Faculdade de Engenharia Elétrica e de Computação

ANÁLISE DE DESEMPENHO DE PROCESSADORES DIGITAIS APLICADOS AO CONTROLE DE BAIXO CUSTO DE MÁQUINAS ELÉTRICAS TRIFÁSICAS

Autor: ***Mário Luis Botêga Jr.***

Banca Examinadora:

Prof. Dr. Ernesto Ruppert Filho – Orientador
Prof. Dr. Manoel Luis de Aguiar
Prof. Dr. Maurício Ferreira Magalhães
Prof. Dr. Edson Adriano Vendrúsculo

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Maio / 2005

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Botêga Jr., Mário Luis
B657a Análise de desempenho de processadores digitais aplicados ao controle de baixo custo de máquinas elétricas trifásicas / Mário Luis Botêga Jr. --Campinas, SP: [s.n.], 2005.

Orientador: Ernesto Ruppert Filho
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Motores elétricos de indução. 2. Motores elétricos – controle eletrônico. 3. Indutores elétricos. I. Ruppert Filho, Ernesto. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: Digital processors performance estimation applied to low cost, three-phase electrical machines
Palavras-chave em Inglês: Induction electric machine, Electric machines – electronic control, Electric inductors.
Área de concentração: Automação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Manoel Luis de Aguiar, Maurício Ferreira Magalhães e Edson Adriano Vendrúsculo.
Data da defesa: 18/05/2005

Resumo

Este trabalho tem como objetivo apresentar um avaliador de desempenho de processadores digitais para utilização em aplicações de controle de motores de indução trifásicos, especialmente aquelas destinadas a aplicações de baixo custo e adequadas a produtos de consumo. A concepção destes produtos exige que o processador seja corretamente dimensionado uma vez que o custo deste dispositivo é significativo face ao custo total do acionamento. O método de avaliação aqui proposto utiliza um conjunto de métricas que permite predizer se um dado processador irá atender às restrições de tempo impostas pela aplicação, de forma a escalonar todas as tarefas que a compõe, bem como estimará o tamanho da memória de programa necessária para implementá-la.

Os dados de diversos processadores disponíveis comercialmente foram aplicados no método de avaliação aqui proposto e um deles foi utilizado no desenvolvimento de um protótipo experimental, onde se coletaram dados para verificar a eficácia do avaliador. Estes resultados, bem como as divergências entre o real e o avaliado, estão apresentados neste trabalho.

Abstract

The aim of this work is to show a digital processor performance simulator, to be used in three-phase induction motor control, specially those which are used in low cost products. The conception of these products demands the correct processor's specification, because its cost is expressive facing the overall drive's cost. The proposed simulation method uses a metrics set which enables to predict if one microprocessor will be in compliance with timing constraints imposed by the application in order to schedule all the software tasks, as well will estimate the necessary program memory size to implement it.

Several commercially available microprocessor's data, were used with this simulation method and one of them were chose to be used in an experimental laboratory prototype, in order to collect data to verify the effectiveness of the proposed method. The results, as well the divergences between experimental and simulation, are shown in this work.

Aos meus pais, Mário e Nair, pelo eterno incentivo, dedico este trabalho.

Agradecimentos

Ao amigo e professor Marco Antonio Robert Alves pela acolhida no início deste trabalho e pelo incentivo.

Ao professor Edmundo da Silva Braga pela confiança a mim atribuída.

Ao professor Ernesto Ruppert Filho pela oportunidade de realizar este trabalho, pela orientação e pelo tempo dispensado em inúmeras conversas.

Ao Marcos Vinicius Lazarini pela inestimável ajuda no desenvolvimento do *software* básico.

Aos professores das disciplinas cursadas pela contribuição para minha formação acadêmica.

E a todos que de alguma forma contribuíram para a realização deste trabalho.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

Símbolo	Descrição	Unidade
α_k	Ângulo do instante de início do *k-ésimo* pulso do MLP	rad/s
δ_k	Largura do *k-ésimo* pulso do MLP	rad
$\delta(t)$	Função Delta de Dirac	-
ϕ	Ângulo de fase	rad
ϕ_{ag}	Fluxo magnético no entreferro	Wb
Δ	Valor do incremento entre dois acessos consecutivos à tabela do seno	-
ΔV_e	Erro médio de tensão devido ao tempo morto	V
Θ_a , Θ_b , Θ_c	Palavras digitais referentes às amostras de tensão das fases “a”, “b” e “c”	-
ω	Freqüência angular	rad/s
ω_1	Velocidade angular da componente fundamental de saída do conversor	rad/s
ω_c	Freqüência angular de corte do FPB	rad/s
ω_M	Máxima freqüência angular sintetizável na saída do conversor	rad/s
ω_{mr}	Velocidade angular do rotor	rad/s
ω_{ms}	Velocidade angular síncrona do motor	rad/s
ω_{sn}	Velocidade angular nominal	rad/s
ω_s	Velocidade angular de alimentação do motor	rad/s
ω_{sl}	Velocidade angular de escorregamento	rad/s
ω_{sw}	Freqüência angular de amostragem	rad/s
AGND	Terra analógico	-
AH, BH, CH	Sinal de controle das chaves superiores da ponte inversora (sinal de *gate*)	-
AH_c, BH_c, CH_c	Sinal de controle das chaves superiores da ponte inversora (saída do MCU)	-
AL, BL, CL	Sinal de controle das chaves inferiores da ponte inversora (sinal de *gate*)	-
AL_c, BL_c, CL_c	Sinal de controle das chaves inferiores da ponte inversora (saída do MCU)	-
ALR	Instrução genérica Aritmética Lógica e Relacional	-
AN*k*	*k-ésimo* canal analógico do MCU	-
a_n	Componente co-senoidal da série de Fourier	-
b_n	Componente senoidal da série de Fourier	-
CA	Corrente alternada	-
CA1	Fase 1 da tensão alternada	-
CA2	Fase 2 da tensão alternada	-
CAD	Conversor analógico para digital	-
CC	Corrente contínua	-
CDA	Conversor digital para analógico	-
C_i	Tempo de execução da *i-ésima* tarefa de *software*	s
ciclos	Número de ciclos de máquina de uma instrução do *software*	-
CPU	Unidade central de processamento (*Central Processing Unit*)	-
D	Ciclo de trabalho (*duty-cycle*)	-
DEC	Instrução genérica Desvio Condicional	-
DEI	Instrução genérica Desvio Incondicional	-
D_i	Prazo (*deadline*) para a execução da *i-ésima* tarefa do *software*	s
DSP	Processador digital de sinais (*Digital Signal Processor*)	-
E_{ag}	Tensão eficaz no entreferro	V
e_{ag}	Tensão no entreferro	V
EEPROM	Memória não volátil, apagável eletricamente	-
E_r	Tensão eficaz induzida no enrolamento do rotor	V
Erro (%)	Erro percentual entre experimento e avaliação	-
exec_time	Tempo de execução do *software*	s
f	Freqüência	Hz
f_1	Freqüência fundamental de saída do conversor	Hz
FPB	Filtro passa-baixas	-
freq (V_i)	Freqüência de execução do *i-ésimo* vértice de uma tarefa do *software*	-

Símbolo	Descrição	Unidade
f_s	Freqüência síncrona do motor / freqüência da rede elétrica	Hz
f_s^*	Referência da freqüência síncrona	Hz
f_{sl}	Freqüência de escorregamento	Hz
$f_{s\,min}$	Freqüência síncrona mínima de excitação do motor	Hz
f_{sn}	Freqüência síncrona nominal do motor	Hz
f_{sw}	Freqüência de chaveamento ou portadora	Hz
G_1	Sinal de disparo do *gate* do TRIAC 1	-
G_2	Sinal de disparo do *gate* do TRIAC 2	-
G_d	Sinal de comando do freio	-
+/-HVDC	Barramento de alta tensão CC (*High Voltage DC*), polaridade positiva e neg.	V
i	Índice da tabela	-
I_{BUS}	Realimentação de corrente do barramento CC	A
IG	Instrução genérica	-
IGBT	Transistor da ponte inversora (*Insulated Gate Bipolar Transistor*)	-
IHM	Interface homem-máquina	-
I_{lim}	Limite de corrente eficaz do enrolamento de estator	A
I_m	Corrente eficaz de magnetização	A
i_m	Corrente de magnetização	A
I_{max}	Corrente eficaz máxima do enrolamento de estator	A
$instr\_size$	Tamanho de uma instrução genérica	*byte*
INT	Interrupção de *software*	-
I/O	Entrada e saída digital	-
I_r	Corrente eficaz do enrolamento do rotor	A
I'_r	Corrente eficaz do enrolamento do rotor referenciada ao estator	A
I_s	Corrente eficaz do enrolamento de estator	A
I_{sa}, I_{sb}, I_{sc}	Corrente eficaz do enrolamento de estator para as fases a, b e c	A
I_{sense}	Amostra de corrente do barramento CC	A
j	Notação complexa	-
k	Variável	-
K	Ganho do controlador V/Hz	-
k_1, k_2, k_3,k_4	Constantes de proporcionalidade no equacionamento do motor de indução	-
L_m	Indutância de magnetização por fase	H
L_s	Indutância por fase do enrolamento de estator	H
$LU(n)$	Limite de utilização do processador para n tarefas do *software*	-
LUT	Técnica de busca direta à tabela (*Look-up Table*)	-
m_a	Índice de modulação em amplitude	-
MAR	Instrução genérica movimentação e armazenagem	-
MCU	Microcontrolador (*Microcontroller unit*)	-
mem_size	Tamanho de memória de uma tarefa	*byte*
m_f	Índice de modulação em freqüência	-
MI	Motor de Indução	-
MLP	Modulador por largura de pulso	-
MSF	Máxima freqüência sintetizável	Hz
n	Variável	-
N	Constante digital	-
N_r	Número de espiras por fase do enrolamento do rotor	-
N_s	Número de espiras por fase do enrolamento de estator	-
Núm. de ciclos IG	Número de ciclos da instrução genérica	-
p	Número de pulsos do MLP	-
P	Número de pólos do motor	-
PC	Computador pessoal	-
PCk	k-*ésima* entrada digital da porta C do MCU	-
$prob\ (e_{ij})$	Probabilidade de execução dos vértices de uma tarefa do *software*	-
p.u.	Sistema “por unidade”	-
Q_{1-6}	Transistores da ponte inversora	-

Símbolo	Descrição	Unidade
RAM	Memória de dados	-
RISC	Processador com conjunto de instruções reduzidas	-
RMA	Algoritmo da Taxa Monotônica	-
ROM	Memória de programa	-
R_r,	Resistência elétrica por fase do enrolamento do rotor	Ω
R_r	Resistência elétrica por fase do enrolamento do rotor referenciada ao estator	Ω
R_s	Resistência elétrica por fase do enrolamento de estator	Ω
s	Escorregamento	-
SC	Sinalizador de sobre corrente	-
SD	Sinal de comando do desligamento dos *gate drives* (*shut down*)	-
$sgn(i)$	Função sinal da corrente	-
SIC	Número de ciclos de máquina da rotina de sintetização da senóide	-
ST	Sinalizador de sobre tensão	-
SUB	Instrução genérica chamada de subrotina	-
t	Tempo	s
T0, T1, T2, T3	Temporizadores digitais	-
Tamanho IG	Tamanho da instrução genérica	*bytes*
t_c	Contagem do *timer* de temporização do T_{sw}	-
T_c	Temperatura do encapsulamento do transistor	°C
T_{clk}	Período do ciclo de máquina do processador	s
t_d	Tempo morto (*dead-time*)	s
T_{em}	Conjugado eletromagnético do motor	Nm
t_f	Tempo para o transistor desligar (*fall time*)	s
T_i	Período da *i-ésima* tarefa do *software*	s
T_j	Temperatura da junção do transistor	°C
t_k	Instante da *k-ésima* amostra de tensão	s
T_k	Instante de início do *k-ésimo* período do MLP	s
T_{max}	Conjugado eletromagnético máximo do motor	Nm
T_n	Conjugado eletromagnético nominal do motor	Nm
T_{OFF}	Tempo da chave desligada	s
T_{ON}	Tempo da chave ligada	s
t_r	Tempo para o transistor ligar (*rise time*)	s
T_{sw}	Período de chaveamento ou período de amostragem	s
$total\_ciclos$	Número total de ciclos de máquina de um *software*	-
U	Fator de utilização do processador	-
$u(t-\delta)$	Função degrau	-
$v(t)$	Tensão instantânea	V
v_a , v_b , v_c	Tensão de fase na saída do modulador	V
V1 – V6	Vértices do *software*	-
$V_{a,b,c}$	Tensão média por fase na saída do modulador	V
V_{aN}	Tensão de fase na saída do modulador referenciada no ponto "*N*"	V
v_{ao}	Tensão de saída do modulador com relação ao terra virtual "*o*"	V
V_{ao}	Tensão máxima de saída do modulador com relação ao terra virtual "*o*"	V
V_{BUS}	Realimentação de tensão do barramento CC	V
v_{contr}	Tensão de controle das chaves	V
V_{contr}	Tensão máxima do sinal de controle das chaves	V
$v_{contr\ a,\ b,\ c}$	Tensão de controle das chaves para as fases a, b e c	V
$v_{contr}(t_k)$	Amplitude da *k-ésima* amostra de tensão do sinal de controle	V
V_d	Tensão contínua do barramento CC	V
V_{ge}	Tensão *gate* emissor do transistor	V
V_{max}	Tensão máxima de fase do motor (limite máximo)	V
V_{min}	Tensão mínima de fase do motor (limite mínimo)	V
$V_{rms\ 1}$	Tensão eficaz da componente fundamental	V
$V_{rms\ LL}$	Tensão eficaz de linha	V
V_s	Tensão eficaz de fase do estator / tensão de alimentação do motor	V

Símbolo	Descrição	Unidade
V_s^*	Referência da tensão do enrolamento de estator	V
V_{sense}	Amostra de tensão do barramento CC	V
V_{SM}	Palavra digital de referência da tensão de saída do inversor	-
V_{sn}	Tensão eficaz nominal de fase do motor	V
v_{sw}	Tensão da referência de chaveamento	V
V_{sw}	Tensão máxima da referência de chaveamento	V
V_{th}	Tensão de limiar da porta lógica (*threshold*)	V
W_a, W_b, W_c	Palavras digitais referente à T_{ON}	-
W_{da}, W_{db}, W_{dc}	Palavras digitais referente à T_{OFF}	-
W_k	Palavra digital referente à largura do k-*ésimo* pulso do MLP	-
$X[i]$	i-*ésimo* valor da tabela para a função seno	-
XIRQ	Entrada de interrupção externa do MCU	-
X_m	Reatância por fase de magnetização do rotor	Ω
X_r	Reatância por fase do enrolamento do rotor	Ω
$x_r(t)$	Saída do FPB	-
X_r'	Reatância por fase do enrolamento do rotor referenciada ao estator	Ω
X_s	Reatância por fase do enrolamento de estator	Ω
$x_s(t)$	Seqüência de pulsos gerada a partir das amostras da senóide	-

Sumário

Capítulo 1

INTRODUÇÃO

O objetivo inicial deste trabalho foi desenvolver um controle de velocidade escalar, de baixo custo, para motores de indução trifásicos, utilizando um inversor controlado por um microcontrolador (MCU), para ser usado em eletrônica de consumo, mais especificamente, em máquinas de lavar roupas, geladeiras e condicionadores de ar. A indústria brasileira, e mesmo grande parte das indústrias estrangeiras, utilizam hoje nesses aparelhos motores de indução monofásicos, com controle do tipo liga/desliga, que apresentam péssimo desempenho sob o ponto de vista técnico e também de conservação de energia, por ele ser barato o suficiente para ser utilizado em produtos de produção em massa, envolvidos em processos de grande competitividade de preços entre fabricantes. Como a parte de potência não apresenta grandes oportunidades de redução de custo, exceto idéias de topologias não convencionais tais como sugeridas em [1], [2] e [3]; o principal elemento passível de redução de custo é o microprocessador da placa de controle, uma vez que depois do estágio de potência ele costuma ser o componente mais caro da aplicação.

Para minimizar o microprocessador a ser utilizado é necessário, antes de mais nada, minimizar o *software* e reduzir as dependências de tempo entre as diversas tarefas que o compõe. Esta é uma das razões para a escolha do controle escalar utilizado neste trabalho. Implementou-se então, em laboratório, um controle escalar em malha aberta de um motor de indução trifásico, usando modulação em largura de pulso do tipo senoidal. No controle foi utilizado um microcontrolador MC68HC11A1, que é um dispositivo barato e que estava disponível no laboratório onde o experimento foi realizado. Verificou-se que o microcontrolador escolhido não era capaz de atender às restrições de tempo impostas pelo *software* desenvolvido para a aplicação. A questão então que ficou foi a seguinte: qual seria o “menor” microprocessador que poderia ser utilizado

neste caso, para assegurar que a aplicação fosse viável do ponto de vista custo? Para responder a esta pergunta iniciou-se um novo tipo de estudo buscando verificar, a partir dos dados disponibilizados pelos fabricantes de microprocessadores, qual seria o "menor" microprocessador possível de ser utilizado na aplicação. Este estudo levou em consideração o *software* inicialmente desenvolvido para o MCU MC68HC11A1. No entanto, foram utilizados neste trabalho dados disponíveis em Folhas de Dados de diversos microprocessadores que se encontram no mercado nacional e internacional entre microcontroladores e processadores digitais de sinais, para efeito de avaliação dos métodos aqui propostos, de verificação da capacidade de um dado microprocessador executar um dado *software*.

