops, contadores, controladores de entrada e saída de dados, controladores de acesso a memória, usando os recursos mais comuns, como os leds, chaves e a programação do FPGA por uma linguagem HDL devidamente escolhida; até projetos de alta complexidade que usam os recursos mais avançados oferecidos pela placa, tais como: decodificador de sinal de TV e áudio, implementação de interfaces de comunicação, tais como USB, sintetização de sons, entre outros.

3.1 A ferramenta de síntese lógica Quartus II

Nesta seção será detalhado o procedimento de criação de um projeto em HDL, a ser implementado no FPGA da placa DE2, usando a ferramenta Quartus II. No projeto exemplo será usada a linguagem de descrição de hardware Verilog HDL, também abreviada como Verilog.

Na implementação de um projeto usando dispositivo FPGA é necessária uma ferramenta de síntese lógica, do tipo CAD (Computer Aided Design), fornecida pelo fabricante do dispositivo FPGA escolhido para o projeto (no nosso caso a placa DE2 e seu respectivo chip FPGA). Essa ferramenta além de programar o dispositivo FPGA, permite a depuração de erros em várias linguagens de descrição de hardware, entre elas, Verilog HDL. Ela permite também simulações do projeto em implementação. Uma imagem da tela inicial do programa pode ser vista na Fig. 15.

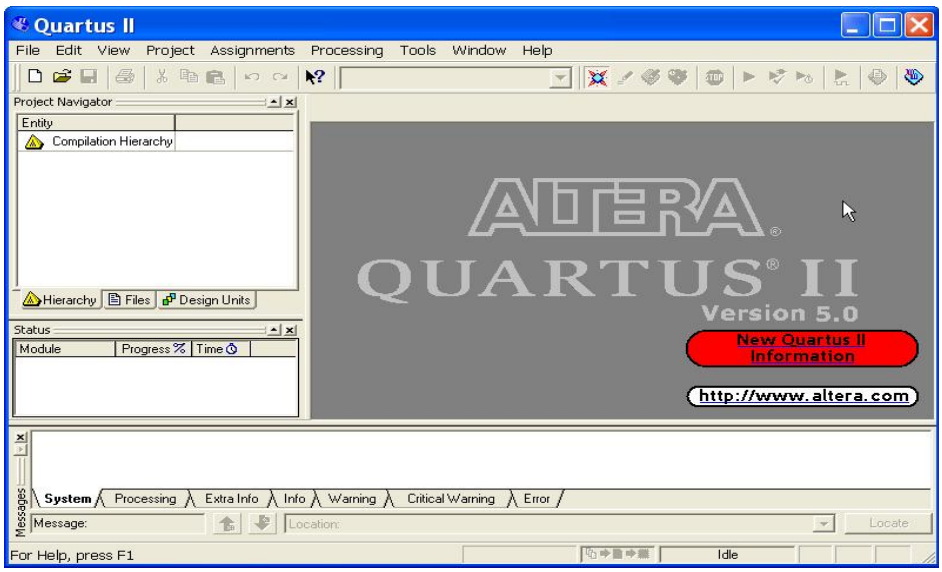

Fig. 15. Tela do Quartus II.

Para trabalhar com o Quartus II é necessário criar um projeto em que são especificados detalhes como: o dispositivo que está sendo utilizado, quais pinos do FPGA estão sendo associados aos leds e chaves, entre outros. O projeto deve incluir também o arquivo que contém a descrição lógica a ser implementada no dispositivo, no caso o arquivo *.v,* de Verilog. No arquivo

.*v* é criado um módulo com entradas e saídas que são associadas com os pinos de entrada e saída do FPGA e a lógica referente a sua implementação.

A criação de um novo projeto pode ocorrer de forma acompanhada, para isso, deve-se iniciar o processo selecionando a opção *File > New Project Wizard* (Fig. 16). Os passos seguintes são de certa forma auto-explicativos (Fig. 16). Foi necessário adicionar dois arquivos ao projeto. Um que permitiu a aquisição de dados pela porta serial do PC, e outro para gerar uma freqüência de 25 kHz, já que a placa da Altera trabalha com uma freqüência de 50 MHz e a serial não tem essa largura banda, pois trabalha com 115 Kbits/seg, assim, foi necessário dividir a freqüência de clock disponível por um fator de 2000. Pode ser visto na Fig. 17 essa guia. Foi selecionado também, o dispositivo referente a placa DE2, Cyclone II – EP2C35F67C6, sendo também observado na Fig. 17. Ao fim da criação de um novo projeto, é mostrado um resumo das opções escolhidas (Fig. 18).