O desenvolvimento e os resultados deste estudo são relatados neste trabalho que inclui também uma apresentação não só do método de controle escalar de motores de indução trifásicos, mas também da metodologia de implementação prática deste controle. O material resultante desta implementação prática está descrito de forma minuciosa no corpo do trabalho. No Capítulo 2 são apresentados os conceitos básicos do motor de indução trifásico necessários ao entendimento dos demais capítulos. No Capítulo 3 é apresentado o controle escalar do motor de indução trifásico e detalhes para a sua implementação prática. No capítulo 4 são apresentadas as métricas para a estimativa de desempenho dos microprocessadores, as quais permitirão predizer se um dado microprocessador será capaz de executar um dado *software* atendendo aos requisitos de tempo, de memória e de desempenho exigidos pela aplicação. O Capítulo 5 apresenta várias simulações de desempenho, para vários microprocessadores, utilizando as técnicas mostradas no capítulo 4. O Capítulo 6 apresenta os resultados experimentais obtidos no protótipo de laboratório, comparando-os com os resultados das simulações. O Capítulo 7 apresenta as conclusões a que se chegou no trabalho e sugere ações para o seu aprimoramento. Além da contribuição que o trabalho apresenta no aspecto de estudar métricas para a estimativa de desempenho de microprocessadores, apresenta também uma descrição minuciosa de tudo o que foi feito na implementação do experimento de laboratório constituindo-se num material que

poderá servir de base de estudo para iniciantes nesta área de pesquisa e que, em geral, não é apresentado com tantas minúcias em livros, teses e artigos sobre esse assunto.

Capítulo 2

APLICAÇÕES E CONCEITOS BÁSICOS

2.1 Processos Domésticos

O processo (lógica de funcionamento) de um produto para uso doméstico, via de regra, é bastante simples no que diz respeito à sua motorização. Geralmente é um acionamento do tipo liga/desliga no qual atuam temporizadores eletromecânicos ou eletrônicos que geram um "perfil" de velocidade para a carga, baseado no tempo de motor ligado e de motor desligado.

Aplicações de uso residencial tais como o controle de refrigeradores, condicionadores de ar, lava-roupas e automação residencial em geral, tradicionalmente tem utilizado motores de indução monofásicos com velocidade fixa para comandar seus compressores e demais dispositivos rotativos. Estes eletrodomésticos regulam a temperatura ou a velocidade relacionada ao seu processo através do acionamento liga/desliga, operando dentro de uma faixa de histerese, fazendo com que o motor opere entre a sua capacidade total (ligado) e capacidade zero (desligado).

Se por um lado esta implementação é barata, por outro, apresenta uma série de desvantagens. O processo de controle causa desconforto ao usuário, a eficiência é reduzida, as perdas na máquina, inerentes ao ciclo liga/desliga, são maiores se comparadas com uma operação contínua e a faixa operacional é limitada. As sucessivas partidas do motor causam distúrbios na rede elétrica, geram interferências eletromagnéticas, bem como o aquecem devido ao regime severo de operação transitória e, finalmente, a operação em velocidade máxima geralmente acarreta elevados níveis de ruídos acústicos desagradáveis [4]. Este tipo de acionamento possui resposta lenta, com uma tendência a apresentar sobre elevações e como já dito, são substancialmente ineficientes.

Como exemplo de aplicação deste tipo de acionamento pode se citar um refrigerador ou um condicionador de ar de uso doméstico que funcionam

continuamente na sua capacidade máxima e, portanto, consomem uma significativa quantidade de energia. Desde que o elemento principal do consumo de energia é o compressor, os fabricantes deste tipo de produto estão sempre buscando ganhos de eficiência no sistema de refrigeração. Nos sistemas atuais o compressor utiliza um motor de indução monofásico com um enrolamento de partida e um de marcha e, em sistemas onde se deseja maior eficiência, é comum a utilização de um capacitor de marcha (permanente) conectado entre os enrolamentos de partida e marcha. Os capacitores de partida normalmente são encontrados em produtos de maior potência, tais como aqueles empregados em refrigeração comercial ou condicionadores de ar, em refrigeradores de uso doméstico eles não são utilizados. A figura 2.1 representa esquematicamente o acionamento do motor para refrigeração doméstica e mostra o conceito de partida comumente empregado nestes produtos, que se baseia na utilização de um relé do tipo PTC (*Positive Temperature Coeficient*) cuja resistência aumenta de acordo com a elevação de temperatura provocada pela corrente que circula pela sua pastilha.

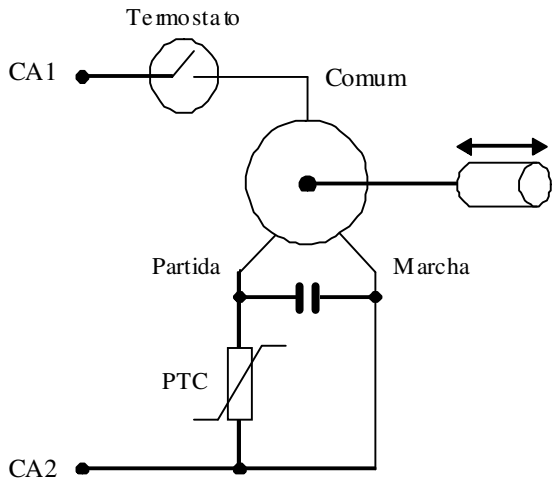


Figura 2.1 – Acionamento de um refrigerador convencional

Na partida, a pastilha do PTC está fria, apresentando uma baixa resistência e conseqüentemente permitindo a circulação de uma corrente elevada (corrente de partida do motor) através do enrolamento de partida. Após alguns segundos esta corrente irá aquecer a pastilha do PTC fazendo com que sua resistência

aumente até praticamente desconectar o enrolamento de partida da rede elétrica. Em aplicações com capacitor de marcha, o enrolamento de partida permanece energizado através deste. Após a partida o motor irá funcionar na sua velocidade máxima, partindo e parando toda vez que o termostato assim comandar.

Um outro exemplo de aplicação doméstica que utiliza intensivamente motores elétricos são produtos para lavanderia, tipicamente lavadoras de roupas. Os produtos atuais baseiam-se em um acionador que utiliza motores de indução monofásicos, geralmente do tipo PSC (*Permanent Split Capacitor*) com dois enrolamentos, um para cada sentido de rotação. Cada um é comandado por uma chave estática bidirecional (TRIAC), como indicado na figura 2.2 [5].

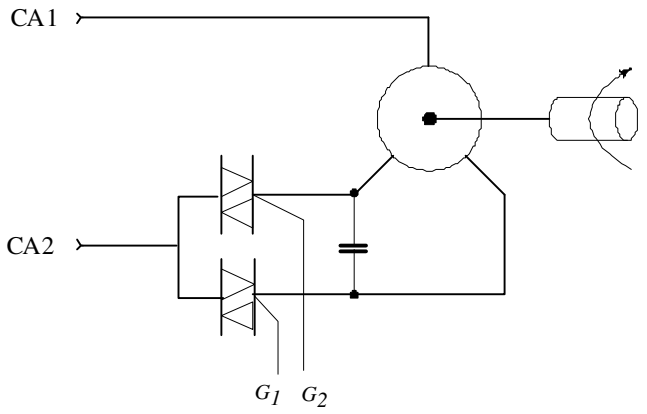


Figura 2.2 – Acionamento de uma lavadora de roupas convencional

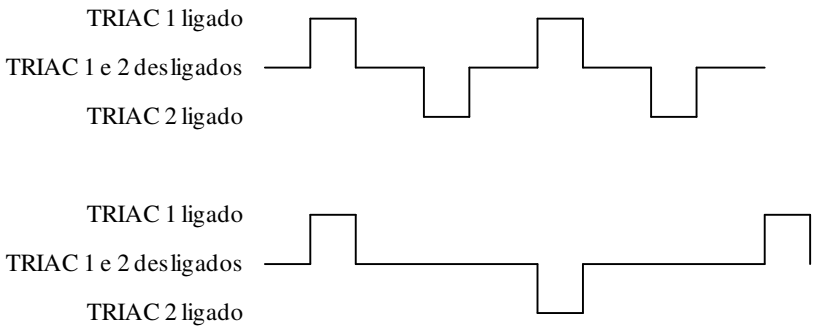


Figura 2.3 – Variação do regime de agitação em uma lavadora de roupas

Os TRIACs são comandados alternadamente para provocar a agitação necessária ao processo. Diferentes regimes de agitação são obtidos por variações do tempo do motor desligado, figura 2.3, entretanto, variações de velocidade de centrifugação são extremamente limitadas.

Associado a estes TRIACs existe um temporizador que controla o número de ciclos da rede elétrica que serão aplicados aos enrolamentos do motor, instituindo o processo de liga/desliga que, além dos distúrbios ocasionados pelos sucessivos regimes transitórios, provoca um outro efeito colateral que é o surgimento de uma componente harmônica de baixa freqüência na rede elétrica.

Uma placa eletrônica que realiza o acionamento do tipo liga/desliga para motores fracionários, custa aproximadamente 3,0 dólares considerando apenas a matéria prima (componentes). O preço de venda (com impostos, encargos trabalhistas, margem de remuneração, custos diretos e indiretos, etc) tipicamente é maior ou igual ao dobro do custo dos componentes, portanto para um controle deste tipo pode-se esperar um preço na faixa de 6,0 a 7,0 dólares.

2.2 Estado da arte

Substituindo-se o motor de indução monofásico, que opera com velocidade fixa, por um acionador composto por um motor de indução trifásico, alimentado a partir de uma rede monofásica e comandado por um inversor do tipo fonte de tensão modulado em largura de pulso (MLP), o desempenho do dispositivo comandado aumentará significativamente. Nesta situação o motor opera continuamente e regula a velocidade de forma suave e precisa, reduzindo-a ou aumentando-a de forma a obter a refrigeração ou a velocidade desejada, de acordo com a aplicação.

O ruído elétrico e as interferências eletromagnéticas geradas pelas sucessivas partidas e paradas do motor serão eliminadas, ou fortemente reduzidas, o ruído acústico pode ser reduzido, a eficiência do compressor aumenta quando operado de forma contínua e, finalmente o desempenho global do motor é melhorado. De forma geral os sistemas que oferecem controle contínuo de velocidade são inerentemente mais eficientes, porém mais complexos e ainda mais caros. Entretanto, de acordo com [6] e [7], sistemas de condicionamento de ar que utilizam inversores de freqüência proporcionam uma elevada redução no consumo de energia elétrica, quando comparado a sistemas de capacidade constante. Devido à elevada eficiência energética o custo do

inversor pode ser compensado em 3 a 4 anos apenas considerando a redução no consumo de energia elétrica.

No caso específico de sistemas de refrigeração pode-se obter ganho de eficiência reduzindo-se a velocidade do compressor sempre que não for necessária a capacidade máxima de refrigeração. As altas velocidades de operação ficam reservadas para refrigeração rápida, ou para quando o refrigerador for abastecido com alimentos com temperatura superiores à temperatura interna do gabinete, ou seja, o uso de um controle de velocidade variável neste tipo de produto proporcionará uma redução no consumo de energia elétrica.

O estado da arte em acionamento para produtos de consumo (eletrodomésticos) baseia-se na substituição do motor de indução monofásico por um trifásico, sem capacitor de partida ou de marcha, com velocidade variável e sem a necessidade de freio eletromecânico, tal como em uma lavadora de roupas convencional. Este motor é acionado por um conversor eletrônico que permite a variação da velocidade, a minimização da emissão de ruídos elétricos na rede, a redução por métodos digitais do conteúdo harmônico entregue ao motor e, aliado a um grande número possibilidades de controle digital, este tipo de aplicação atenderá aos requisitos de economia e desempenho cada vez mais severos impostos por diversos paises.

Uma placa eletrônica que realiza este tipo de acionamento, novamente considerando motores fracionários, custa aproximadamente 15,0 dólares apenas a matéria prima. Para o preço de venda pode-se esperar algo na faixa de 25,0 a 30,0 dólares.

Uma comparação direta com o acionamento do tipo liga/desliga mostra uma diferença considerável, porém como mencionado, a eficiência energética amortizará em alguns anos este custo extra.

Nos dois casos considerados, acionamentos do tipo liga/desliga e velocidade variável, a contribuição do processador no custo da matéria prima fica em torno de 20 a 25% do total.

2.3 O Mercado

Dentre os produtos de consumo passíveis de utilização deste tipo de acionamento foram citados os produtos para refrigeração doméstica (refrigeradores, *freezers* e condicionadores de ar) e lavanderia (lavadoras de roupas automáticas). Estes produtos ou requerem dispositivos que permitam a variação contínua da sua velocidade, ou permitem a utilização destes no sentido de melhorar seu desempenho.

De acordo com a referência [8], o mercado brasileiro para estes produtos tem-se mantido relativamente estabilizado nos últimos cinco anos, figura 2.4. Porém os volumes produzidos são significativos e o número de empresas que participam deste mercado ainda é relativamente reduzido, constituindo um mercado inexplorado para o acionamento de velocidade variável.

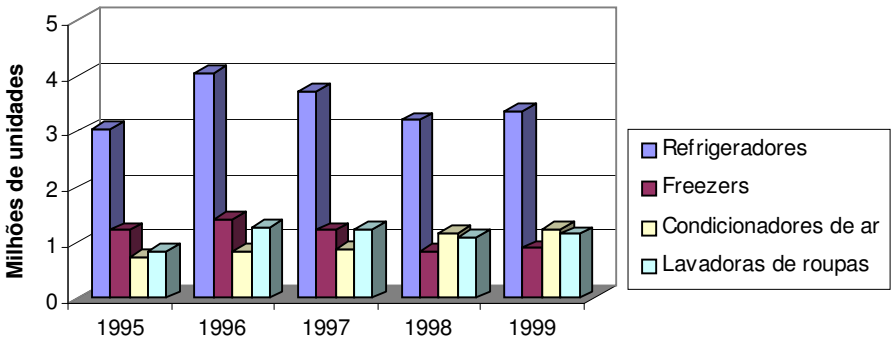


Figura 2.4 – Evolução do mercado brasileiro

Os controles com velocidade variável destinados a aplicações de consumo tem sofrido grande progresso impulsionados por recentes desenvolvimentos na Academia e empresas do setor, onde o Japão tem se destacado no desenvolvimento de sistemas de ar-condicionado (para todas as faixas de potência – inclusive a residencial) comandados por inversores, cujo objetivo principal é o aumento da eficiência e a redução do consumo de energia elétrica. De acordo com [9], a figura 2.5 ilustra a evolução deste mercado no Japão e exemplifica sua potencialidade.

A referência [10] compara o mercado de eletrodomésticos japonês com o americano na utilização de produtos de uso doméstico controlados por inversores

(especificamente condicionadores de ar, refrigeradores, lavadoras de roupas e aspiradores de pó), figura 2.6. Observa-se que 37% do mercado japonês já utiliza inversores, contra apenas 1% do americano, o que ressalta o grande potencial desta tecnologia.

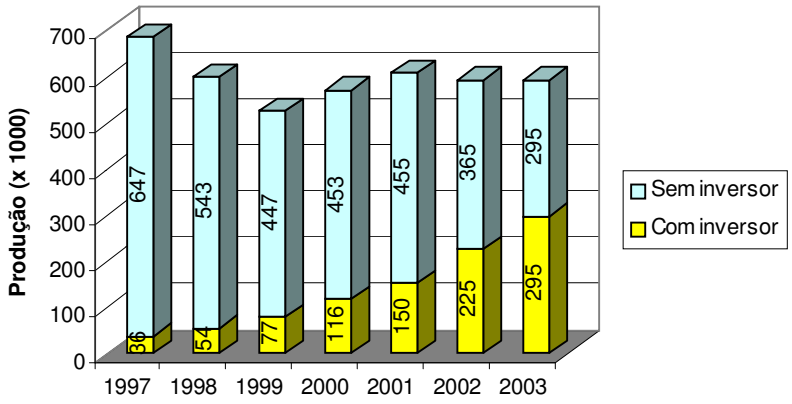


Figura 2.5 – Evolução do mercado japonês de condicionadores de ar de baixa potência utilizando inversores de freqüência

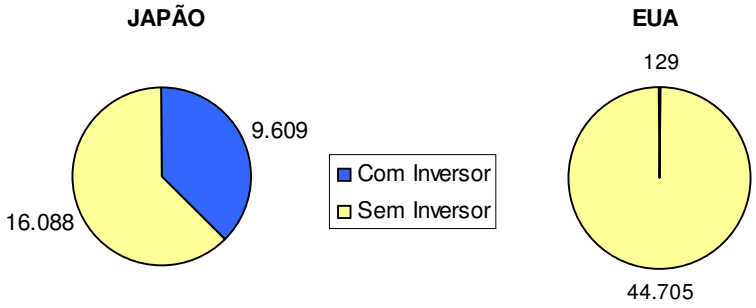


Figura 2.6 – Comparação do mercado japonês com o americano, em x1000 unidades/ano

2.4 Conceitos básicos sobre motores de indução trifásicos

Uma rápida revisão dos conceitos de máquinas elétricas de indução faz-se necessário para se introduzir a técnica de acionamento utilizada ao longo deste trabalho.