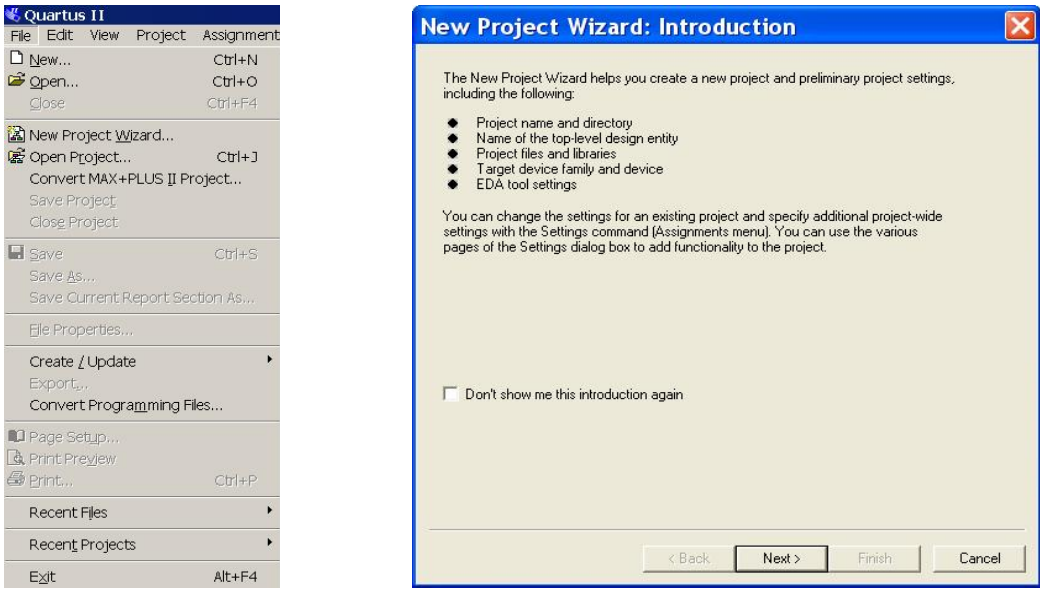

Fig. 16. Guia *file*>N*ew Project Wizard; S*eqüência de passos na edição Wizard

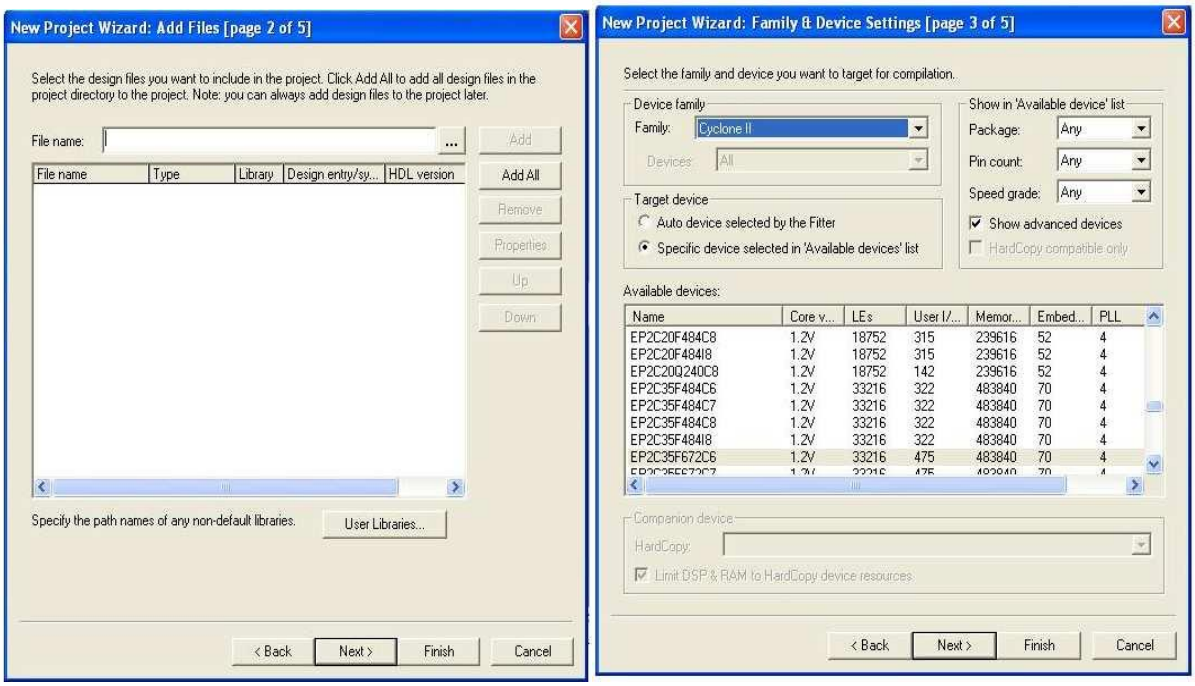

Fig. 17. Guia A*dd Files;* Selecionando o dispositivo.

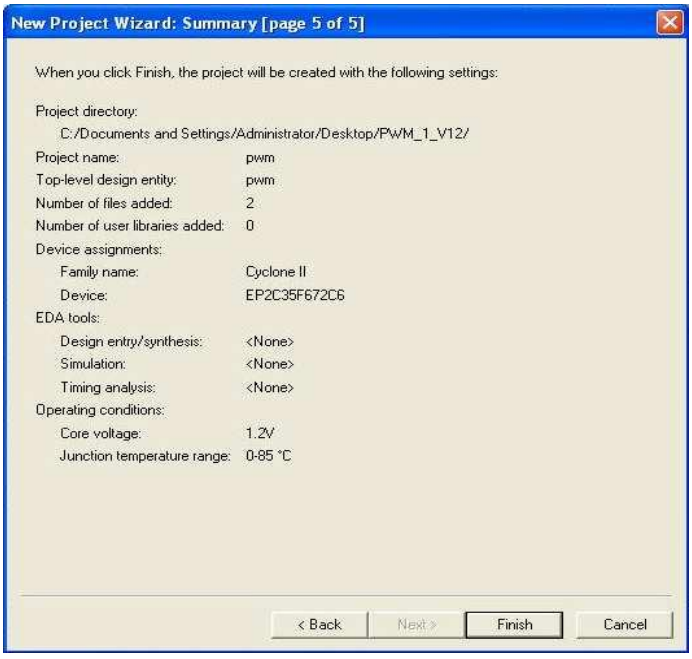

Fig. 18. Resumo das opções escolhidas

Concluída a primeira etapa, na etapa seguinte do processo de criação de um projeto HDL deve-se realizar a descrição do hardware a ser implementado através de uma linguagem de descrição de hardware, sendo no presente trabalho selecionada a linguagem Verilog.

4. Desenvolvimento de uma lógica em C

O pulso PWM nomalmente utilizado pode ser visto na Fig. 19 apresentando um pulso alto e um intervalo de tempo em nível baixo. Nesse trabalho se desenvolveu uma estratégia de software e hardware que permite subdividir, como mostrado na Fig. 20, esse pulso alto em vários pulsos menores. Com essa estratégia é possível minimar ou eliminar a tensão de modo comum e a oscilação do conjugado.

Para essa estratégia, partimos de um sinal PWM com divisão do pulso alto em vários pulsos menores. Utilizamos, para teste, cinco pulsos, onde a soma deles, tem como valor o pulso alto. Para isso, utilizamos uma matriz, para armazenamento dos valores das porcentagens (10%, 25%, 30%, 25% e 10%) que o pulso original será dividido. As porcentagens usadas não é padrão, foi escolhida para realização do projeto, podendo ser usada qualquer proporção para os valores dos recortes que serão gerados. Para o cálculo do pulso baixo, fazemos a diferença, do pulso alto, com o periodo total. Dividimos esse valor por seis, pois, vemos, na Fig. 17, que, serão necessários, seis espaços com tempo OFF.

Fig. 19. Esquema de um pulso PWM, em um período de 1 pu

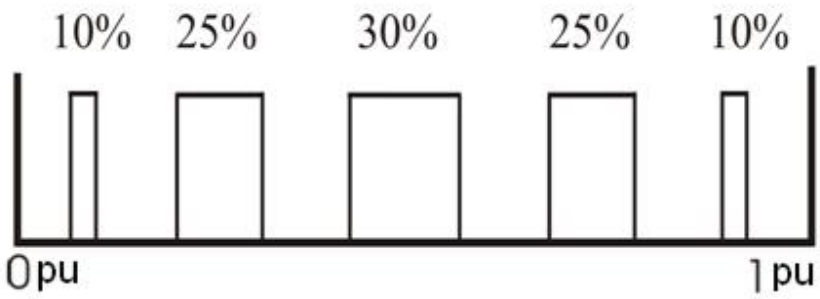

Fig. 20. Esquema de recortes proposto.

Na Fig. 16, supondo um tempo em alta de 0,4 pu, teríamos respectivamente, usando as porcentagens da Fig. 17:

Recorte 1 = 0,04 pu

Recorte 2 = 0,10 pu

Recorte 3 = 0,12 pu

Recorte 4 = 0,10 pu

Recorte 5 = 0,04 pu

Em seguida, foi feito a conversão, para bits. Como nosso período tem 1pu, e, como foi explicado anteriormente, vão ser geradas, 16 palavras, com 16 bits cada, ou seja, um total de 256 bits. Então, através da equação (1), será possível calcular o número de bits 1 e 0, para geração, respectivamente dos pulsos alto e baixo, a partir dos tempos dos recortes 1, 2, 3, 4 e 5 e dos tempos dos pulsos baixos:

$$x = \frac{(tempo) \times (256)}{1} \tag{12}$$

onde a variável *tempo*, é o valor que vai ser convertido.

Para o exemplo acima, com pulsos de 10%, 25%, 30%, 25% e 10% do tempo do pulso alto do PWM padrão, os tamanhos das sequências de bits serão:

Recorte 1 = 10,24

Recorte 2 = 25,60

Recorte 3 = 30,72

Recorte 4 = 25,60

Recorte 5 = 10,24

Para o cálculo do pulso baixo, como dito anteriormente, temos:

$$y = 1 - 0{,}40 = 0{,}6 \text{ pu} \quad (13)$$

$$div = 0{,}6/6 = 0{,}1 \text{ pu} \quad (14)$$

Então, para as sequências de zero:

Zero 1 = 25,60

Zero 2 = 25,60

Zero 3 = 25,60

Zero 4 = 25,60

Zero 5 = 25,60

Zero 6 = 25,60

Com os tempos calculados, e observando a Fig. 20, realizamos os cálculos abaixo:

(**1**) **25,60**

(**2**) 25,60+10,24 = **35,84**

(**3**) 25,60+10,24+25,6 = **61,44**

(**4**) 25,60+10,24+25,6+25,6 = **87,04**

(**5**) 25,60+10,24+25,6+25,6+25,6 = **112,64**

(**6**) 25,60+10,24+25,6+25,6+25,6+30,72 = **143,36**

(**7**) 25,60+10,24+25,6+25,6+25,6+30,72+25,6 = **168,96**

(**8**) 25,60+10,24+25,6+25,6+25,6+30,72+25,6+25,6 = **194,56**

(**9**) 25,60+10,24+25,6+25,6+25,6+30,72+25,6+25,6+25,6 = **220,16**

(**10**) 25,60+10,24+25,6+25,6+25,6+30,72+25,6+25,6+25,6+10,24 = **230,4**

(**11**) 25,60+10,24+25,6+25,6+25,6+30,72+25,6+25,6+25,6+10,24+25,6 = **256**

Fig. 21. Cálculos dos pulsos.

Aqueles valores calculados que apresentem parte fracionária serão truncados, já que os mesmos representam número de bits (é impossível haver 10.4 bits, por exemplo). A parte

fracionária dos recortes é somada ao pulso 3 (Ex. 10.4 bits, sendo o 0.4 somado ao pulso 3 dos recortes).

Observa-se na Fig. 21 que são determinados onze valores. Seguindo a sequência, temos o primeiro valor, como onde vai ter início o recorte, e o segundo valor, a largura do mesmo. Do **Exemplo 1** visto na Fig. 22**,** tomando os dois primeiros valores, o (1) diz onde vai ter início e o (2), estabelece, até onde vai o pulso (largura).

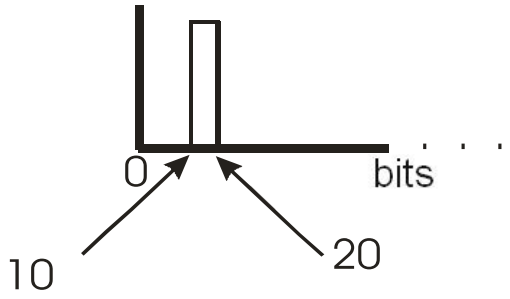

Fig. 22. Mostra o Exemplo 1.

A seguir na Fig. 23, temos o fluxograma, do código proposto para a determinação da matriz de valores 1 e 0, de tamanho 16X16.

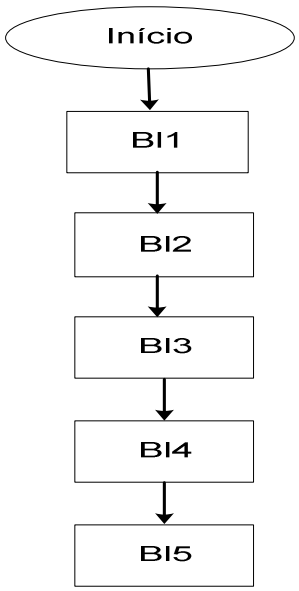

Fig. 23. Fluxograma do programa

No Bl - 1 são realizados os cálculos das porcentagens dos pulsos recortados a partir do pulso alto do sinal PWM padrão. Também é realizada a conversão para o valor em bits (12). No Bl - 2, são realizados os cálculos da Fig. 18. Esses valores vão sendo armazenados de maneira alternada em uma matriz, ocorrendo a gravação da seguinte maneira: o valor do pulso baixo (*div*, ver equação 14) é gravado na primeira posição, em seguida o valor do primeiro recorte, que equivale a 10% do valor do pulso alto original é gravado. Logo após, é gravado mais uma vez o valor de *div* e depois o valor do segundo recorte, que equivale a 25% do pulso alto original. E assim sucessivamente, tendo como base a Fig. 17. No Bl – 3 é realizada a geração da nossa matriz de dados. Nessa etapa é gerada uma grande linha de 256 elementos (bits), correspondentes aos valores calculados no Bl – 2 . **Exemplo 2**: o primeiro valor obtido, foi dez, então, ele age como um contador, inserindo zeros na matriz, até chegar ao valor final, no caso, 10:

0000000001...

No Bl - 4, é realizada a inversão. Ela é necessária para a visualização dos recortes. Depois de gerada, a linha com 256 bits, esse bloco vai sendo deslocado de modo que ao detectar um 1 ocorre a inversão do valor do bit.

Exemplo 3:

Antes: ... 0000000010000010000000001000000001

Depois: ...0000000011111110000000001111111111

O Bl - 5 vai “dividir” essa linha de 256 bits, em linhas com 16 bits. No final, obtem-se a nossa matriz, de 16 linhas com 16 bits cada.

Então, para o processo, foi gerada uma matriz com 256 posições, sendo a mesma disposta de 16 linhas por 16 colunas (16X16). A segunda etapa é a divisão (recortes) do pulso alto em pulsos menores, ocorrendo o preenchimento de cada posição desta matriz com 0 e 1. O “zero” corresponderá a chave no estado aberto e o “um” ao estado fechado. A medida que o valor do pulso alto for mudando, ocasionará mudanças nas disposições dos elementos dessa matriz.

5. Implementação em VERILOG

Foi criada uma FIFO, para gravação dos dados, com 256 posições e cada locação de memória é de 8 bits. O dados são enviados pelo PC usando-se a comunicação serial UAR (Universal Asynchronous Receiver) com a seguinte configuração:

Baud Rate: 115200bps;
Data bits: 8;
Parity: Nenhuma;
Stop bits: 1.

Pode ser visto na Fig. 24 o diagrama de blocos do processo. Foi necessário acrescentar dois arquivos ao programa principal. Um que permitiu a aquisição de dados pela porta serial do PC (bloco SERIAL), e outro para gerar uma freqüência de 25 kHz, já que a placa da Altera trabalha com uma freqüência de 50 MHz e a serial não tem essa largura banda, pois trabalha com 115 Kbits/seg, assim, foi necessário dividir a freqüência de clock disponível por um fator de 2000 (bloco ÷2000).

O bloco serial, implementado no FPGA, ativa um sinal WR (nível alto), indicando ao FIFO que há um byte a ser escrito no mesmo. Quando outro byte é recebido pelo bloco serial, o mesmo é salvo numa posição subsequente a que foi salvo o byte anterior, sendo que esse processo é repetido até que todo FIFO seja preenchido. Ao final do preenchimento, o flag EN sinaliza que o FIFO está preenchido e, com isso, dar-se início a geração do sinal PWM. O sinal RD é habilitado para indicar que dados estão sendo lidos do FIFO. Os dados são retirados do FIFO em palavras de 16 bits e deslocados bit a bit para geração do PWM.

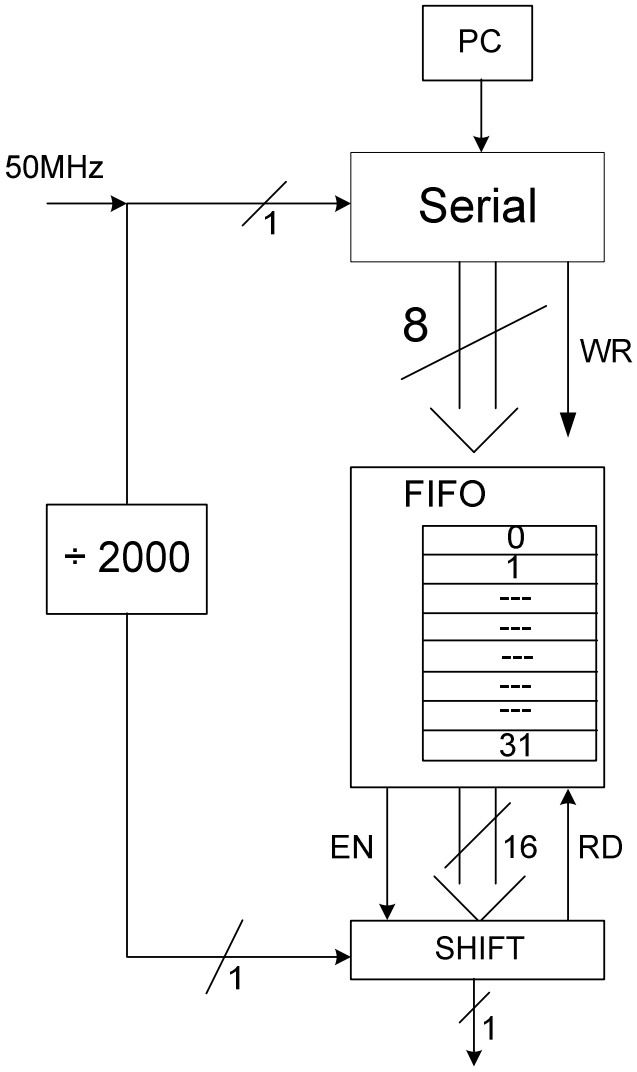

Fig. 24. Diagrama de blocos.