A velocidade síncrona da máquina de indução (MI) é diretamente proporcional à freqüência de alimentação quando o estator da máquina é alimentado por uma fonte de tensão alternada trifásica balanceada.

$$\omega_{ms} = \frac{2}{P}\omega_s = \frac{4\pi f_s}{P} \; rad/s \tag{2.1}$$

Sendo ω_{ms} a velocidade angular síncrona mecânica do motor cuja freqüência, denominada freqüência síncrona, é f_s, ω_s a velocidade angular elétrica de alimentação em rad/s e P o número de pólos da máquina. Se a velocidade angular mecânica do rotor for ω_{mr}, a relação entre a velocidade do campo girante e a do rotor é dada por:

$$\omega_{sl} = \omega_{ms} - \omega_{mr} \tag{2.2}$$

Nesta equação ω_{sl} é a velocidade angular de escorregamento do motor, com freqüência correspondente chamada freqüência de escorregamento f_{sl}. Dela é obtida a grandeza s, denominada escorregamento da máquina.

$$s = \frac{\omega_{ms} - \omega_{mr}}{\omega_{ms}} \tag{2.3}$$

A frequência de escorregamento será:

$$f_{sl} = s \cdot f_s \tag{2.4}$$

O circuito equivalente por fase da máquina, em estado estacionário e com os parâmetros do rotor referidos ao estator, é mostrado na figura 2.7, onde o subscrito "s" indica parâmetros do estator, "r" do rotor e "ag" do entreferro entre o estator e o rotor [11].

O fluxo magnético no entreferro ϕ_{ag} gira à velocidade síncrona, relativa aos enrolamentos de estator. Em conseqüência, uma força eletromotriz, freqüentemente chamada tensão do entreferro (E_{ag}) é induzida em cada fase do estator, para uma dada freqüência de excitação f_s. V_s é a tensão eficaz de fase, R_s a resistência elétrica do enrolamento de estator por fase e $X_s = \omega_s \cdot L_s$ é a reatância de dispersão por fase do enrolamento de estator. A corrente do estator I_s apresenta uma componente de magnetização I_m a qual gera o fluxo ϕ_{ag}. A diferença I_s-I_m é a corrente do enrolamento do rotor I_r vista do lado do estator [11] e [12], representada por I'_r. Na figura 2.7, os parâmetros R'_r e X'_r são a resistência e a reatância de dispersão do enrolamento do rotor por fase, respectivamente, referenciadas ao estator e dadas pelas equações abaixo.

$$R'_r = \left(\frac{N_s}{N_r}\right)^2 \cdot R_r \quad \text{e} \quad X'_r = \left(\frac{N_s}{N_r}\right)^2 \cdot X_r \tag{2.5}$$

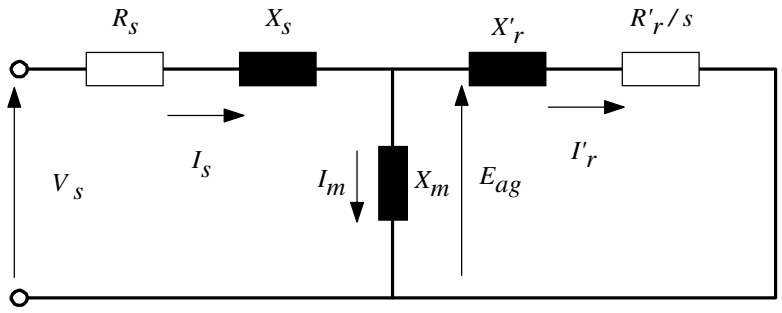


Figura 2.7 – Circuito equivalente por fase do MI

Da análise do circuito magnético pode-se escrever:

$$N_s \cdot \phi_{ag} = L_m \cdot i_m \tag{2.6}$$

Sendo N_s o número de espiras de uma fase do estator e L_m a indutância de magnetização por fase. Da lei de Faraday, pode-se escrever:

$$e_{ag} = N_s \cdot \frac{d\phi_{ag}}{dt} \tag{2.7}$$

Como a excitação é considerada senoidal, o fluxo no entreferro pode ser escrito como $\phi_{ag}(t) = \phi_{ag}\, sen\, \omega_s t$. Substituindo-o em (2.7), resulta em:

$$e_{ag} = N_s \cdot \omega_s \cdot \phi_{ag} \cdot \cos \omega_s t \tag{2.8}$$

Cujo valor eficaz vale:

$$E_{ag} = k_1 \cdot f_s \cdot \phi_{ag} \tag{2.9}$$

$$k_1 = \sqrt{2} \cdot \pi \cdot N_s \tag{2.10}$$

De maneira análoga pode-se determinar a tensão induzida no enrolamento do rotor E_r, para uma determinada freqüência de escorregamento $f_{sl} = s \cdot f_s$ [Hz] vista do lado do estator, então:

$$E_r = k_1 \cdot f_{sl} \cdot \phi_{ag} \tag{2.11}$$

A interação de ϕ_{ag} com o campo produzido pelas correntes do rotor resulta no torque eletromagnético do motor que pode ser escrito como:

$$T_{em} \cong k_2 \cdot \phi_{ag} \cdot I_r \tag{2.12}$$

Com T_{em} normalmente dado em Nm ou kgm.

Utilizando-se um motor com rotor tipo gaiola pode-se escrever:

$$E_r = R_r \cdot I_r + jX_r \cdot I_r \tag{2.13}$$

Utilizando as equações (2.11), (2.13) e considerando $R_r >> X_r$, pode-se escrever:

$$I_r \cong k_3 \cdot \phi_{ag} \cdot f_{sl} \tag{2.14}$$

Como E_{ag} é muito mais significativa do que a queda de tensão sobre os enrolamentos de estator, pode-se fazer a seguinte aproximação para freqüências mais altas:

$$V_s \cong E_{ag} \tag{2.15}$$

Combinando as equações (2.9), (2.12), (2.14) e considerando (2.15):

$$T_{em} \cong k_4 \cdot \phi_{ag}^2 \cdot f_{sl} = k_4 \cdot \left(\frac{V_s}{k_1 \cdot f_s} \right)^2 \cdot f_{sl} \tag{2.16}$$

Da figura 2.7, pode-se escrever:

$$V_s = E_{ag} + (R_s + jX_s) \cdot I_s \tag{2.17}$$

Usando (2.9) e (2.15), obtêm-se:

$$V_s \cong k_1 \cdot \phi_{ag} \cdot f_s \tag{2.18}$$

Nas equações acima *k1, k2, k3* e *k4* são constantes de proporcionalidades.

De acordo com (2.18), qualquer redução na freqüência de alimentação f_s sem que seja alterada a tensão terminal V_s, causará um aumento do fluxo magnético, ϕ_{ag}, no entreferro. Como os motores de indução são projetados para operarem no joelho da sua curva de magnetização, com o objetivo de melhor aproveitamento do material magnético, o aumento do fluxo irá saturar o motor. Este efeito provoca um aumento na corrente de magnetização, distorce a corrente e tensão de linha, aumenta as perdas no núcleo magnético, bem como as perdas no cobre do estator e produz um ruído acústico de alta intensidade.

Da mesma forma, uma redução no fluxo também deve ser evitada para se manter a capacidade de torque do motor. Portanto, o controle da freqüência síncrona abaixo do seu valor nominal é geralmente seguido pela redução da tensão de fase da máquina, tal que o fluxo seja mantido constante, conforme a equação (2.18). Para uma determinada faixa de variação de tensão abaixo do seu valor nominal e com velocidade angular síncrona abaixo do seu valor nominal ω_{sn}, o fluxo do entreferro poderá ser mantido constante no seu valor nominal e o torque permanecerá constante em seu valor nominal T_n, de acordo com a equação (2.16). A variação do torque em função da velocidade angular síncrona da máquina está representado na figura 2.8.

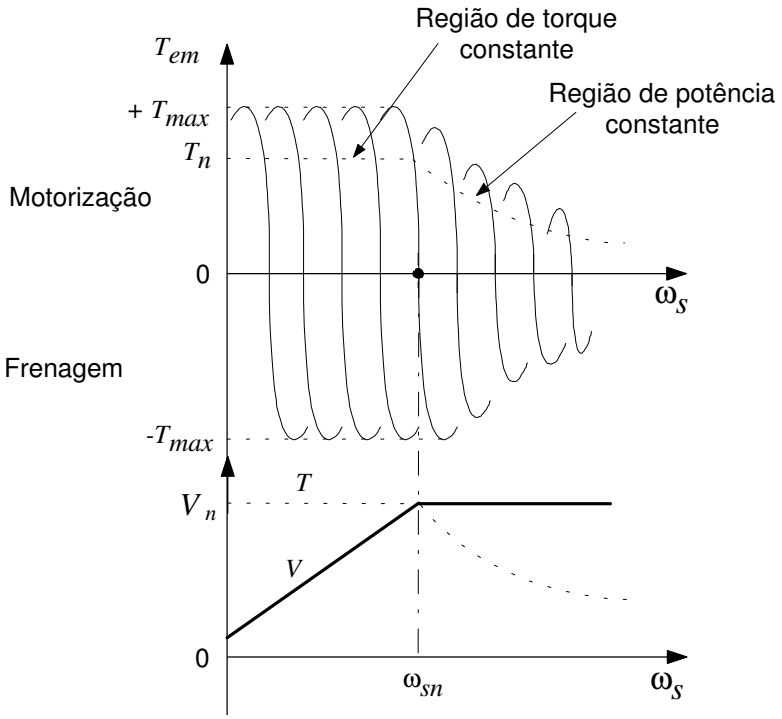


Figura 2.8 – Curva torque x velocidade para controle de freqüência síncrona variável e fluxo constante

Entretanto, operar a máquina mantendo-se o fluxo constante requer uma malha fechada de fluxo, tornando o controle complicado, dada a dificuldade de se medir o fluxo. Desta forma, o fluxo é controlado indiretamente operando a máquina através de um controlador Volts/Hertz constante, abaixo e acima da freqüência nominal, exceto – como será visto adiante - para a faixa de baixas

freqüências onde a queda de tensão sobre a resistência do estator deixa de ser desprezível.

Capítulo 3

CONTROLE ESCALAR DA MÁQUINA DE INDUÇÃO TRIFÁSICA

3.1 Introdução

A técnica de controle escalar baseia-se no controle da freqüência síncrona e da amplitude da tensão aplicada no enrolamento de estator da máquina, sendo que os sinais de comando e os de realimentação são quantidades contínuas e proporcionais às suas respectivas variáveis. O controle de velocidade por meio da variação de freqüência e da tensão permite a operação abaixo e acima da velocidade nominal. Esta capacidade pode ser atrativa em muitas aplicações dado que, devido à sua construção robusta, a maior parte dos motores de indução pode operar em até duas vezes a sua velocidade nominal sem problemas mecânicos. Esta técnica de controle é, ainda hoje, bastante utilizada em acionamentos industriais, cujos processos não requeiram alto desempenho e é a preferida na maior parte das aplicações de velocidade variável de baixo custo. Para fazer uso de toda capacidade de torque do motor tanto na partida quanto em baixas velocidades, a relação V/Hz deve ser aumentada em baixas freqüências para compensar a queda de tensão na resistência elétrica do enrolamento de estator que passa a ser significativa quando comparada com a tensão no entreferro.

3.2 Controle V/Hz

A figura 3.1 mostra a relação típica tensão-freqüência do controlador V/Hz, o qual apresenta uma não linearidade para freqüências acima da nominal. É permitido comandar a freqüência acima de f_{sn} desde que a tensão aplicada aos terminais dos enrolamentos de estator sature em seu valor nominal V_{sn}. Para muitas aplicações de velocidade variável onde pequenas variações na velocidade da máquina são toleradas, um simples sistema de malha aberta utilizando um controlador Volts/Hertz (V/Hz) com compensação de tensão em baixas freqüências pode ser satisfatório [13]. A figura 3.2 mostra um diagrama de blocos

desse controle no qual f_s^* e V_s^* são as referências de entrada para o modulador em largura de pulso.

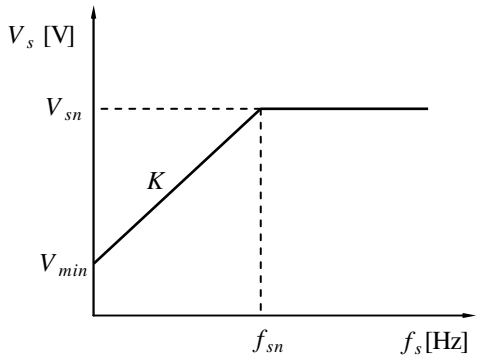


Figura 3.1 – Perfil de controle para V/Hz

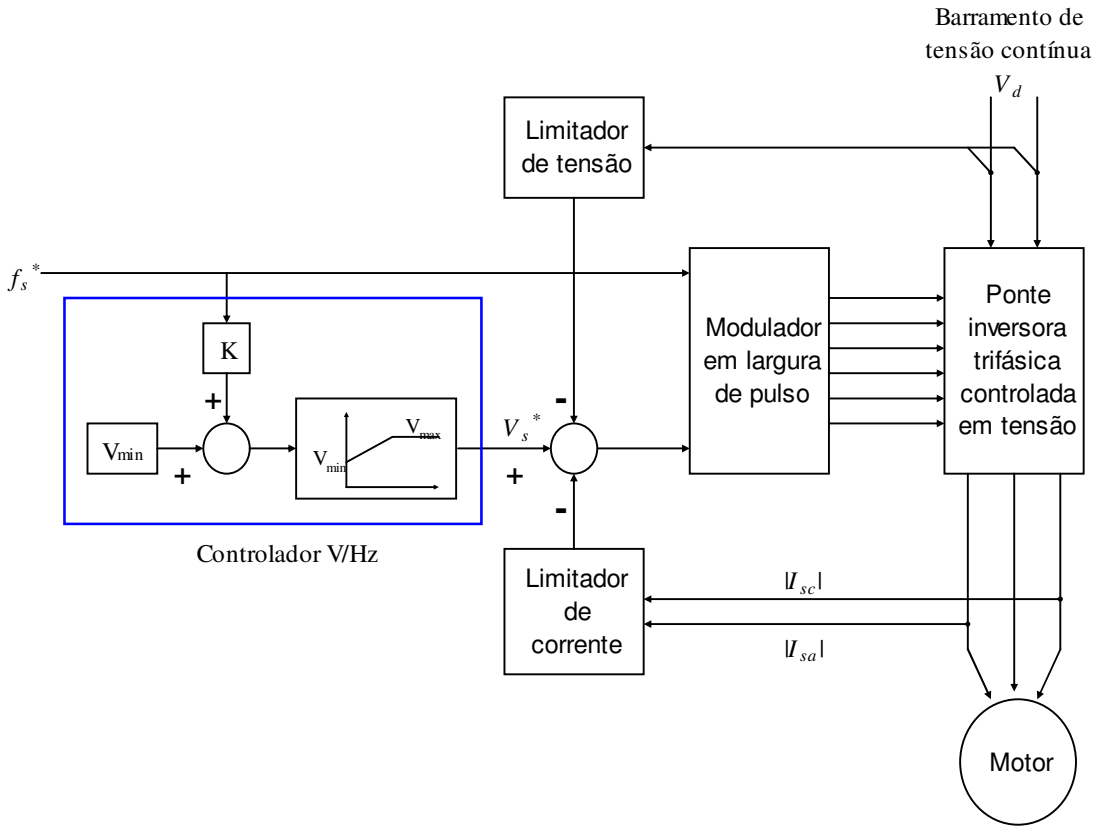


Figura 3.2 – Diagrama de blocos do controle V/Hz

O esquema de controle é definido como controlador Volts/Hertz porque a tensão de comando V_s^* é gerada diretamente do sinal de referência de freqüência

f_s^* , através de um ganho constante K, tal como indicado na figura 3.2. Para proteção do conversor normalmente se utiliza um limitador de corrente, cujos sensores de corrente podem estar tanto no barramento de corrente contínua quanto nas saídas do inversor e um limitador de tensão no barramento de corrente contínua que limita as sobretensões no banco de capacitores e no motor.

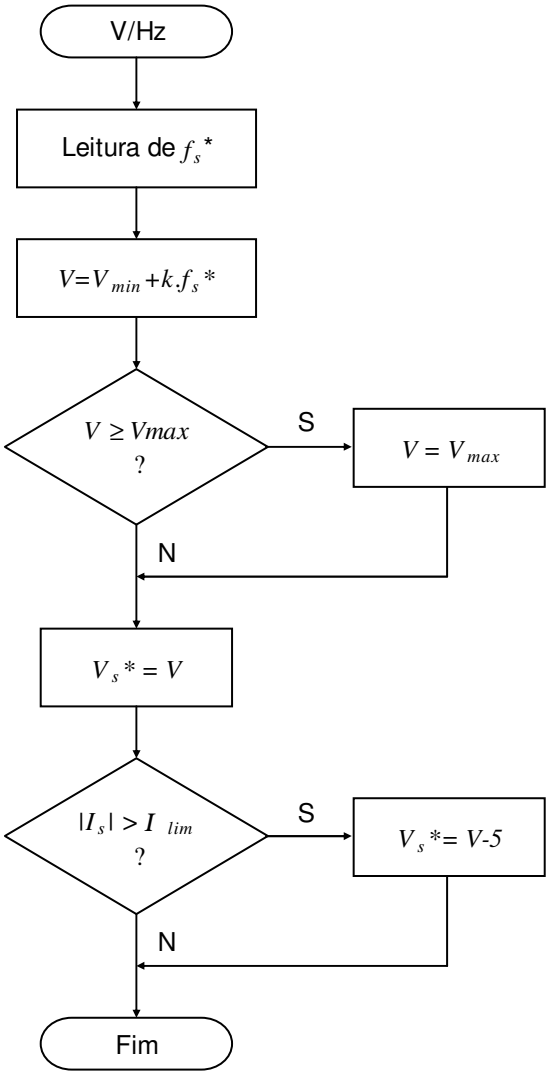


Figura 3.3 – Algoritmo V/Hz

A figura 3.2 representa estas proteções amostrando duas das correntes dos enrolamentos de estator, as quais atuam na malha de controle reduzindo a tensão de comando V_s^* (referência da tensão aplicada ao enrolamento de armadura do motor) com o objetivo de reduzir a corrente elétrica no enrolamento

de armadura. Este processo pode ser mantido até um determinado limite, além do qual o conversor deve ser desligado para proteger os transistores do inversor contra falha por sobrecorrente. A figura 3.3 apresenta o algoritmo do controlador V/Hz com proteção contra sobrecorrente.