6. Resultados Experimentais

Para a variação do *tal* e conseqüente geração das matrizes foi utilizado o programa Turbo C. Foi escrito um código para gerar as variações do *tal*. Com esta variação foram gerados os dados que serão escritos nas matrizes. Estes dados, como dito anteriormente, vão se referir aos tempos dos recortes. Nesse código foi utilizado uma senóide, com o período de amostragem igual a 10.5 e^{-3} (Ts = 10.5 e^{-3}) e freqüência de 60 Hz , ocorrendo a geração das matrizes. Cada bit é enviado com uma taxa de 0,002 segundos. Pode ser observado nas figuras subseqüentes que o período dos 32 bytes que compõe o PWM recortado tem valor é igual a 20 ms.

As matrizes obtidas ao fim da execução dos algoritmos, quando transmitidas serialmente, reproduzirão sinais PWM com os múltiplos pulsos programados. Nas Fig. 25 a Fig. 30 são apresentadas as curvas PWM, geradas utilizando a placa DE2 da Altera, modificando o valor de *tal*, sendo esta variação compreendida entre 0 e 1. Para o valor de *tal* igual a 1, os recortes não ficaram visíveis, logo, utilizou-se um valor próximo de 1 (*tal* = 0,96).

As curvas apresentadas nessas figuras representam o sinal PWM com múltiplos pulsos e uma onda quadrada de referência, que fica em nível baixo enquanto estão sendo enviados os 32 bytes da matriz, mostrando assim o sincronismo e a estabilidade do sinal PWM gerado.

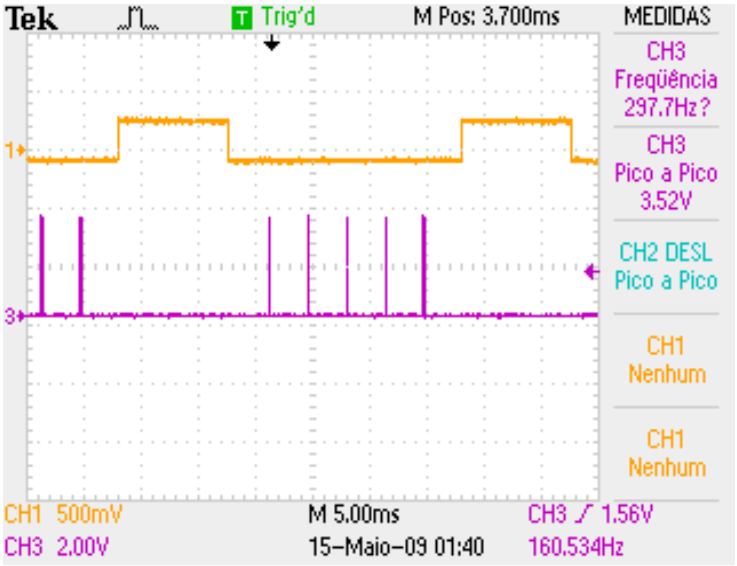

Fig. 25. Tal = 0 pu.