3.3 Controle V/Hz compensado com "*Voltage Boost"*

A figura 3.4 mostra a relação típica tensão-freqüência do controlador V/Hz compensado por *"voltage boost"*, a qual apresenta duas não linearidades. Para aplicações onde se deseja resposta em baixas velocidades é permitido comandar a freqüência abaixo de f_{s_min} desde que a tensão sature em V_{min}. Esta compensação é denominada *"voltage boost"*. Para se obter a relação entre V_s e f_s , V_s deve ser variada de acordo com a expressão (3.1).

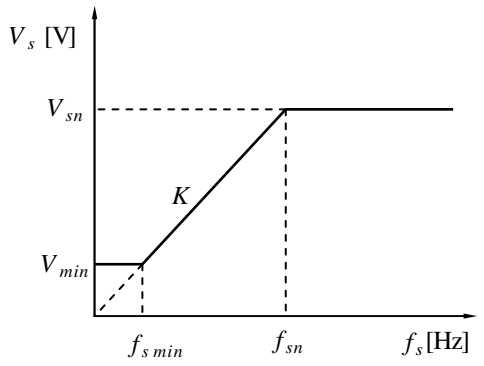


Figura 3.4 – Perfil de controle para V/Hz com "*voltage boost"*

$$V_s = V_{\min} + K \cdot u\left(f - f_{s\min}\right) \cdot f_s \tag{3.1}$$

Sendo V_{min} escolhido para produzir fluxo nominal em baixas velocidades, a constante K é escolhida para fornecer a tensão nominal V_{sn} para a freqüência nominal f_{sn} e $u(f\text{-}f_{s\ min})$ é a função degrau, sendo que $u(f\text{-}f_{s\ min}) = 0$ para $f < f_{s\ min}$, impondo V_{min} para freqüências inferiores a $f_{s\ min}$ e $u(f\text{-}f_{s\ min}) = 1$ para $f \geq f_{s\ min.}$

O controlador V/Hz com compensação por *"voltage boost"* está representado na figura 3.5 onde foi inserida a saturação de tensão para baixas freqüências de operação do motor.

Variando-se a freqüência e a tensão de excitação dentro da faixa compreendida por $f_{s\ min}$ e $f_{s\ n}$ da figura 3.4, o fluxo magnético e a velocidade de

escorregamento serão mantidos constantes, o torque eletromagnético permanecerá constante e a velocidade do rotor variará com a freqüência de excitação, como pode ser observado na figura 3.6 onde, para cada freqüência de excitação f_s, existirá uma velocidade mecânica síncrona ω_{ms} correspondente a um dado torque. A curva característica torque por velocidade para pequenos escorregamentos desloca-se paralelamente a si mesma.

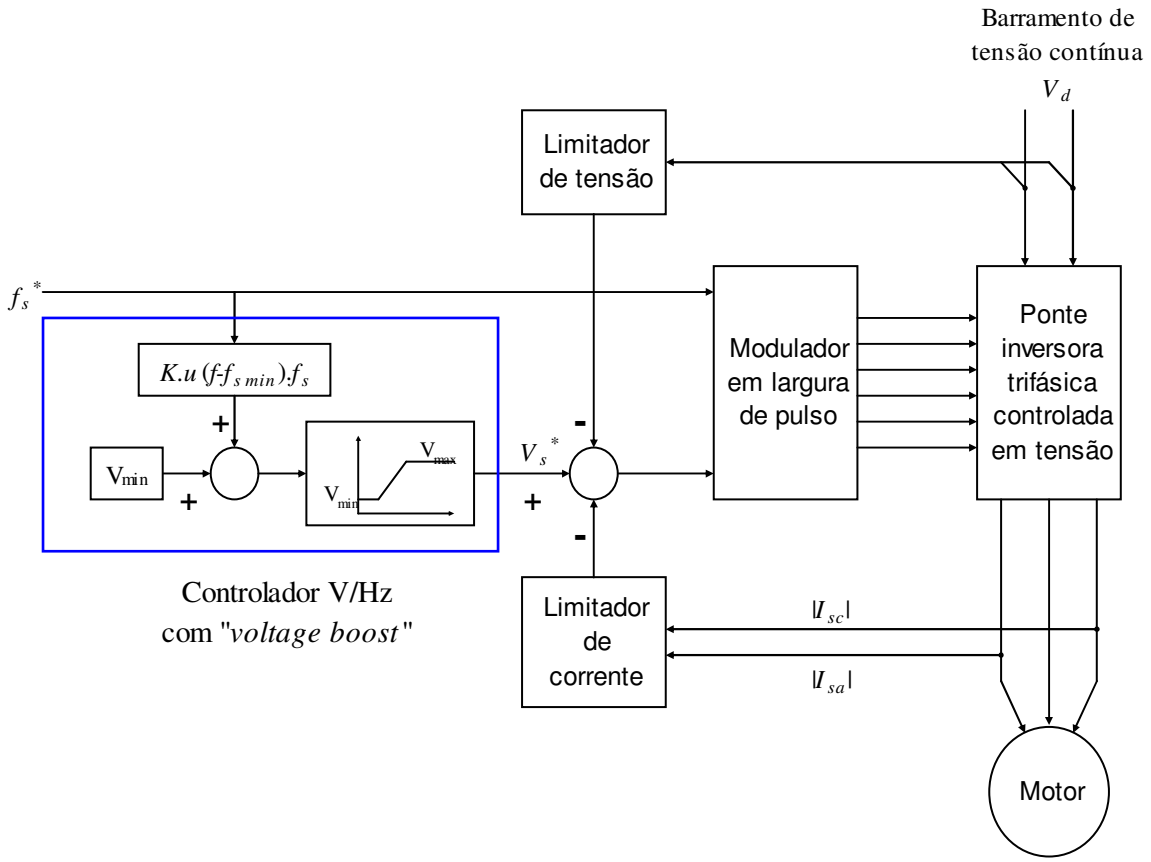


Figura 3.5 – Controle V/Hz compensado com *"voltage boost"*

A figura 3.7 mostra o fluxograma para implementação do algoritmo V/Hz com *"voltage boost"*, onde V_d é a tensão do barramento de corrente contínua do inversor. Pode-se observar a inclusão da saturação da tensão para as baixas freqüências.

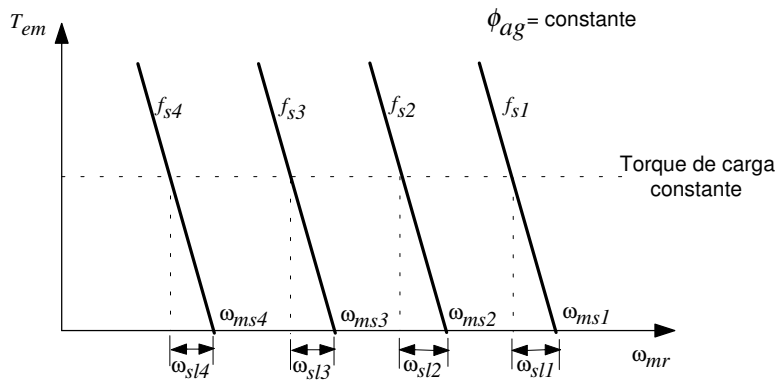


Figura 3.6 – Característica torque x velocidade para diferentes freqüências

Verifica-se então que para controlar a velocidade do motor é necessário variar tanto o valor eficaz da tensão de alimentação do motor como a sua freqüência de excitação.

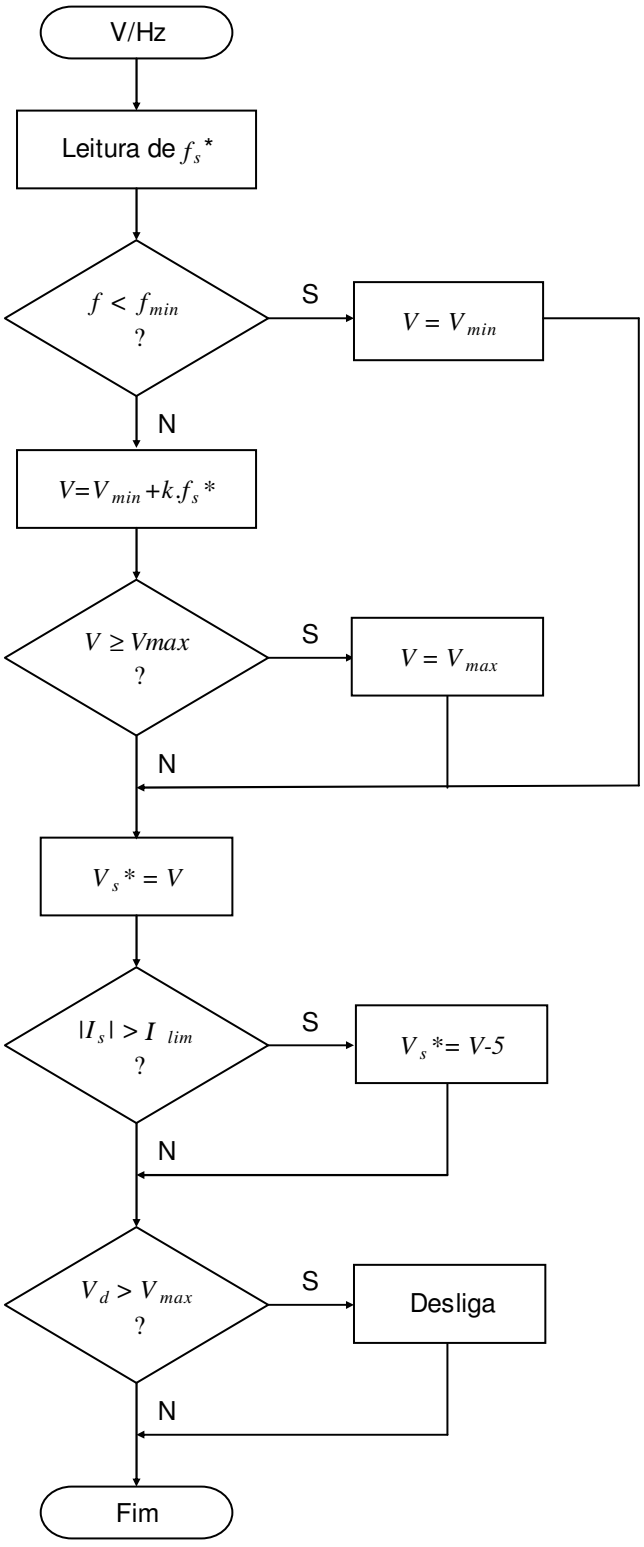


Figura 3.7 – Algoritmo V/Hz com *"voltage boost"*

3.4 Modulação em largura de pulso (MLP senoidal)

Mostrou-se na seção anterior que para controlar a velocidade do motor de indução trifásico é necessário variar o valor eficaz da tensão de alimentação do motor bem como a sua freqüência. Para tal utiliza-se o inversor trifásico que é um conversor eletrônico de potência, que converte uma tensão contínua em três tensões, cujas componentes fundamentais são defasadas entre si de 120°, através do controle da condução da corrente elétrica que passa pelas suas chaves semicondutoras.

Uma técnica comumente utilizada para isso é a modulação da tensão em largura de pulso (MLP) do tipo senoidal. A figura 3.8 ilustra o diagrama de blocos deste tipo de acionamento onde um conversor de entrada, normalmente um retificador controlado ou não, produz uma tensão contínua (V_d), um estágio intermediário filtra a tensão, ou corrente contínua, dependendo do tipo de inversor utilizado (existem dois tipos básicos, inversor tipo fonte de tensão e fonte de corrente; este trabalho considera o tipo fonte de tensão, já que o tipo fonte de corrente se aplica melhor a sistemas de alta potência) e o estágio de saída denominado inversor, que converte a tensão contínua em uma tensão alternada, com freqüência e amplitude variáveis. As correntes de saída de um inversor trifásico ideal seriam senóides puras defasadas de 120°. Entretanto, nos circuitos práticos isso não ocorre, pois elas apresentam harmônicas fazendo com que correntes distorcidas circulem pelos enrolamentos de estator do motor.

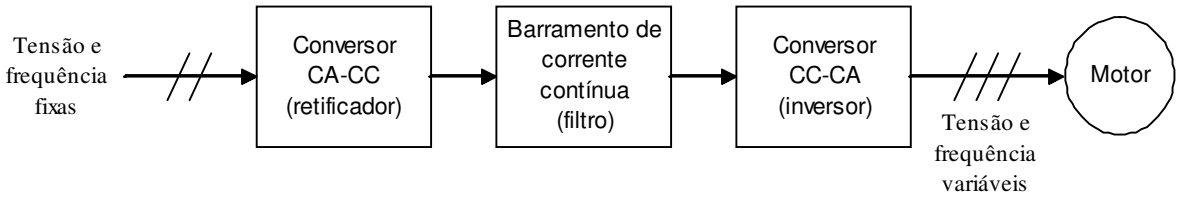


Figura 3.8 – Conversor CA/CA tipo fonte de tensão

A tensão eficaz de saída por fase do inversor (V_a) pode ser controlada comandando-se as chaves eletrônicas do conversor CC-CA com uma freqüência (f_{sw}) cujo período (T_{sw}) é dado pela expressão (3.2) e, ajustando-se o os tempos

T_{ON} e T_{OFF}, tempo de chave ligada e desligada respectivamente. Este método é chamado de modulação em largura de pulso (MLP).

$$T_{sw} = T_{ON} + T_{OFF} \tag{3.2}$$

O ciclo de trabalho (D) da chave é definido por (3.3) e é a razão entre o tempo de chave ligada (T_{ON}), com relação ao tempo total de chaveamento (T_{sw}), figura 3.10. Para uma freqüência de chaveamento (f_{sw}) fixa, T_{sw} é constante, desta forma para se variar *D* deve-se variar o T_{ON}, teoricamente de 0 a 100%.

$$D = \frac{T_{ON}}{T_{sw}} \tag{3.3}$$

Em uma MLP triangular, com freqüência de chaveamento constante, o sinal de controle das chaves, o qual controla o estado ligado ou desligado, é gerado comparando-se uma tensão de controle contínua (v_{contr}) com um sinal periódico (v_{sw}), que geralmente é um sinal triangular ou dente-de-serra com amplitude constante, o qual estabelece a freqüência de chaveamento (f_{sw}), figura 3.9. A representação gráfica da tensão média de saída, idealmente filtrada, é mostrada na figura 3.10.

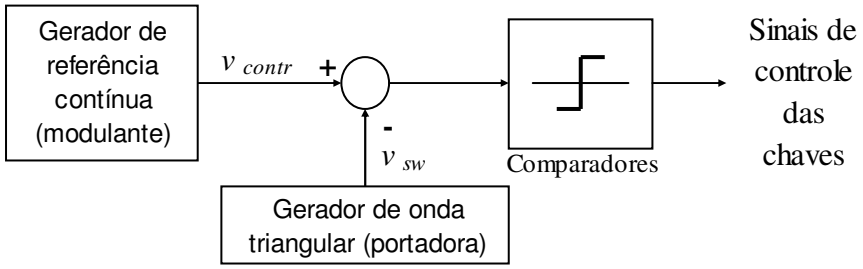


Figura 3.9 – Diagrama de blocos da geração do sinal MLP com tensão de controle contínua

Na discussão anterior considerou-se que o sinal modulante (v_{contr}) era um sinal contínuo, ou de variação muito lenta no tempo, com o objetivo de gerar uma tensão média contínua na saída, cuja amplitude pode ser controlada pelo ciclo de trabalho das chaves do modulador.

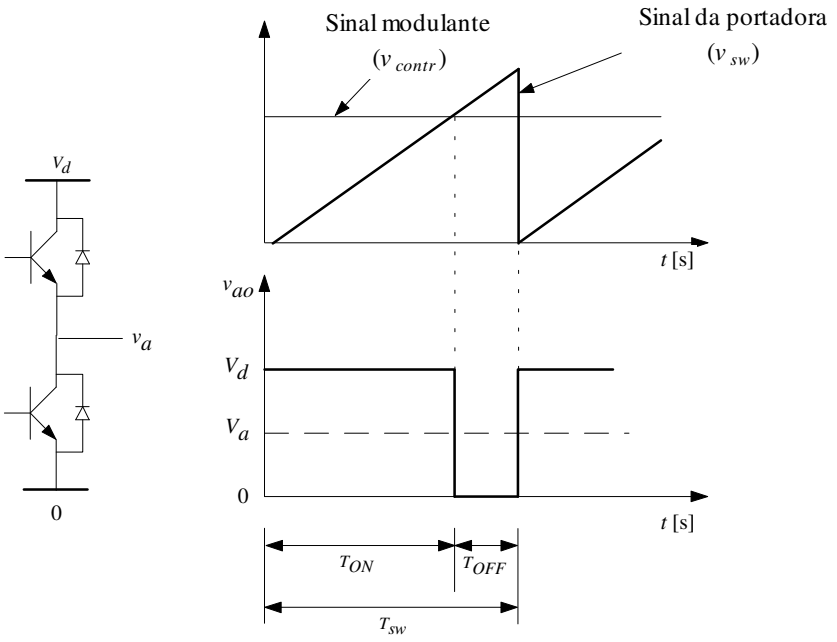


Figura 3.10 – Formas de ondas do MLP com tensão de controle constante

Na modulação senoidal o sinal modulante, ou de controle, deve ser senoidal, na freqüência de saída desejada (componente fundamental de excitação do motor), o qual deve ser comparado com um sinal triangular, figura 3.11. Novamente, a freqüência do sinal triangular estabelece a freqüência de chaveamento ou freqüência portadora. O sinal de controle tem freqüência f_1, sendo f_{sw} muito maior do que f_1, como representado na figura 3.12.