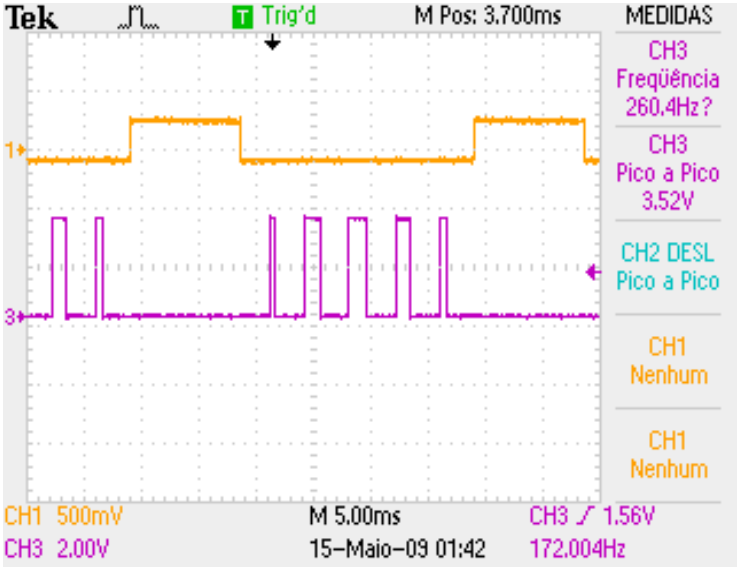

Fig. 26. Tal = 0,25 pu

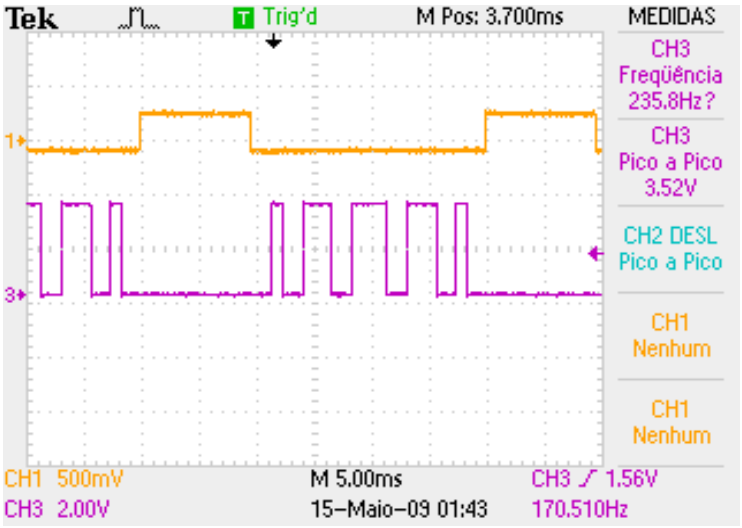

Fig. 27. Tal = 0,50 pu

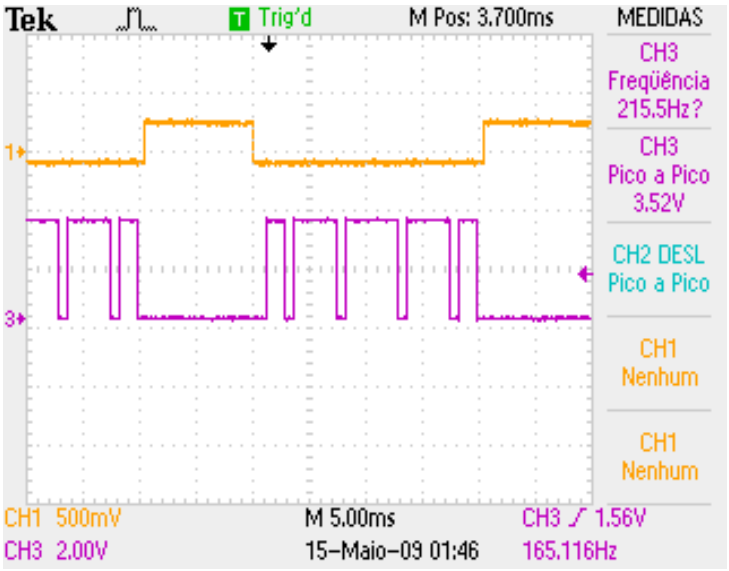

Fig. 28. Tal = 0,75 pu

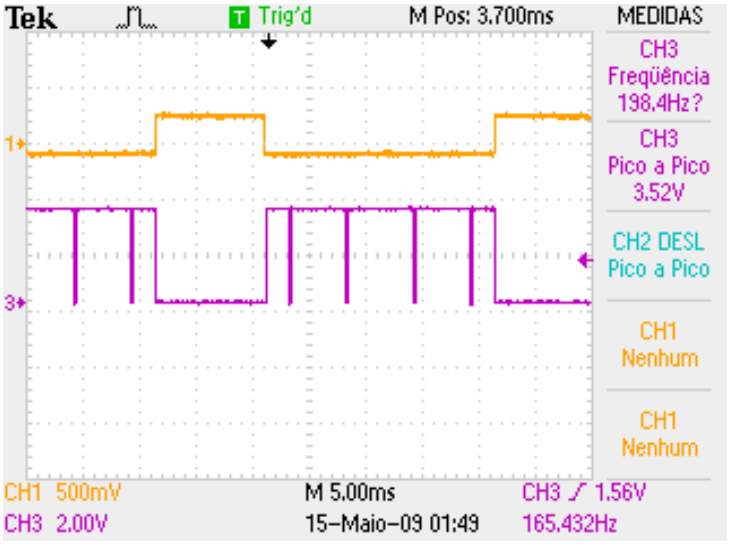

Fig. 29. Tal = 0,96 pu

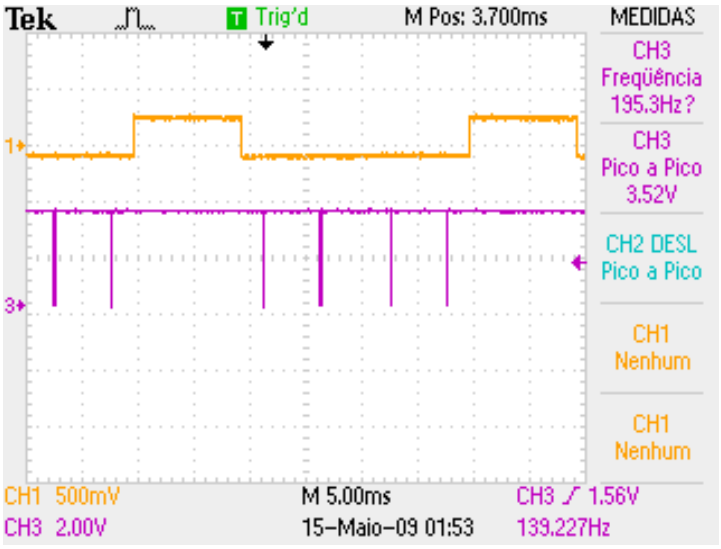

Fig. 30. Tal = 1 pu

Para variações contínuas do valor do pulso alto, temos variações contínuas dos recortes. Vemos nas Fig. 31 e Fig. 32, que em dois períodos consecutivos o valor do pulso 3, em cada período, tem valores diferenciados. Essa variação é devido a variação da onda senoidal, usada para gerar o PWM. Na Fig. 27, temos o valor do recorte 3 igual a 3,6ms. Já na Fig. 28, temos para o recorte 3, o valor de 3,4ms.

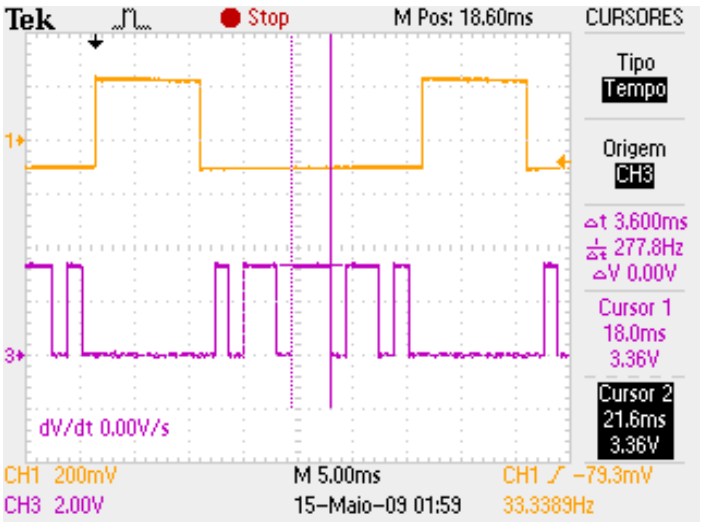

Fig. 31. Precisão dos pulsos gerados. Pulso 3 de valor 3,6ms.

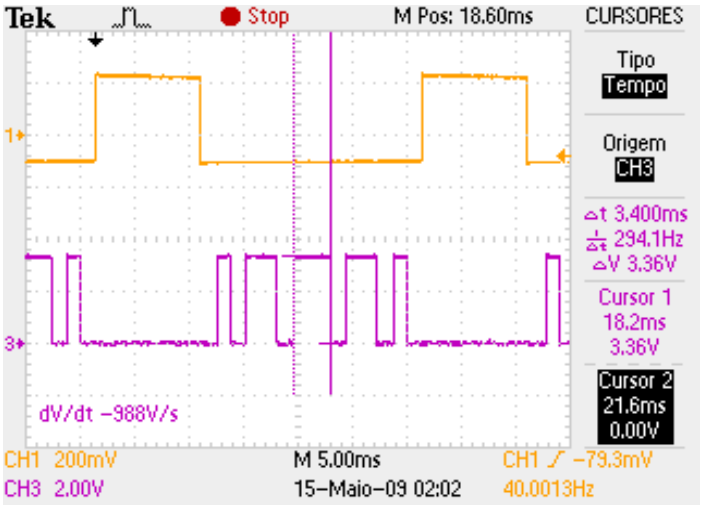

Fig. 32. Precisão dos pulsos gerados. Pulso 3 de valor 3,4ms.

Nesses resultados observa-se que é possível ocorrer a distribuição dos pulsos ao longo de um período PWM. Para o propósito de ser aplicado no acionamento de inversores, este sinais não seriam aplicados. Para este tipo de acionamento, o nível alto entre cada envio de dados seria eliminado (ver Fig. 33, destaque em vermelho), tornando o sinal contínuo. A forma de onda amarela indicada na Fig. 33 permanece em nível baixo quando se está enviando o FIFO de 32 bytes, e volta em seguida ao nível alto.

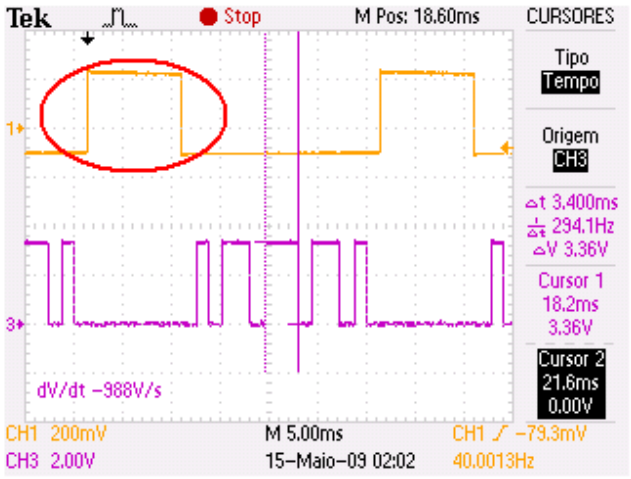

Fig. 33. Precisão dos pulsos gerados. Pulso 3 de valor 3,4ms.

7. Conclusão

Este trabalho apresenta uma forma alternativa de geração de sinais PWM com múlitplos pulsos utilizando um recurso disponível em dispositivos dedicados, desenvolvidos para uso na implementação de sistemas de acionamento, como por exemplo, a DE2 da Altera, usado na implementação do gerador PWM.

Foram apresentados casos de geração de sinais PWM com múltiplos pulsos em aplicações de acionamento de sistemas hexafásicos com eliminação da tensão de modo comum. A escolha do exemplo se deu pela atualidade do assunto e pelo fato das soluções de implementação, propostas na literarura, fazerem uso de dispositivos FPGA. A solução apresentada no trabalho pode ser uma alternativa ao uso de dispositivos periféricos adicionais, como os casos com FPGA.

A implementação do gerador, em termos de programação, faz uso de um algoritmo simples, que pouco exige da CPU.

8. Referências Bibliográficas

Bakhshai, A. R., Joos, G. and Jim, H. (1998). Space vector pwm control of a split-phase induction machine using the vector classification technique, *Proceedings of Applied Power Electronics Conference*, pp. 802 – 808.

Correa, M., Jacobina, C., da Silva, C., Lima, A. and da Silva, E. (2003a). Six-phase ac drive system with reduced common-mode voltage, *Proceedings of IEEE InternationalElectric Machines and Drives Conference,IEMDC'03*, pp. 1852 – 1858.

Correa, M., Jacobina, C., da Silva, C., Lima, A. and da Silva, E. (2003b). Vector and scalar modulation for six-phase voltage source inverters, *Proceedings of IEEE Power Electronics Specialist Conference, PESC'03*, pp. 562 – 567.

Jacobina, C. B., Lima, A. M. N. and Silva, E. R. C. (1997). Pwm space vector based in digital scalar modulation, *Proceedings of Power Electronics Specialist Conference*, pp. 606 – 611.

Jacobina, C., de Rossiter Correa, M., da Silva, E. and Lima, A. (1999). Induction motor drive system for low-power applications, *IEEE Trans. on Industry Applications* **35**: 56 – 61.

Jian, L., Zhe, Z., Xianggen, Y. and Minghao, W. (2007). Fpga implementation of a multilevel spwm for three-level npc inverter, *Proceedings of the 41st International Universities Power Engineering Conference*, pp. 175 – 179.

Julian, A. L., Lipo, T. A. and Oriti, G. (1996). Elimination of common-mode voltages in three phase sinusoidal power converters, *Proceedings of Power Electronics Specialist Conference*, pp. 1968 – 1972.

Manjrekar, M. D. and Lipo, T. A. (1999). An auxiliary zero state synthesizer to reduce common mode voltage in three-phase inverters, *Proceedings of Industry Applications Society Conference*, pp. 55 – 59.

User Manual, DE2 Altera. Disponível em: http://www.altera.com/education/univ/materials/boards/unv-de2-board.html. Acessado em Janeiro de 2009.

Mondal, G., Gopakumar, K., Tekwani, P. N. and Levi, E. (2007). A reduced-switch-count five-level inverter with common-mode voltage elimination for an open-end winding induction motor drive, *IEEE Trans. on Industry Applications* **54**: 2344 – 2351.

9. ANEXO

Código em VERILOG

PWM.V

```
module PWM(CLOCK_50, GPIO_1, UART_RXD, LEDG, LEDR, KEY);

input CLOCK_50;
input UART_RXD;
input [0:0]KEY;

output [1:0] GPIO_1;
output [7:0] LEDG;
output [5:0] LEDR;

////////////////////////////////////////////////
wire RxD_data_ready;
wire [7:0] RxD_data;
wire Sinal;

reg [5:0]Cont;
reg [4:0]Contador;

reg Saida;
reg [7:0]Contador_2;

reg [0:0]Flag;
reg [255:0]Matriz;

reg Saida_2;

async_receiver deserializer(.clk(CLOCK_50), .RxD(UART_RXD),
.RxD_data_ready(RxD_data_ready), .RxD_data(RxD_data));
Clock_Quatro_Hz   Clock(Sinal, CLOCK_50);


always @(negedge CLOCK_50 or negedge KEY[0])
begin

   if(!KEY[0])
   begin
      Cont <= 6'b0;
      Contador <= 6'b0;
   end
```

```
    else
    begin

//Início da  FIFO
        if(RxD_data_ready)
        begin
          case (Contador)

            5'd0:   Matriz[7:0]     <= RxD_data;
            5'd1:   Matriz[15:8]    <= RxD_data;
            5'd2:   Matriz[23:16]   <= RxD_data;
            5'd3:   Matriz[31:24]   <= RxD_data;
            5'd4:   Matriz[39:32]   <= RxD_data;
            5'd5:   Matriz[47:40]   <= RxD_data;
            5'd6:   Matriz[55:48]   <= RxD_data;
            5'd7:   Matriz[63:56]   <= RxD_data;
            5'd8:   Matriz[71:64]   <= RxD_data;
            5'd9:   Matriz[79:72]   <= RxD_data;
            5'd10:  Matriz[87:80]   <= RxD_data;
            5'd11:  Matriz[95:88]   <= RxD_data;
            5'd12:  Matriz[103:96]  <= RxD_data;
            5'd13:  Matriz[111:104] <= RxD_data;
            5'd14:  Matriz[119:112] <= RxD_data;
            5'd15:  Matriz[127:120] <= RxD_data;
            5'd16:  Matriz[135:128] <= RxD_data;
            5'd17:  Matriz[143:136] <= RxD_data;
            5'd18:  Matriz[151:144] <= RxD_data;
            5'd19:  Matriz[159:152] <= RxD_data;
            5'd20:  Matriz[167:160] <= RxD_data;
            5'd21:  Matriz[175:168] <= RxD_data;
            5'd22:  Matriz[183:176] <= RxD_data;
            5'd23:  Matriz[191:184] <= RxD_data;
            5'd24:  Matriz[199:192] <= RxD_data;
            5'd25:  Matriz[207:200] <= RxD_data;
            5'd26:  Matriz[215:208] <= RxD_data;
            5'd27:  Matriz[223:216] <= RxD_data;
            5'd28:  Matriz[231:224] <= RxD_data;
            5'd29:  Matriz[239:232] <= RxD_data;
            5'd30:  Matriz[247:240] <= RxD_data;
            5'd31:
            begin
                                        Matriz[255:248] <= RxD_data;
                                        Flag <= 1'b1;
                                end
          endcase
```

```
            Contador <= Contador + 4'h1;
            Cont <= Cont + 1'b1;
        end

        if(Contador_2 == 8'd0)
                                Flag <= 1'b0;

    end
end
//----------------------------------------------------------------------------
    /*always @ (Contador)
    begin
        if(Contador[5] == 1'b1)
            Flag <= 1'h1;
        else
            Flag <= 1'h0;
    end*/
//----------------------------------------------------------------------------

always @ (negedge Sinal or negedge KEY[0])
begin
    if(!KEY[0])
    begin
        Contador_2 <= 8'd255;
        Saida <= 1'h0;
    end

    else
    begin



        if(Flag)
        begin
            Contador_2 <= Contador_2 - 1'b1;
            Saida <= Matriz[Contador_2];
        end


        else
            Contador_2 <= 8'd255;
            //Flag <= 1'h0;



    end
```

```
end

always @ (Contador_2)
begin

	if((Contador_2==8'd255) || (Contador_2==8'd0))
		Saida_2 <= 1'h1;
	else
		Saida_2 <= 1'h0;

end


assign GPIO_1[1] = Saida;
assign LEDG[7:0] = RxD_data;
assign LEDR[5:0] = Cont;
assign GPIO_1[0] = Saida_2;

endmodule
```