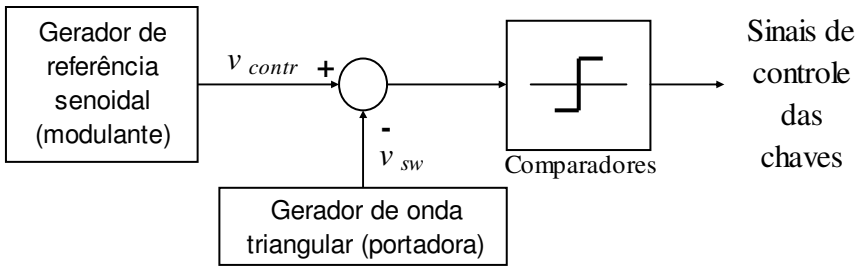


Figura 3.11 – Diagrama de blocos da geração do sinal MLP senoidal monofásico

A lei de formação da figura 3.12(b) se baseia na comparação dos sinais v_{contr} e v_{sw}, resultando nas seguintes tensões de saída, independente do sentido da corrente de saída [14]:

$$v_{contr} > v_{sw} \quad \text{Q1 está ligado} \quad v_{ao} = 0{,}5.V_d \tag{3.4}$$

$$v_{contr} < v_{sw} \quad \text{Q2 está ligado} \quad v_{ao} = -0{,}5.V_d \tag{3.5}$$

Para facilitar o entendimento assume-se que o ponto médio "*o*" do barramento CC, figura 3.12, esteja disponível, embora na maior parte dos inversores isso não é necessário, como também não está disponível. Desta forma, a tensão de fase v_{ao} pode assumir valores positivos e negativos, tal como mostrado na figura 3.12(b). Na prática no entanto, a tensão de fase v_{aN} , referenciada no ramo "negativo" do barramento CC, assume apenas dois valores discretos, zero ou V_d, figura 3.13. Já a tensão de linha, por exemplo $v_{ab}= v_{aN} - v_{bN}$, assumirá valores de $-V_d$ e $+V_d$ [11], figura 3.14.

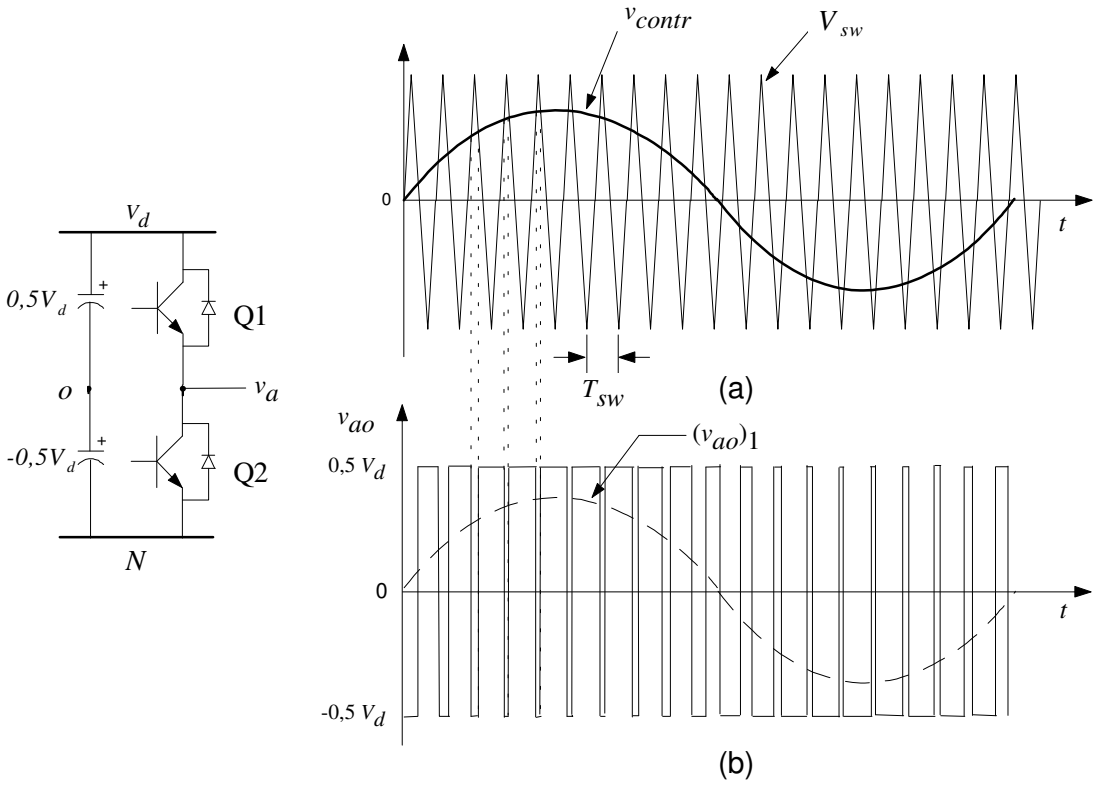


Figura 3.12 – Tensão de fase da MLP referenciada no ponto "*o*"

Dois parâmetros devem ser definidos para o modulador, o primeiro deles é o índice de modulação em amplitude (m_a), que é a relação entre as tensões máximas da figura 3.12(a), ou seja:

$$m_a = \frac{V_{contr}}{V_{sw}} \tag{3.6}$$

Sendo V_{contr} o valor máximo do sinal modulante e V_{sw} é o valor máximo do sinal triangular (portadora) da figura 3.12, com $m_a \leq 1$.

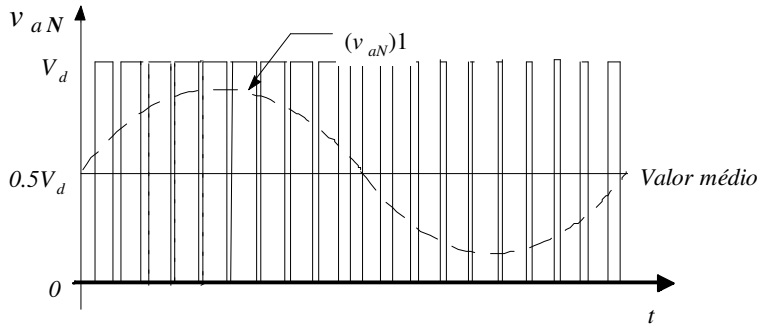


Figura 3.13 – Tensão de fase da MLP referenciada no ponto "*N*"

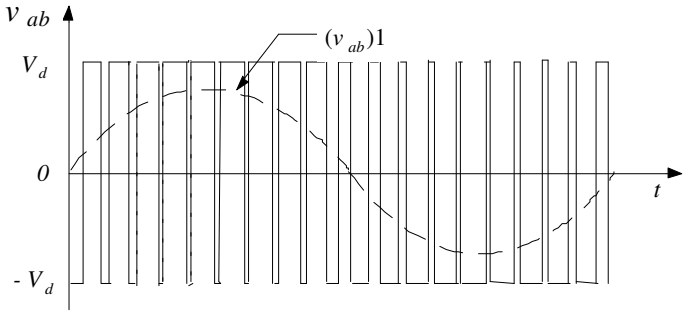


Figura 3.14 – Tensão de linha da MLP (v_{ab}= v_{aN} - v_{bN})

O sinal modulante pode ser definido por:

$$v_{contr}(t) = V_{contr} \cdot sen(\omega_1 t + \phi) \tag{3.7}$$

Onde ϕ é a diferença de fase entre o sinal modulante e o triangular (quando ϕ = 0, o MLP é dito síncrono) e ω_1 é a freqüência angular fundamental deste sinal. De acordo com [11], a tensão na saída do inversor é dada por:

$$v_{ao}(t) = \frac{v_{contr}(t)}{V_{sw}} \cdot \frac{V_d}{2} \quad \text{para} \quad V_{contr} \leq V_{sw} \tag{3.8}$$

Desta forma pode-se escrever, para a modulação síncrona que a primeira harmônica da tensão na carga é dada por:

$$(v_{ao})_1 = \frac{V_{contr}}{V_{sw}} \cdot \frac{V_d}{2} \cdot \operatorname{sen} \omega_1 t \; = \; m_a \cdot \frac{V_d}{2} \cdot \operatorname{sen} \omega_1 t \quad \text{para} \;\; m_a \leq 1{,}0 \tag{3.9}$$

Portanto, a componente fundamental na saída do inversor $(v_{ao})_1$ varia senoidalmente em fase com o sinal de controle. A expressão (3.9) mostra que o valor máximo da componente fundamental varia linearmente com o índice de modulação em amplitude, para $m_a \leq 1$. Desta forma a faixa de m_a compreendida entre 0 e 1 é referenciada como faixa linear. De acordo com [11], nesta faixa de operação, o MLP gera harmônicas múltiplas da freqüência de chaveamento.

O segundo parâmetro do modulador a ser definido é o índice de modulação em freqüência (m_f), dado pela expressão (3.10).

$$m_f = \frac{f_{sw}}{f_1} \tag{3.10}$$

Este parâmetro fornece o número de pulsos do MLP em um ciclo da senóide. Devido à relativa facilidade de se filtrar as componentes harmônicas de altas freqüências, é desejável utilizar freqüências de chaveamento tão altas quanto possível, de tal forma que a indutância dos enrolamentos do motor filtra a corrente que se aproximará de uma senóide. Entretanto, altas freqüências de chaveamento implicam em maiores perdas por chaveamento nas chaves eletrônicas do conversor CC-CA, portanto deve-se buscar um compromisso entre perdas e qualidade da corrente do motor. Para pequenos valores de m_f (tipicamente $m_f \leq 21$) o sinal triangular e o de controle devem ser sincronizados, isto requer que m_f seja um número inteiro. A razão para se utilizar um MLP síncrono, é que no assíncrono (m_f não inteiro), a relação entre as fases da portadora e do sinal modulante não é fixa, gerando um conjunto de pulsos na saída do inversor não repetitivo entre ciclos diferentes, introduzindo uma componente de corrente contínua e sub-harmônicas da freqüência fundamental, causando pulsações de torque e de velocidade para baixas freqüências, conhecidas como batimento de freqüência. Isso implica que a freqüência de chaveamento deve variar em função da freqüência fundamental de saída. De acordo com [15], a modulação assíncrona senoidal não representa nenhum inconveniente sensível quando comparada com a modulação síncrona desde que a freqüência de chaveamento seja muito maior que a fundamental de saída (da ordem de 30 a 40 vezes), nesta situação as amplitudes das sub-harmônicas tornam-se desprezíveis. Deve-se ressaltar que a modulação assíncrona é

tecnicamente mais fácil de se implementar uma vez que não há necessidade de se sincronizar portadora com o sinal modulante. Para se variar a freqüência fundamental de saída e manter f_{sw} próxima ao seu valor máximo, m_f deve variar como representado na figura 3.15, na qual $m_{f1} < m_{f2} < m_{f3} < m_{f4} < m_{f5} < m_{f6}$. Esta figura, de acordo com [11] e [16], representa uma relação teórica entre a freqüência de saída do inversor e a freqüência de chaveamento, onde se admite uma variação de f_{sw} com o objetivo de se manter m_f inteiro e múltiplo de 3, para se reduzir o conteúdo harmônico na saída do inversor, pois desta forma somente as harmônicas impares estarão presentes (f_{sw}, 3.f_{sw}, 5.f_{sw},...), já que as pares desaparecerão uma vez que os coeficientes cosenoidais da série de Fourier serão nulos [11] e [17]. Já a referência [17] representa esta mesma relação de forma mais próxima da realidade, tal como será visto nas figuras 3.26 e 3.27. Ainda de acordo com as referências [11] e [16], para impedir a ocorrência de *jitter* (indecisão) nas freqüências onde ocorrem mudanças dos números de pulsos, deve ser incluída uma pequena histerese, como apresentado na figura 3.15.

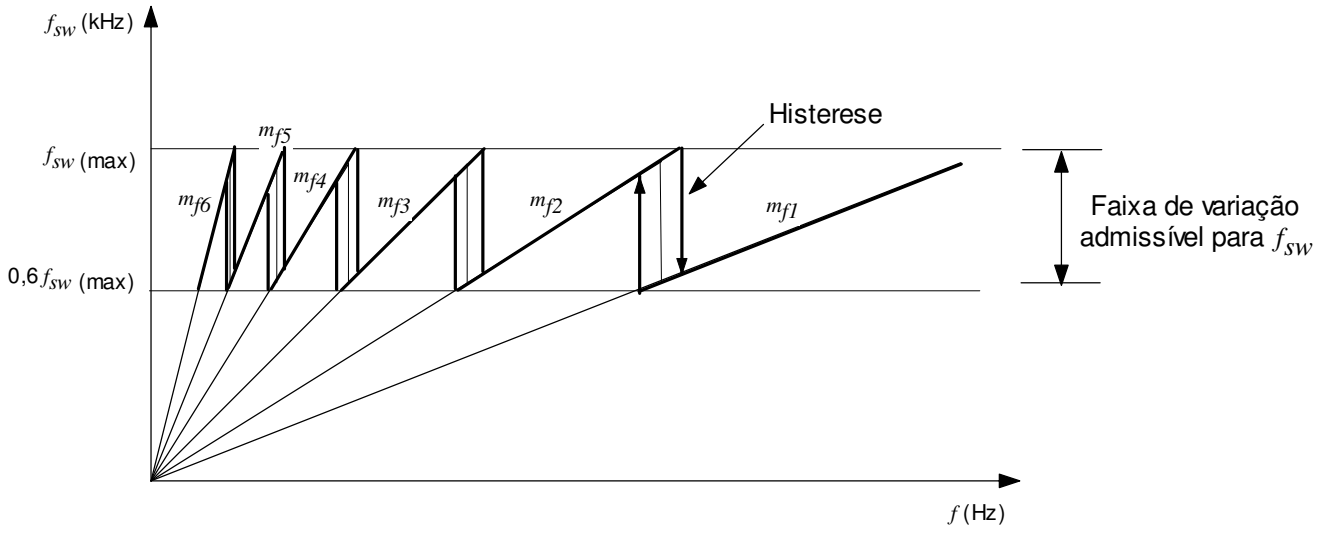


Figura 3.15 – Relação teórica entre freqüência de saída e a freqüência de chaveamento, parametrizada em m_f

3.5 Inversor MLP trifásico

De maneira similar aos inversores monofásicos, o objetivo dos inversores trifásicos que usam modulação por largura de pulso é sintetizar a amplitude e a

freqüência das tensões de saída do inversor, a partir de uma tensão de entrada continua V_d.

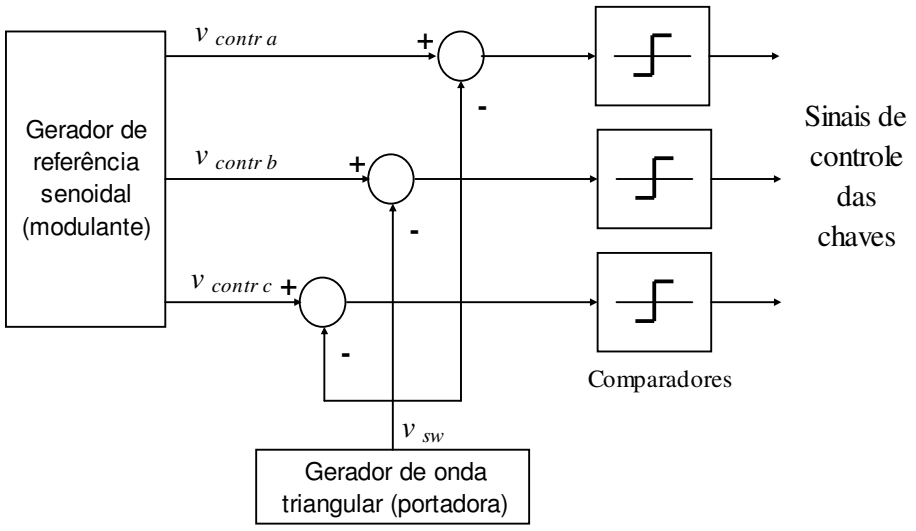


Figura 3.16 – Diagrama de blocos da geração do sinal MLP senoidal trifásico

Para obter as tensões de saída balanceadas, uma mesma forma de onda triangular é comparada com três sinais de controle senoidais, defasados de 120°, a figura 3.16 representa o diagrama de blocos desta topologia.

O inversor trifásico mais freqüente consiste de três pernas [11], uma para cada fase, tal como mostrado na figura 3.17. A tensão de saída para cada fase depende da tensão do barramento de corrente contínua e do estado (ligado ou desligado) das chaves.

A figura 3.17 mostra uma ponte inversora trifásica onde as chaves estão representadas por seis transistores genéricos (Q1 a Q6) comandados pelos sinais de controle AH, AL, BH, BL e CH, CL, com diodos de retorno em anti-paralelo, sendo V_d a tensão contínua do barramento e v_a, v_b e v_c são as tensões de saída. Uma restrição óbvia neste tipo de arranjo é que duas chaves de uma mesma perna nunca podem conduzir simultaneamente. Nos inversores atuais estas chaves são implementadas por transistores do tipo MOSFETs ou IGBTs.