Async_receiver.V (BLOCO SERIAL)

```
module async_receiver(clk, RxD, RxD_data_ready, RxD_data, RxD_endofpacket, RxD_idle);
input clk, RxD;
output RxD_data_ready;  // onc clock pulse when RxD_data is valid
output [7:0] RxD_data;

parameter ClkFrequency = 50000000; // 50MHz
//parameter ClkFrequency = 27000000; // 27MHz
parameter Baud = 115200;

// We also detect if a gap occurs in the received stream of characters
// That can be useful if multiple characters are sent in burst
//  so that multiple characters can be treated as a "packet"
output RxD_endofpacket;  // one clock pulse, when no more data is received (RxD_idle is going
high)
output RxD_idle;  // no data is being received

// Baud generator (we use 8 times oversampling)
parameter Baud8 = Baud*8;
parameter Baud8GeneratorAccWidth = 16;
parameter Baud8GeneratorInc = ((Baud8<<(Baud8GeneratorAccWidth-
7))+(ClkFrequency>>8))/(ClkFrequency>>7);
reg [Baud8GeneratorAccWidth:0] Baud8GeneratorAcc;
```

```
always @(posedge clk) Baud8GeneratorAcc <= Baud8GeneratorAcc[Baud8GeneratorAccWidth-1:0] + Baud8GeneratorInc;
wire Baud8Tick = Baud8GeneratorAcc[Baud8GeneratorAccWidth];

//////////////////////////
reg [1:0] RxD_sync_inv;
always @(posedge clk) if(Baud8Tick) RxD_sync_inv <= {RxD_sync_inv[0], ~RxD};
// we invert RxD, so that the idle becomes "0", to prevent a phantom character to be received at startup

reg [1:0] RxD_cnt_inv;
reg RxD_bit_inv;

always @(posedge clk)
if(Baud8Tick)
begin
  if( RxD_sync_inv[1] && RxD_cnt_inv!=2'b11) RxD_cnt_inv <= RxD_cnt_inv + 1;
  else
  if(~RxD_sync_inv[1] && RxD_cnt_inv!=2'b00) RxD_cnt_inv <= RxD_cnt_inv - 1;

  if(RxD_cnt_inv==2'b00) RxD_bit_inv <= 0;
  else
  if(RxD_cnt_inv==2'b11) RxD_bit_inv <= 1;
end

reg [3:0] state;
reg [3:0] bit_spacing;

// "next_bit" controls when the data sampling occurs
// depending on how noisy the RxD is, different values might work better
// with a clean connection, values from 8 to 11 work
wire next_bit = (bit_spacing==10);

always @(posedge clk)
if(state==0)
  bit_spacing <= 0;
else
if(Baud8Tick)
  bit_spacing <= {bit_spacing[2:0] + 1} | {bit_spacing[3], 3'b000};

always @(posedge clk)
if(Baud8Tick)
case(state)
  4'b0000: if(RxD_bit_inv) state <= 4'b1000;  // start bit found?
  4'b1000: if(next_bit) state <= 4'b1001;  // bit 0
  4'b1001: if(next_bit) state <= 4'b1010;  // bit 1
```

```
  4'b1010: if(next_bit) state <= 4'b1011;  // bit 2
  4'b1011: if(next_bit) state <= 4'b1100;  // bit 3
  4'b1100: if(next_bit) state <= 4'b1101;  // bit 4
  4'b1101: if(next_bit) state <= 4'b1110;  // bit 5
  4'b1110: if(next_bit) state <= 4'b1111;  // bit 6
  4'b1111: if(next_bit) state <= 4'b0001;  // bit 7
  4'b0001: if(next_bit) state <= 4'b0000;  // stop bit
  default: state <= 4'b0000;
endcase

reg [7:0] RxD_data;
always @(posedge clk)
if(Baud8Tick && next_bit && state[3]) RxD_data <= {~RxD_bit_inv, RxD_data[7:1]};

reg RxD_data_ready, RxD_data_error;
always @(posedge clk)
begin
  RxD_data_ready <= (Baud8Tick && next_bit && state==4'b0001 && ~RxD_bit_inv);  // ready
only if the stop bit is received
  RxD_data_error <= (Baud8Tick && next_bit && state==4'b0001 &&  RxD_bit_inv);  // error if
the stop bit is not received
end

reg [4:0] gap_count;
always @(posedge clk) if (state!=0) gap_count<=0; else if(Baud8Tick & ~gap_count[4])
gap_count <= gap_count + 1;
assign RxD_idle = gap_count[4];
reg RxD_endofpacket; always @(posedge clk) RxD_endofpacket <= Baud8Tick &
(gap_count==15);

endmodule
```

clock_um_hz.V (BLOCO REDUÇÂO DO CLOCK)

```
module Clock_Um_Hz(Sinal_Saida, Relogio);

	output Sinal_Saida;

	input Relogio;

	reg [24:0]Registradores; //Para contar até 6250000
	reg Sinal_Saida; //Registrador que irá "gerar" a frequência desejada,
			//que nesse caso é de 4Hz

	always @ (posedge Relogio)
	begin
```

```
            if (Registradores == 25'd6249999)
            begin
                Registradores <= 1'b0;
                Sinal_Saida = ~Sinal_Saida;
            end

            else
                Registradores <= Registradores + 1'b1;

        end

endmodule
```

Código em C

```
/* ********************************************************************
Trabalho de CoclusÆo de Curso
Aluno: Luciano de Macedo Barros
ModulacÆo utilizando pulsos recortados

/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ */
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include "graph.h"
#include "datastru.h"

#define ws _ws
//************************************************************
//************************************************************
//************************************************************

/*   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    */
/* +++++++++++++++ Defini‡Æo da nova interrup‡Æo RTC ++++++++++++++++++++ */
void interrupt myint8(...);        // Nova interrup‡Æo do rel¢gio
void interrupt (*oldint8)(...);    // Apontador para a int do rel¢gio
```

```
void interrupt myint9(...);
void interrupt (*oldint9)(...);
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* +++++ Atribui‡Æo do ponteiro do vetor de interrup‡Æo de rel¢gio +++++ */
/* ++++++++++++++++++++ … constante TIMER. ++++++++++++++++++++++++++++ */
#define TIMER 0x08
#defineKEYBD 0x09
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
// Defini‡Æo dos endere‡os de I/O do controlador de interrup‡Æo
// (IC8259_PC), contador (IC8254_PC) e PPI (IC8255_PC) do PC.
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
#define IC8259_PC 0x020              // Controlador de interrup‡Æo do PC
#define IC8254_PC 0x040              // Contadores
#defineIC8255_PC 0x060                    // Interface programavel de 24 pinos
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
// EOI representa o valor que deve ser escrito no controlador de interrupcao
// do PC (IC8259_PC), informando o fim do atendimento de uma interrupcao.
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
#define EOI 0x20

// Vari veis utilizadas pelo programa.
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
char hc = 0;    // chave de encerramento de execucao


//////////////////////////////////////////////////
//////////////////////////////////////////////////
unsigned dl=1; //115.2kbps
unsigned lcrimg = 0x003;

//////////////////////////////////////////////////
//////////////////////////////////////////////////


float Ts = 200e-3;//10.5e-3; //100e-6   // 1/ts e a frequencia com que a interrupcao de relogio
                  // do PC passara a ser solicitada.
float t=0.;

float tp1=0., tp2=0., tp3=0., tp4=0., tp5=0., tp6=0.;
float toff_a = 0.0;        // contagem para canal a
float toff_b = 0.0;        // contagem para canal b
```

```
float toff_c = 0.0;        // contagem para canal c
float toff_d = 0.0;        // contagem para canal d
float toff_e = 0.0;        // contagem para canal e
float toff_f = 0.0;        // contagem para canal f
unsigned int nc_a, nc_b, nc_c, nc_d, nc_e, nc_f;

int lsb1, msb1, lsb2, msb2, lsb3, msb3, lsb4, msb4, lsb5, msb5, lsb6, msb6;

////////////////////////////////////////////////
////////////////////////////////////////////////
int n, teste, p, k, L, a[256], i, b[256], v, naotal, H, T=256;

float tal, tal2, dv, c[5] = {0.1, 0.25, 0.3, 0.25, 0.1}, tal1, x, d[30], w, r, e[20];
////////////////////////////////////////////////
////////////////////////////////////////////////



**********************************************************************************/
void main(void)
 {

outportb(COM1+3,lcrimg|0x80);
outportb(COM1+0,dl&0xff);
outportb(COM1+1,dl>>8);
outportb(COM1+3,lcrimg);


        {
                puts("Erro: Falta memoria !");
                getch();
                exit(1);
                }

        for(ng = 0; ng < ad; ng++)
                {
                TABELA[ng] = (20./ad)*ng-10.;
                //((float)ng - ad/2.)*(2.*vr/(float)ad);
                }

        // Fim da construcao das tabelas para leitura dos conversores AD
 clrscr();
 do{
 ch=0; hc=0; nb=0; nr=0;
 executa();
 //if(hc==0x02){ resultado(); }
```

```
textattr(0x07);
clrscr();
cprintf("Outra realiza‡Æo ? (<ESC> - sai)");
jaentrou=0;
}while(getch()!=27);

        if (1) {
                ofstream out("out.txt",ios::out);
                out<<tp1<<endl;
                for (int i=0; i<16; ++i) {
                        for (int j=0; j<16; ++j) {
                                out<<((b[i]>>(15-j))&1);
                        }
                        out<<endl;
                }
        }

free(TABELA);
free(out);
//Desprograma a paralela
outportb(DATA,0);
outportb(CONTROL, 0);
//DesProgramaPorta();
}

void executa(void)
 {

        clrscr();     // Limpa a tela do PC

        fs=60.0;
        ws = 2.0*pi*fs;
        kp =0.01;//5.0

/ ++++++++++++++++++   Programa‡Æo dos contadores   ++++++++++++++++++++ //
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// O bloco a seguir desabilita todas as interrup‡äes mascar veis, no entanto,
// antes disso, a m scara original de interrup‡Æo , salva em "imr".
/* ++++++++++++++++++   Desabilita‡Æo de interrup‡äes   ++++++++++++++++++ */
imr = inportb(IC8259_PC+1);
md = 0XFF;
puts("Opera‡Æo em tempo real iniciada...");
puts("Tecle <space> para interromper.");
do
 {
        outportb(IC8259_PC+1,md);
```

```
 } while( (inportb(IC8259_PC+1) != md) );

/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ */
// O bloco abaixo seta o novos vetores de interrup‡Æo para rel¢gio. O vetor
// de interrup‡Æo , o endere‡o que aponta para a primeira linha da rotina
// que atende ao pedido de interrup‡Æo.
/* +++++++++++++++++   Seta novo vetor de interrup‡Æo   ++++++++++++++++++ */
setvect(TIMER,myint8);  // Novo vetor da interrup‡Æo de rel¢gio
setvect(KEYBD,myint9);
/* /* +++++++++++++++ Habilita‡Æo de interrup‡äes ++++++++++++++++ */
md = 0XFC;
do
 {
        outportb(IC8259_PC+1,md);
 } while( (inportb(IC8259_PC+1) != md) );

          do{

                  enable();

                 }while( (ch != 0x01) );



/* +++++++++++++++ Desabilita‡Æo de interrup‡äes ++++++++++++++++ */
md = 0XFF;
do
 {
        outportb(IC8259_PC+1,md);
 } while( (inportb(IC8259_PC+1) != md) );
/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ */

/* +++++++ Restaura o vetor original de interrup‡Æo +++++++++ */
setvect(TIMER,oldint8);  // Vetor original da interrupcao de relogio
setvect(KEYBD,oldint9);
/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ */
/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ */
```

```
/* ++++ Seta a frequencia da interrupcao de relogio em 0xFFFF ++++ */
lsb = 0XFF;
msb = 0XFF;
outportb(IC8254_PC,lsb);
outportb(IC8254_PC,msb);
/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ */

/* ++++++++ Restaura a m scara original de interrup‡Æo ++++++++++ */
do
 {
        outportb(IC8259_PC+1,imr);
 } while( (inportb(IC8259_PC+1) != imr) );
enable();

}

          E =100.0;//60.0*sqrt(2.);
          static double my_theta = 0;
          my_theta += 2*M_PI*Ts/10;
          vs10ref = (0.75*sin(my_theta))*E/2;
// Determinador de periodo do PWM
//Conversor do Gerador

        tp1=(vs10ref/E+0.5);  //
        //tp1=Ts-tp1;
        //if(tp1 > Ts) tp1=Ts;
        //if(tp1 < 0.) tp1=0.;

        H = tp1;
        e[10]=256;
        unsigned int dados [16];
        for (n = 0; n<16; n++) dados[n]=0;

//////////////////////////////////////////////////Bl1 (inicio)////////////////////////////////////////////////////////
        for(n=0; n<5; n++){
                        tal1 = tp1*c[n];
                        x = (tal1*256); //(256, tal1*(1/102.4));
                        d[n] = x;
                        //d[n] = tp1*c[n];
                        //H -= d[n];
                        }
//////////////////////////////////////////////////Bl1 (fim)////////////////////////////////////////////////////////

                //w = (((tp1)*(256)));
```

```
                //H = w - (2*(d[0]+d[1]) + d[2]);

          d[2] = d[2] + H;
          tal2 = 1. - tp1;
          dv = tal2/6.;
          p = ((dv)*(256));

        //naotal = T - tp1;
        //p = naotal/6;
                //teste = 0;


////////////////////////////////////////////////Bl2 (inicio)//////////////////////////////////////////////////////////

for(r=0, n=0; n<10; n++){
  if(n%2==0)
  r = r + p;

  else{
  r = r + d[n/2];
  }
  e[n] = r;

}
////////////////////////////////////////////////Bl2 (fim)////////////////////////////////////////////////////////////

////////////////////////////////////////////////Bl3 (inicio)//////////////////////////////////////////////////////////

  for(n=0,i=0;i<e[10];i++){
                while(i < e[n]){
                        b[i] = 0;
                        i++;
                }
                b[i] = 1;
                n++;
        }
////////////////////////////////////////////////Bl3 (fim)////////////////////////////////////////////////////////////

////////////////////////////////////////////////Bl4 (inicio)//////////////////////////////////////////////////////////

        teste = 0;
        for (n = 0; n<256; n++){
                        if(b[n]==1) teste = !teste;
                        dados[n/16] = (dados[n/16]<<1) | teste;
                }
////////////////////////////////////////////////Bl4 (fim)////////////////////////////////////////////////////////////
```

```
        for (n = 0; n<16; n++) {
                b[n]=dados[n];
        }



         unsigned int vetor [16]=
         {0xFFFF,0x0000,0xFFFF,0X0000,
          0xFFFF,0x0000,0xFFFF,0X0000,
          0xFFFF,0x0000,0xFFFF,0X0000,
          0xFFFF,0x0000,0xFFFF,0X0000};
                */

#define bitswap(x) (          \
        ( (x)&0x80 ? 0x01 : 0 ) | \
        ( (x)&0x40 ? 0x02 : 0 ) | \
        ( (x)&0x20 ? 0x04 : 0 ) | \
        ( (x)&0x10 ? 0x08 : 0 ) | \
        ( (x)&0x08 ? 0x10 : 0 ) | \
        ( (x)&0x04 ? 0x20 : 0 ) | \
        ( (x)&0x02 ? 0x40 : 0 ) | \
        ( (x)&0x01 ? 0x80 : 0 )   \
)
////////////////////////////////////////////////Bl5 (inicio)//////////////////////////////////////////////////////////

        for(i=0; i<16; i++){

                while (!(inportb(COM1+5)&0x20));
                outportb(COM1,bitswap(dados[i]>>8));
                while (!(inportb(COM1+5)&0x20));
                outportb(COM1,bitswap(dados[i]));

}
////////////////////////////////////////////////Bl5 (fim)//////////////////////////////////////////////////////////


        outportb(AD1674_1+1,0x0000);




        outportb(IC8255_2+2,0x00);

          outportb(IC8259_PC,EOI);  // sinaliza‡Æo de fim de interrup‡Æo
```

```
}
/* ========================================================================= */
/* ========================= Fim da rotina ========================= */
/* ========================================================================= */

//
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++




void interrupt myint9(...)
        {

        static char al = 0;

        enable();
        ch = inportb(IC8255_PC);
        sc = ch;
        al = inportb(IC8255_PC+1);
        outportb(IC8255_PC+1,(al | 0x80));
        outportb(IC8255_PC+1,al);
        if( (ch == 0x39) ) hc = 0x01;
        outportb(IC8259_PC,EOI);

        }
```