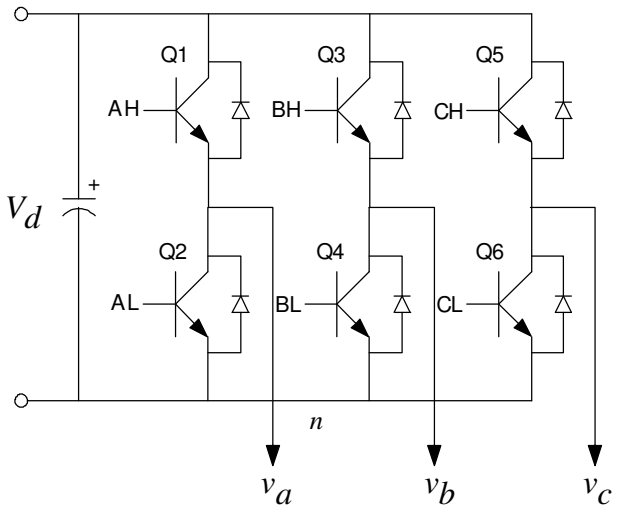


Figura 3.17 – Ponte inversora trifásica

3.5.1 Análise do inversor MLP trifásico

A forma geral da série de Fourier para um sinal periódico é dada por:

$$v(t) = \sum_{n=1}^{\infty} (a_n \cos n\omega t + b_n \operatorname{sen} n\omega t) \tag{3.11}$$

Sendo:

$$a_n = \frac{1}{\pi} \int_0^{2\pi} v(t) \cos n\omega t \; d\omega t \tag{3.12}$$

$$b_n = \frac{1}{\pi} \int_0^{2\pi} v(t) \operatorname{sen} n\omega t \; d\omega t \tag{3.13}$$

Quando o sinal $v(t)$ possuir simetria de um quarto de onda, apenas as harmônicas impares, entre aquelas que possuem componentes senoidais, existirão [18], portanto os coeficientes da equação (3.11) se reduzem a:

$$b_n = \frac{1}{\pi} \int_0^{2\pi} v(t) sen n\omega t \; d\omega t \qquad \text{para } n\text{= 1, 3, 5,...} \tag{3.14}$$

Sendo que $a_n = 0$ para n= 1, 3, 5,... e $b_n = 0$ para n= 2, 4, 6,... Então a equação (3.11) pode ser re-escrita por:

$$v(t) = \sum_{n=1,3,5...}^{\infty} b_n \operatorname{sen} n\omega t \tag{3.15}$$

De acordo com [12], a integral em (3.14) pode ser calculada pela soma dos p pulsos do MLP para meio ciclo da senóide (p=m_f/2) e pode ser escrita da forma a seguir:

$$b_n = \sum_{k=1}^{p} \frac{2V_d}{n\pi} \operatorname{sen} \frac{n\delta_k}{2} \left[\operatorname{sen} n\left(\alpha_k + \frac{\delta_k}{2} \right) - \operatorname{sen} n\left(\pi + \alpha_k + \frac{\delta_k}{2} \right) \right] \qquad (3.16)$$

Onde α_k é o instante de início do *k-ésimo* pulso do MLP, definido pela expressão (3.27) e δ_k é a largura deste pulso, ambos representados na figura 3.23. δ_k pode ser determinado diretamente da figura 3.23 simplesmente convertendo a *k-ésima* amostra da senóide ($ma.sen(\omega t)$) em uma largura de pulso, obtida pela multiplicação desta amostra pelo período de chaveamento (T_{sw}) [19], então:

$$\delta_k = T_{sw} \cdot m_a \cdot sen(\omega t_k) \qquad (3.17)$$

A tensão eficaz da componente fundamental de saída do inversor ($V_{rms\ 1}$) é dada pela contribuição de todos os pulsos do MLP para meio ciclo do sinal de saída do inversor [12] por:

$$V_{rms\ 1} = V_d \left(\sum_{k=1}^{p} \frac{\delta_k}{\pi} \right)^{1/2} \qquad (3.18)$$

Para $m_a \leq 1.0$, a componente fundamental varia linearmente com a taxa de modulação em amplitude e o valor de pico para a tensão de fase é dado por:

$$(V_a)_1 = m_a \cdot \frac{V_d}{2} \qquad (3.19)$$

Uma forma de se representar a tensão eficaz de linha, na faixa linear, independente do número de pulsos e, em termos do índice de modulação m_a [11] é:

$$V_{rms\ LL_1} = \frac{\sqrt{3}}{\sqrt{2}} (V_a)_1 = \frac{\sqrt{3}}{2\sqrt{2}} m_a V_d \cong 0.61 m_a V_d \qquad (3.20)$$

Observa-se que a tensão eficaz de saída fica limitada a aproximadamente a 60% da tensão do barramento de corrente contínua. De acordo com [20], esta é uma das maiores desvantagens do MLP senoidal, isto é, a pobre utilização da tensão do barramento de corrente contínua. Uma forma de melhorar a utilização da tensão do barramento CC é através da técnica de injeção de terceira harmônica, que melhora a sua utilização em torno de 15,5% [23].

3.5.2 Gerador de onda senoidal

O gerador de onda senoidal é fundamental para a implementação de um modulador MLP, seja ele digital ou analógico. Para o caso específico da implementação digital, existem diversos métodos para a implementação em *software* de tal gerador, os quais admitem otimizações quanto à velocidade (tempo de processamento), precisão e utilização de memória.

Dentre estes métodos, o mais utilizado é o "Busca Direta à Tabela", recomendado para aplicações onde não se requer elevada precisão e que possuam restrições de tempo de processamento [21].

Neste método, também conhecido como *Lookup Table – LUT*, os valores tabelados geram apenas uma aproximação do sinal senoidal, isso se deve aos erros de truncamento, de arredondamento e ao comprimento limitado da palavra digital. De forma geral, quanto maior a tabela, maior resolução ela fornecerá e conseqüentemente mais próximo de um sinal senoidal será a sintetização do sinal de saída. Os valores da função seno são armazenados em memória, como indicado na tabela 3.1. O comprimento desta tabela é um compromisso entre a quantidade de memória ROM disponível para armazená-la, contra a precisão desejada, ou seja, pode-se discretizar um ciclo da senóide de 1 em 1 grau, de ½ em ½, etc.

Na tabela 3.1, n é o número de amostras ou comprimento da tabela, i é o índice da tabela, com $0 \leq i \leq n$ e 0 (zero) é o endereço base. Este índice pode ser utilizado como parâmetro do ângulo e $X[i]$ é o valor da função seno, armazenado na $i\text{-}ésima$ posição da tabela, sendo $i*360°/n$ o ângulo em graus [22]. Levando-se em consideração que a função seno possui simetria de um quarto de onda, exceto pelo sinal, somente um quarto de onda precisa realmente ser armazenada na tabela. Desta forma, para um mesmo comprimento de tabela pode-se ter uma precisão maior, porém às custas de maior processamento a cada acesso à tabela.

De acordo com [21], [22] e [23], a freqüência da senóide gerada digitalmente depende do incremento delta (Δ) entre dois acessos sucessivos à tabela, do intervalo de tempo (T_{sw}) e do comprimento da tabela. Se delta for unitário todas as posições da tabela serão lidas seqüencialmente e a tabela é acessada a cada T_{sw} segundos. Por outro lado, se Δ for maior do que a unidade as

posições da tabela serão lidas com o incremento de delta, desta forma, a freqüência sintetizada será:

$$f_s = \frac{\Delta}{n \cdot T_{sw}} = \frac{\Delta}{n} \cdot f_{sw} \text{ Hz}, \quad \text{com} \quad \Delta \leq n/2 \tag{3.21}$$

Tabela 3.1 – Tabela dos valores do seno

Índice	Seno
n	$X[n] = sen[(n)*360/n]$
.	.
.	.
i	$X[i] = sen[(i)*360/n]$.
.	.
.	.
2	$X[2] = sen[(2)*360/n]$
1	$X[1] = sen[(1)*360/n]$
0	$X[0] = sen[(0)*360/n]$

A restrição para o máximo valor de delta deve ser feita para satisfazer o Teorema de Nyquist. Se delta for um número inteiro, somente múltiplos de $1/n.T_{sw}$ serão gerados, porém, se for permitido que ele seja um número real positivo, qualquer freqüência até a máxima freqüência sintetizavel (MSF) pode ser gerada. De acordo com [22] a MSF depende da forma com que a rotina de sintetização é implementada e do processador (tempo do ciclo de máquina), considerando-se que apenas esta rotina será executada pelo processador. A MSF é um valor teórico, pois geralmente não pode ser atingido na prática, dado que o processador também será utilizado para executar outras funções que compõem o *software* da aplicação, no entanto ela pode ser calculada por:

$$MSF = \frac{1}{2 \cdot T_{clk} \cdot SIC} \text{ Hz} \tag{3.22}$$

Sendo T_{clk} o tempo do ciclo de máquina em segundos e SIC o número de ciclos de máquina das instruções que compõem a rotina de sintetização da senóide.

De acordo com [24] o processo de reconstituição da senóide pode ser feito pelo diagrama de blocos da figura 3.18, onde as amostras $X[i]$ são submetidas a um conversor digital para analógico (CDA), gerando uma seqüência de pulsos

$x_s(t)$, a qual é limitada em banda pelo filtro passa-baixas (FPB), cuja saída $x_r(t)$ é o sinal recuperado. Matematicamente:

$$x_s(t) = \sum_{i=0}^{n} X[i] \cdot \delta\,(t - i\Delta T_{sw}) \tag{3.23}$$

Sendo $\delta\,(t)$ é a função delta de Dirac. A freqüência de corte do FPB (ω_c) deve ser posicionada como sendo maior do que a máxima freqüência sintetizável na prática (ω_M) e do menor que a freqüência de amostragem (ω_{sw}).

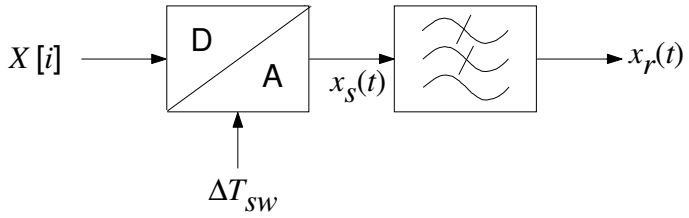


Figura 3.18 – Reconstituição da senóide

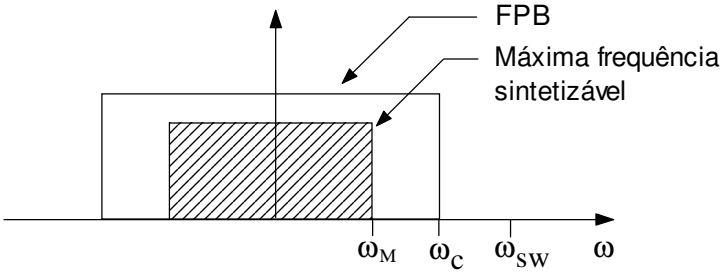


Figura 3.19 – Filtro limitador de faixa

Para satisfazer o teorema de Nyquist:

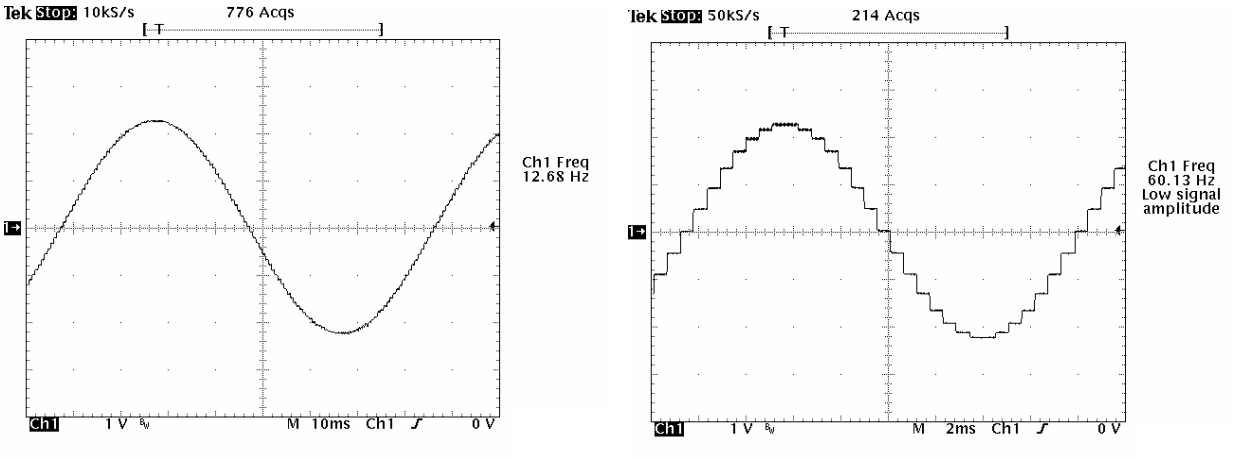
$$\omega_{sw} > 2\omega_M \tag{3.24}$$

A freqüência de corte será [24]:

$$\omega_M < \omega_c < \omega_{sw} \tag{3.25}$$

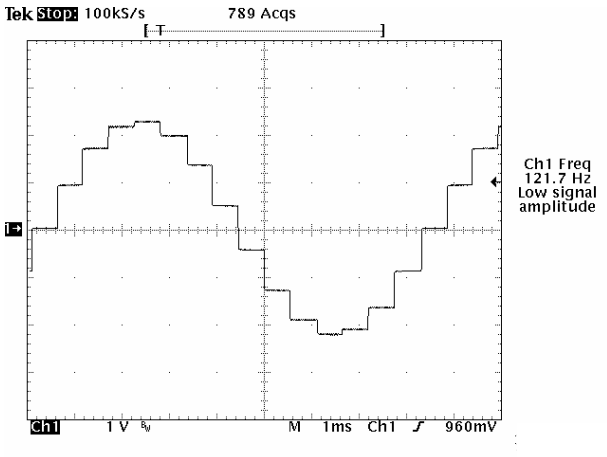
Neste trabalho optou-se por utilizar uma tabela de n=360 posições, discretizando a senóide de 1 em 1 grau. Como a restrição maior é em tempo de processamento e não em memória ROM, optou-se por discretizar um ciclo completo, evitando-se assim processamento adicional para determinação do quadrante e sinal da amostra. A figura 3.20 mostra o resultado da implementação do algoritmo LUT para três freqüências, com delta inteiro e positivo. Para freqüências baixas, figura 3.20(a), com Δ=1, as 360 posições da tabela são lidas.

Para a freqüência nominal, 60 Hz, figura 3.20(b), com Δ=12, a tabela é lida de 12 em 12 posições. Finalmente, para freqüências altas, 120 Hz, figura 3.20(c), com Δ=24, a tabela é lida de 24 em 24 posições. Claramente pode ser observado que quanto maior for Δ, maior será a distorção harmônica imposta à reconstituição. Além do erro de amostragem (influência do Δ), um outro fator que influencia a distorção harmônica é o erro de quantização. Ele é devido ao comprimento finito da palavra digital utilizada para armazenar os valores da tabela, por exemplo, 8 bits, 16 bits, etc.



(a) f≈12 Hz, Δ=1 (360 amostras) (b) f≈ 60 Hz, Δ=12 (30 amostras)

Figura 3.20(a) e (b) – Influência de delta na sintetização da senóide



(c) f≈120 Hz, Δ=24 (15 amostras)

Figura 3.20(c) – Influência de delta na sintetização da senóide

3.5.3 Modulação regular simétrica

Dentre as técnicas de modulação por largura de pulso senoidais, duas delas tem sido largamente utilizadas em aplicações digitais em tempo real, que são a Modulação Regular Simétrica e a Modulação Regular Assimétrica [25].

Estas técnicas são orientadas à implementação em *software*, sintetizando os sinais de MLP através de um método híbrido de busca direta à tabela e processamento matemático [26], diferindo da amostragem natural, onde um sinal senoidal de referência é comparado analogicamente com uma portadora triangular. Na implementação digital a portadora triangular é substituída pelo período de amostragem (T_{sw}).

A modulação regular simétrica, ou uniforme, necessita de apenas uma amostra da senóide para cada ciclo da portadora e recebe este nome porque tais amostras são uniformemente distribuídas ao longo do sinal a ser sintetizado, ver figura 3.21.

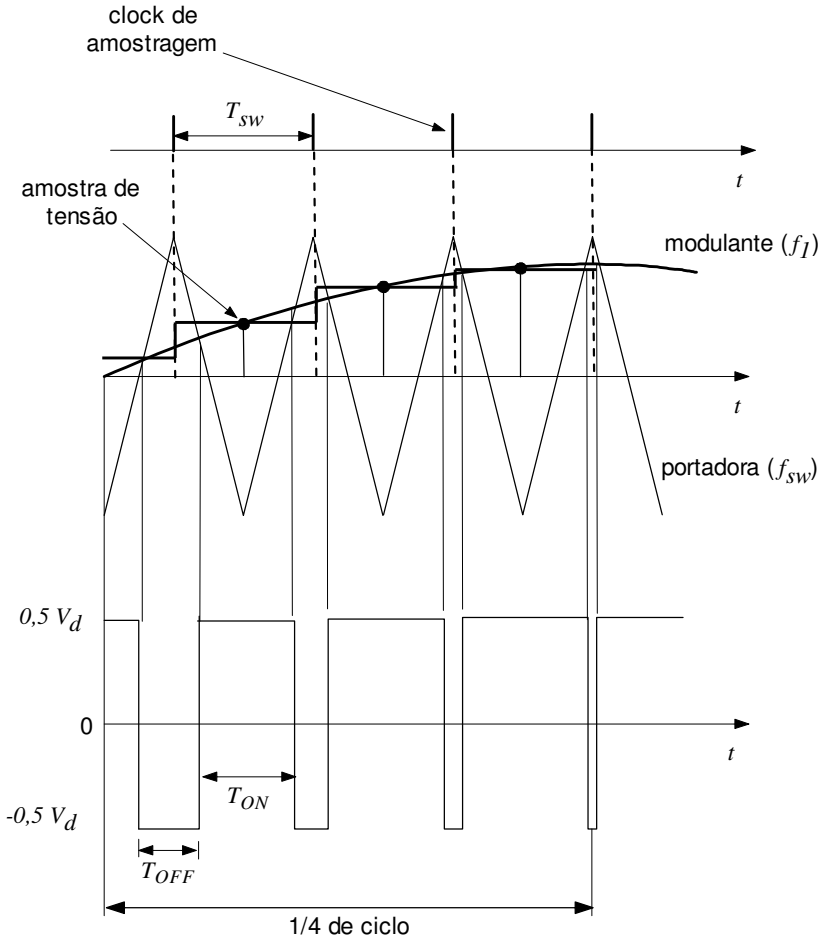


Figura 3.21 – Princípio de modulação por amostragem regular simétrica

Já a modulação regular assimétrica necessita de duas amostras da senóide para cada ciclo da portadora e recebe este nome devido ao fato de que a "comparação" do sinal modulante com a portadora ocorre em diferentes níveis do sinal amostrado, figura 3.22. Estes dois métodos de modulação podem ser implementados pelo método dos quatro temporizadores (*timers*), um *timer* para temporizar cada fase de saída e o quarto para o período de chaveamento T_{sw}.

O método simétrico produz padrões de chaveamento MLP com menor conteúdo harmônico do que o assimétrico, bem como permite que a malha de controle opere em uma taxa duas vezes superior à freqüência de chaveamento do método simétrico. Naturalmente, utilizar um método com duas amostras, ou duas atualizações por ciclo requer maior capacidade computacional [27]. A escolha da técnica apropriada é um compromisso entre capacidade computacional do dispositivo gerador do MLP e do desempenho aceitável para a aplicação.

A geração de sinais modulados em largura de pulso via *software,* em tempo real, com modulação regular simétrica ou assimétrica tem sido utilizada desde 1975. Foi apresentada em 1981 para microprocessadores de 8 bits, em 1983 para microprocessadores de 16 bits e tem sido utilizada tanto em desenvolvimentos acadêmicos quanto em aplicações industriais [25]. No entanto, a literatura tem citado inúmeras melhorias para a técnica básica, buscando melhor desempenho, redução de conteúdo harmônico, redução dos recursos computacionais necessários à implementação do modulador e principalmente redução de custo. Outros métodos de implementação tem sido propostos pela literatura, tais como, método dos três *timers*, método de um *timer*, porém, o método dos quatro *timers* tem-se mostrado mais eficiente quando comparado a estes últimos [28], pois utiliza um menor número de interrupções e operações aritméticas, entretanto necessita de mais *hardware* do processador (*timers*).

Os métodos aqui citados suportam estratégias de minimização ou eliminação seletiva de harmônicas, porém às custas de maior capacidade computacional do processador e tempo de processamento, o que não foi explorado neste trabalho.

Por motivos da baixa capacidade computacional do processador escolhido para a implementação do protótipo este trabalho considera o método da

amostragem regular simétrica, sem aplicar nenhuma técnica de minimização ou eliminação seletiva de harmônicas, as quais ficam pendentes para uma eventual continuidade deste trabalho.

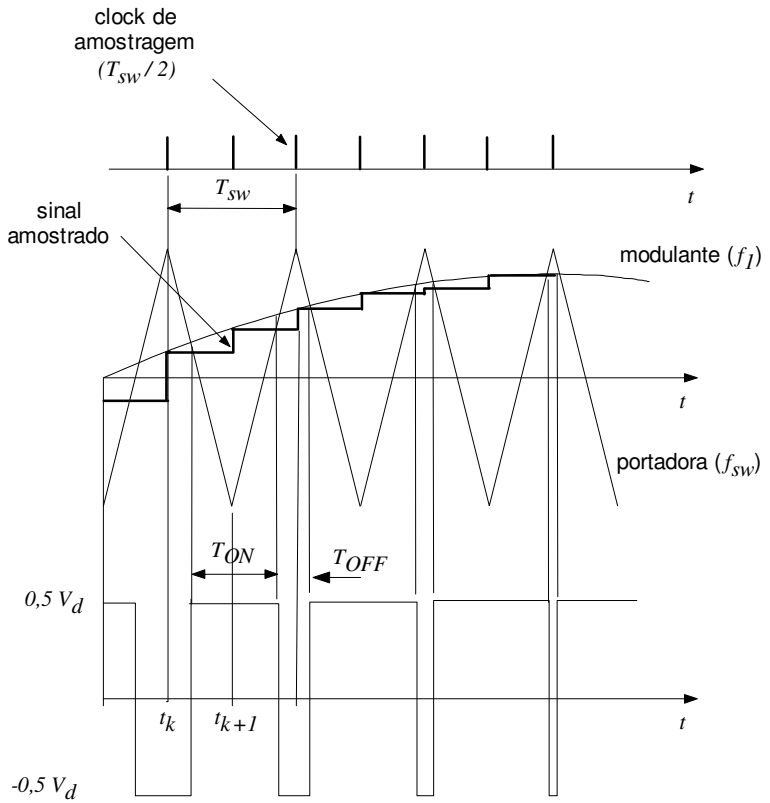


Figura 3.22 – Princípio de modulação por amostragem regular assimétrica

Tal como indicado na equação (3.17), a referência [19] propõe um método de cálculo para a largura do pulso ($\delta_k = T_{ON}$), o qual varia em passos definido por (3.26), para todo k inteiro e positivo, sendo t_k o *k-ésimo* instante de amostragem da portadora.

$$t_k = T_k + \frac{T_{sw}}{2} = k \cdot T_{sw} + \frac{T_{sw}}{2} \tag{3.26}$$

O tempo de chave desligada (T_{OFF}) pode ser determinado a partir da expressão (3.2), o qual é uniformemente distribuído antes e depois de T_{ON} como pode ser observado na figura 3.23.

A partir das expressões (3.2) e (3.17) podem ser derivados os tempos de chave ligada e desligada para as três fases, defasando-as de 120° elétricos.

Esta técnica mostra que no processo de amostragem regular as posições dos pulsos são regularmente espaçadas e a largura destes pulsos são precisamente definidas, tal que é possível escrever expressões trigonométricas simples que definem tais larguras. Utilizando estas expressões é possível gerar, via *software* e em tempo-real, os sinais de MLP com amostragem regular simétrica.

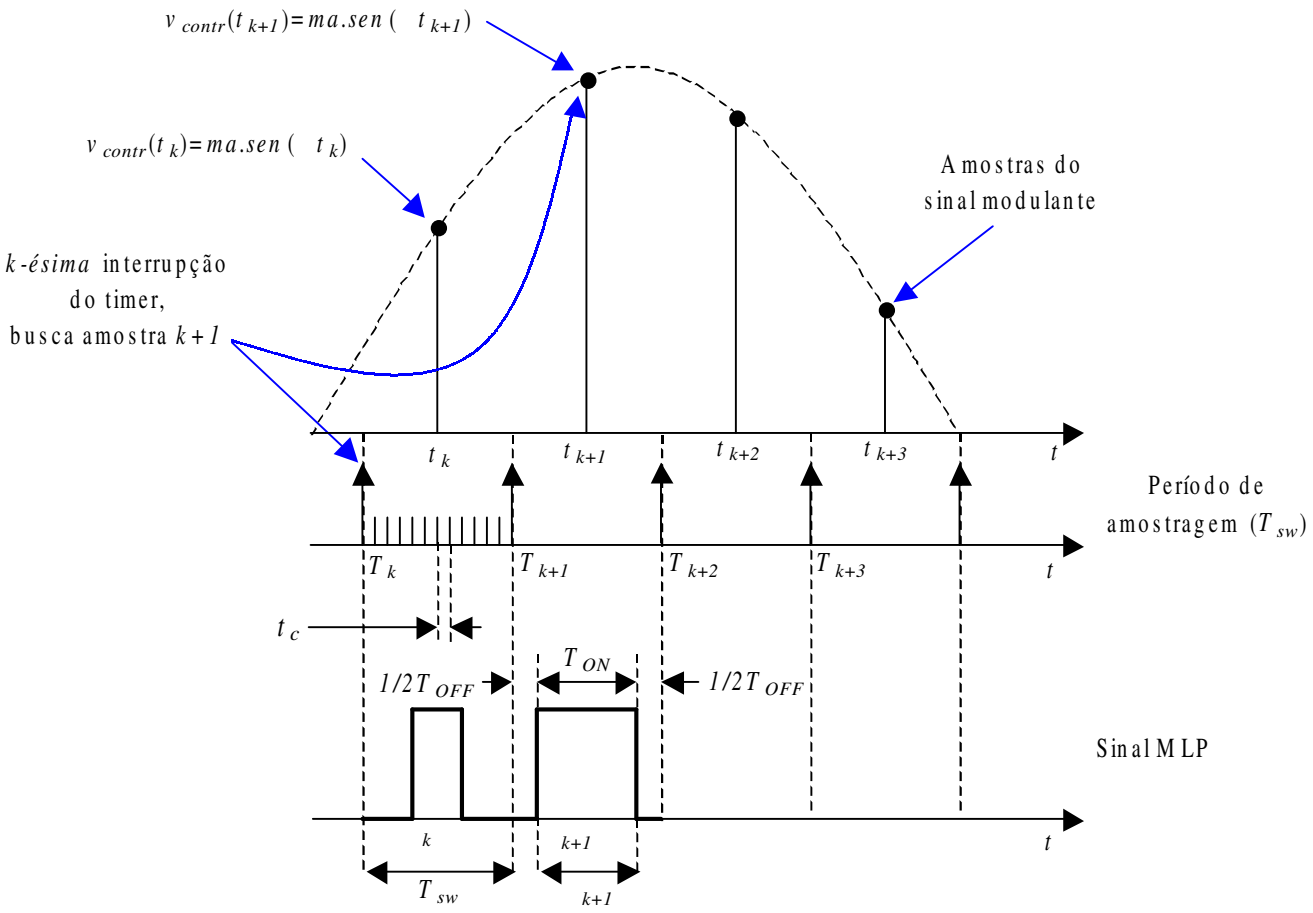


Figura 3.23 – Detalhe do *k-ésimo* e *k-ésimo+1* pulsos da MLP para modulação regular simétrica

A figura 3.23 representa dois pulsos da MLP, sendo que na *k-ésima* interrupção do temporizador que gera a base de tempo do período de amostragem (ou seja no instante, $t = T_k = k.T_{sw}$), é feita a amostra do sinal modulante $v_{contr}(t_{k+1})$, isso indica que o cálculo da MLP é feito um período de T_{sw} em avanço [26]. Se a temporização de T_{ON} e T_{OFF} fosse executada no mesmo período no qual ela é calculada não haveria tempo suficiente para realizar as duas tarefas (cálculo +

temporização). Como será visto no Capítulo 5, a tarefa do *software* que realiza o cálculo da MLP consume uma boa parcela do período do T_{sw}. Isso pode ser visualizado na figura 5.1, que mostra o encadeamento das tarefas do *software* dentro de um período do T_{sw}. Nesta figura, a tarefa indicada por "PWM(k)" é a tarefa que realiza a temporização de T_{ON} e T_{OFF} calculados no ciclo anterior. Por outro lado, se a temporização fosse feita no mesmo período no qual foi calculada, ela só poderia ser executada após a execução da tarefa "Duty-Cycle(k)", que é a tarefa que efetivamente calcula T_{ON} e T_{OFF}. Desta forma, durante o período $k.T_{sw}$ são feitos os cálculos de T_{ON} e T_{OFF} para o próximo período, eles são armazenados na memória RAM do processador e executados em $(k+1).T_{sw}$.

O início da temporização do *k-ésimo* pulso (α_k) é definido por (3.27). Neste instante será vetorizada uma interrupção no MCU e será atualizado o conteúdo do registrador do temporizador, que irá controlar a largura do pulso δ_k, representado pelas palavras digitais $W_{a,}$, W_b e W_c (3.29) a (3.31). Neste instante também, o sinal de controle da base, ou porta, dos transistores da ponte inversora devem mudar de estado. Vencido este tempo estes sinais deverão ser complementados (níveis lógicos invertidos). Deve ser observado que $W_{a,}$, W_b e W_c temporizam apenas os T_{ON}. Os tempos das chaves desligadas T_{OFF} são obtidos algebricamente por (3.2) e digitalmente são determinados por $W_{da,}$, W_{db} e W_{dc} (3.32) a (3.34).

$$\alpha_k = t_k - \frac{\delta_k}{2} \quad \text{para } k = 1, 2, 3... \tag{3.27}$$

Em uma MLP digital a expressão (3.17) precisa ser manipulada para se determinar o valor das palavras digitais que representarão o número de contagem dos temporizadores de cada uma das fases de saída do inversor.

Como mostrado na seção 3.4, a tensão de fase na saída do inversor pode ser zero ou V_d. Desta forma, em uma MLP senoidal, o valor médio nulo da senóide estará deslocado de 0,5 p.u. e será obtido com um ciclo de trabalho (D) igual à 50%, figura 3.24. Então, a expressão (3.17) pode ser reescrita conforme indicado por (3.28) [23].

$$\delta_k = T_{sw} \cdot \left(\frac{1}{2} \cdot m_a \cdot sen(\omega t_k) + \frac{1}{2} \right) \tag{3.28}$$

Se m_a=0 (senóide com amplitude zero), δ_k = ½.T_{sw}, o que significa D=50%, como indicado acima. Se m_a=1, δ_k dependerá do valor do seno. Se $sen(\omega t)$=1, δ_k = T_{sw} e D=100%, representando o valor máximo da senóide. Já se $sen(\omega t)$=-1, δ_k = 0 e D=0%, representando o valor mínimo da senóide, tal como mostrado na figura 3.24.

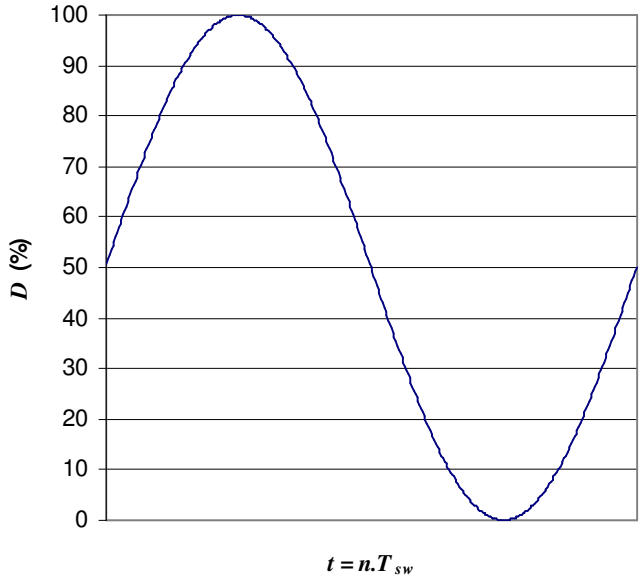


Figura 3.24 – Variação do ciclo de trabalho na geração dos pulsos da MLP para modulação senoidal

A expressão (3.28) precisa ainda ser reescrita para colocá-la na forma digital, de acordo com [26], as palavras digitais que representam as larguras os pulsos "W" (número de contagem do temporizador de δ_k, que pode variar de t_c=0 a t_c=256, ver figura 3.23) podem ser calculadas por:

$$W_a = 0{,}5 \cdot N \cdot \left(\frac{V_{SM}}{V_d} \cdot \Theta_A + 1 \right) \tag{3.29}$$

$$W_b = 0{,}5 \cdot N \cdot \left(\frac{V_{SM}}{V_d} \cdot \Theta_B + 1 \right) \tag{3.30}$$

$$W_c = 0{,}5 \cdot N \cdot \left(\frac{V_{SM}}{V_d} \cdot \Theta_C + 1 \right) \tag{3.31}$$

Os tempos das chaves desligadas "W_d" são dados por:

$$W_{da} = 0{,}5 \cdot (N - W_a) \tag{3.32}$$

$$W_{db} = 0{,}5 \cdot (N - W_b) \tag{3.33}$$

$$W_{dc} = 0{,}5 \cdot (N - W_c) \tag{3.34}$$

N é uma constante igual a 256 para processadores de 8 bits; V_{SM} é a palavra digital que representa a referência de tensão do estator (V^*_s) e Θ_A, Θ_B e Θ_C são as amostras do seno para as fases A, B e C respectivamente, previamente calculadas e armazenadas na memória do MCU. A figura 3.25 representa as larguras dos pulsos da MLP para as três fases de saída.

Para as expressões (3.29) a (3.34) pode-se fazer o mesmo teste de consistência que foi feito para a expressão (3.28), isto é, se V_{SM}=0, $W_{a,b,c}$ =0,5.N, ou seja, $W_{a,b,c}$=128 contagens, o que equivale a D=50% e $W_{da,db,dc}$= $0{,}5.N$, o que equivale aos outros 50% do ciclo com a chave desligada. Se V_{SM}=1, $W_{a,b,c}$ dependerá dos valores de Θ_A, Θ_B e Θ_C. Se Θ=1, $W_{a,b,c}$ =N, ou $W_{a,b,c}$ =256 contagens, o que equivale ao valor máximo da senóide, ou D=100% e se Θ=-1, $W_{a,b,c}$ =0, ou $W_{da,db,dc}$=256 contagens, o que equivale ao valor mínimo da senóide, ou D=0%. Deve ser observado que 256 contagens do temporizador de 8 bits equivalem a um período de chaveamento (T_{sw}) da MLP. Na figura 3.23 cada contagem deste temporizador é representada por t_c.

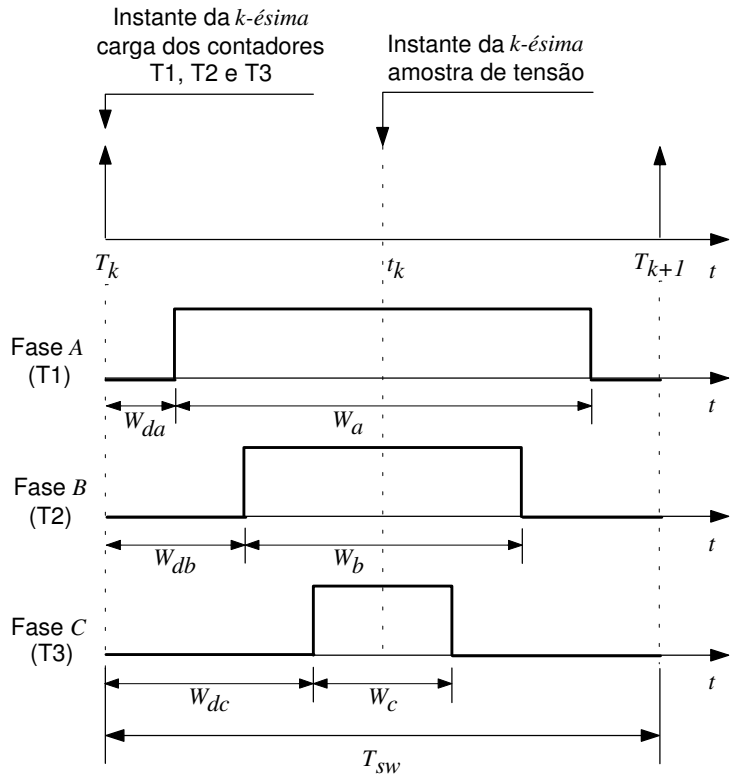


Figura 3.25 – Sintetização dos pulsos MLP para as 3 fases

A freqüência do sinal a ser sintetizado na saída do inversor é dada pela expressão (3.21). Nesta expressão, delta (Δ) é a variável de incremento (*offset*) do ponteiro de leitura da tabela do seno, que neste trabalho foram escolhidos os seguintes valores: Δ = 1, 2, 3, 4, 6, 9, 12, 18, 24; pois de acordo com o exposto na seção 3.4, deve-se sempre que possível procurar manter o índice de modulação em freqüência (m_f) inteiro e múltiplo de 3, para reduzir o conteúdo harmônico na saída do inversor. Para baixas freqüências, um ciclo completo da senóide possuirá 360 amostras (leitura da tabela de 1 em 1 grau elétrico) e em altas, 15 amostras (leitura de 24 em 24 graus). Obviamente esse artifício acarreta distorções harmônicas na sintetização da senóide, figura 3.20, no entanto ele é necessário para se compensar a baixa velocidade de processamento do microcontrolador utilizado na implementação do protótipo experimental.

A tabela 3.2 permite que se redesenhe a figura 3.15, agora representada pela figura 3.26, observando o decréscimo de m_f com o aumento da freqüência de saída f_s. A mudança do índice de modulação em freqüência ocorre em faixas de 15 Hz, com uma histerese de $\pm$1 Hz em torno do ponto de mudança, faz-se isso para evitar instabilidade numérica quando do processamento dos extremos das faixas. Entretanto manter a faixa de mudança de m_f fixa em 15 Hz traz um inconveniente acréscimo da freqüência de chaveamento f_{sw}, como pode ser observado na figura 3.26 para as baixas freqüências de saída. Este acréscimo de f_{sw} nem sempre é interessante, pois pode acarretar sobre carga de processamento. Desta forma pode-se dizer que as freqüências de chaveamento não estão otimizadas.

Tabela 3.2 – m_f em função de f_s e f_{sw} não otimizada

Δ	m_f	Início da faixa de Δ		Fim da faixa de Δ	
		f_s (Hz)	f_{sw} (Hz)	f_s (Hz)	f_{sw} (Hz)
1	360	3	1080	16	5760
2	180	14	2520	29	5220
3	120	27	3240	42	5040
4	90	40	3600	55	4950
6	60	53	3180	68	4080
9	40	66	2640	81	3240
12	30	79	2370	94	2820
18	21	92	1840	107	2140
24	15	105	1575	120	1800

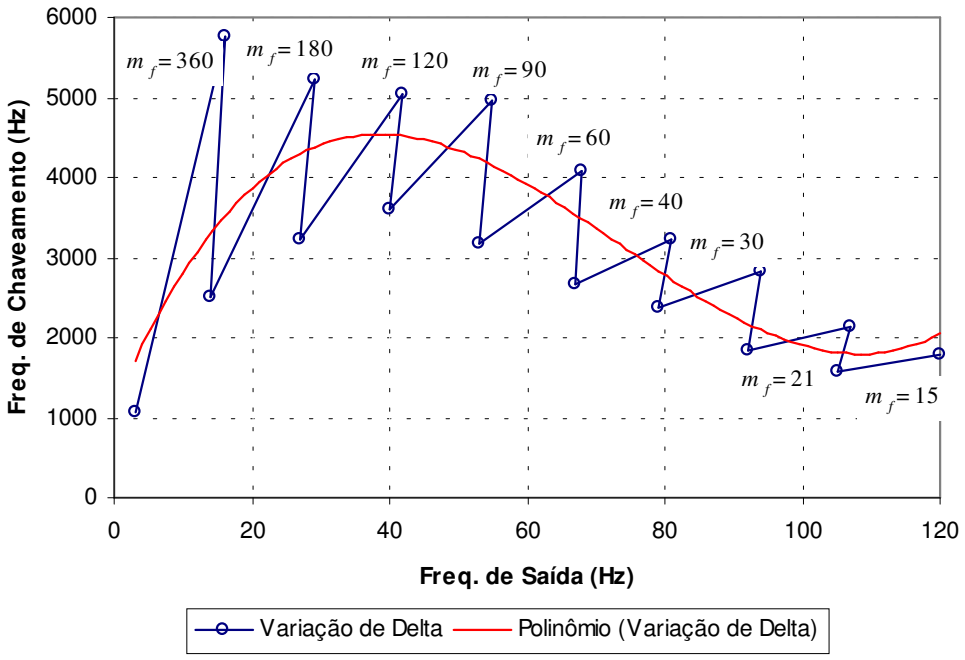


Figura 3.26 – Freqüências de saída e de chaveamento não otimizada

A tabela 3.3 mostra uma possível redefinição (ou otimização) das faixas de mudança do índice de modulação m_f, de forma a manter a variação da freqüência de chaveamento dentro de uma faixa aceitável para a aplicação. Na figura 3.27, o polinômio tende a uma reta tornando esta figura parecida com a 3.15. Quanto menor for a faixa de variação da freqüência de chaveamento, melhor será a implementação digital. Deve ser observado que é muito comum implementações de MLP com f_{sw} fixa, porém neste caso deve-se permitir que m_f assuma valores não inteiros para sintetizar todas as freqüências dentro da faixa de interesse.

Tabela 3.3 – m_f em função de f_s e f_{sw} otimizada

Δ	m_f	Início da faixa de Δ		Fim da faixa de Δ	
		f_s (Hz)	f_{sw} (Hz)	f_s (Hz)	f_{sw} (Hz)
1	360	3	1080	7	2520
2	180	6	1080	14	2520
3	120	13	1560	20	2400
4	90	19	1710	26	2340
6	60	25	1500	40	2400
9	40	39	1560	60	2400
12	30	59	1770	80	2400
18	21	79	1659	114	2394
24	15	113	1695	120	1800

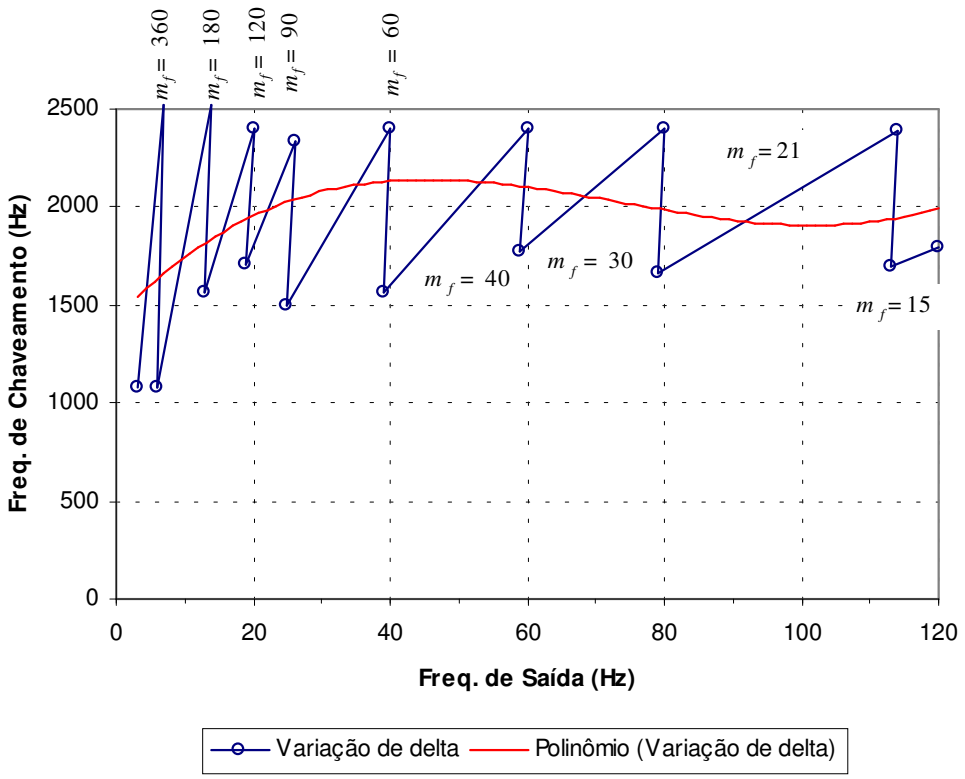


Figura 3.27 – Freqüências de saída e de chaveamento otimizada

3.6 Fluxogramas

O fluxograma da figura 3.28 implementa a rotina principal (MAIN) do modulador MLP trifásico, a qual escalona (chama) as tarefas de tratamento (sintetização) de cada uma das fases de saída do inversor, cujos fluxogramas estão representados na figura 3.29, podendo-se observar que para inverter o sentido de rotação do motor basta mudar a leitura de duas fases.

Sempre que a interrupção do contador do período de amostragem (T_{sw}), vinculada à interrupção do temporizador (T0) for vetorizada, três contadores (T1, T2 e T3) devem ser carregados com os tempos das chaves desligadas W_{da}, W_{db} e W_{dc}, de cada uma das fases de saída, como mostrado na figura 3.25, calculados pelas expressões (3.32), (3.33) e (3.34) respectivamente. No final desta contagem cada um destes temporizadores devem ser carregados com os seus respectivos tempos de pulso W_a, W_b e W_c, calculados pelas expressões (3.29), (3.30) e (3.31) respectivamente. Esta técnica é conhecida como Método dos Quatro Temporizadores (*timers*) [29] e [30].

A figura 3.30 mostra as rotinas de interrupção do processador para temporizar T_{sw} (INT T0) e o escalomento (chamada) de cada uma das tarefas de processamento das fases de saída (INT Tx, onde x=1, 2 e 3), para gerar os sinais MLP das fases “a”, “b” e “c”.

As figuras 3.28 a 3.30 representam as tarefas periódicas que compõem o *software* do modulador por largura de pulso, entretanto existem outras tarefas, chamadas aperiódicas, cujo escalonamento não pode ser previsto de uma forma determinística. Geralmente estas tarefas são de detecção e tratamento de falhas, ou vinculadas a interrupções de *hardware*. No protótipo experimental foram implementadas rotinas de detecção e proteção de sobrecorrente (SC) e sobretensão (ST), escalonadas por interrupção externa de *hardware* (EXT INT), figura 3.31, cuja função é proteger o circuito de potência contra eventuais falhas elétricas.

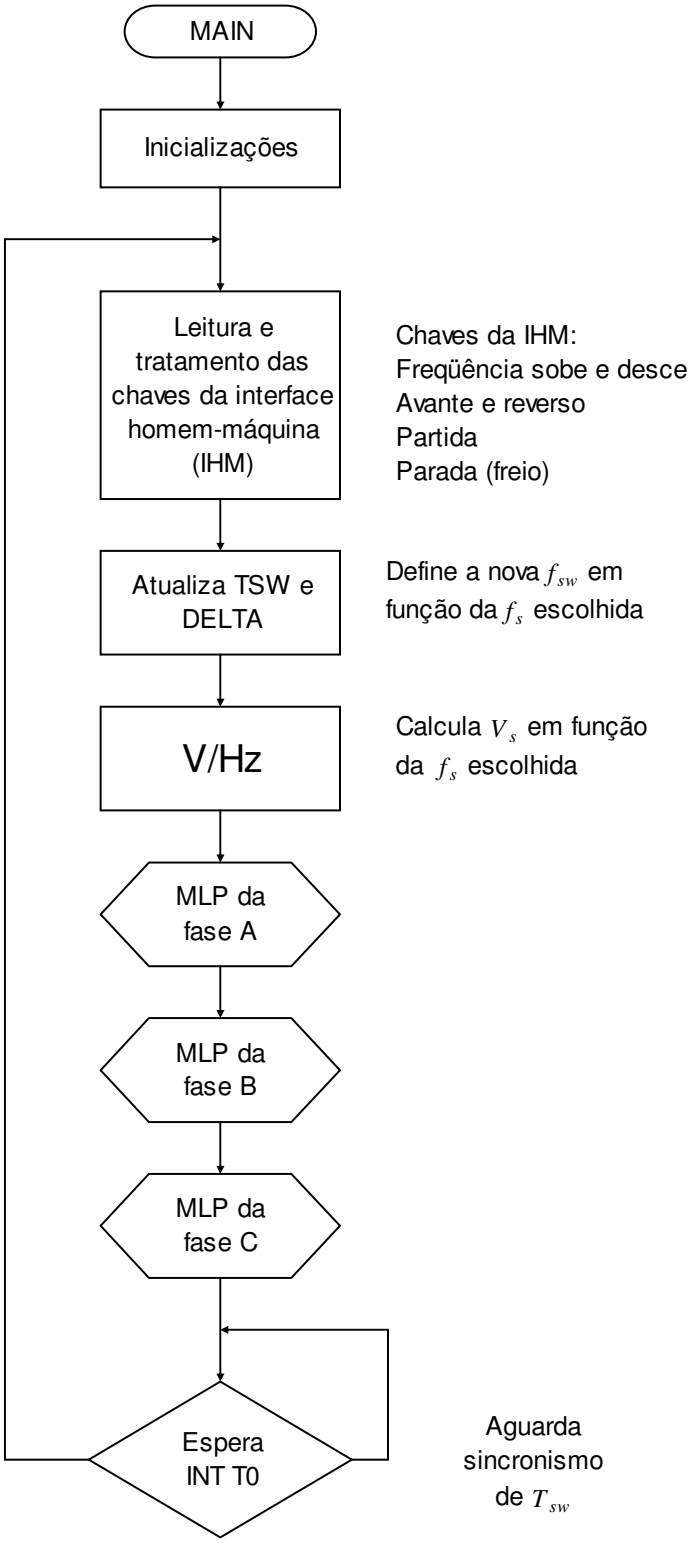


Figura 3.28 – Fluxograma do modulador MLP.
Tarefa principal (MAIN)

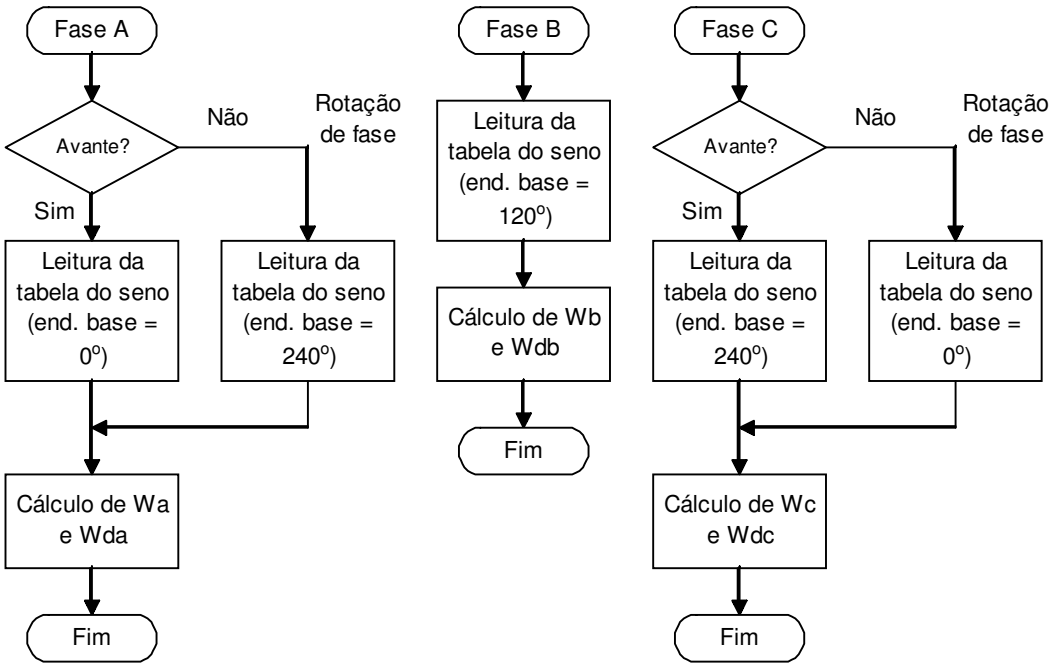


Figura 3.29 – Fluxogramas de tratamento das fases (cálculo da MLP)

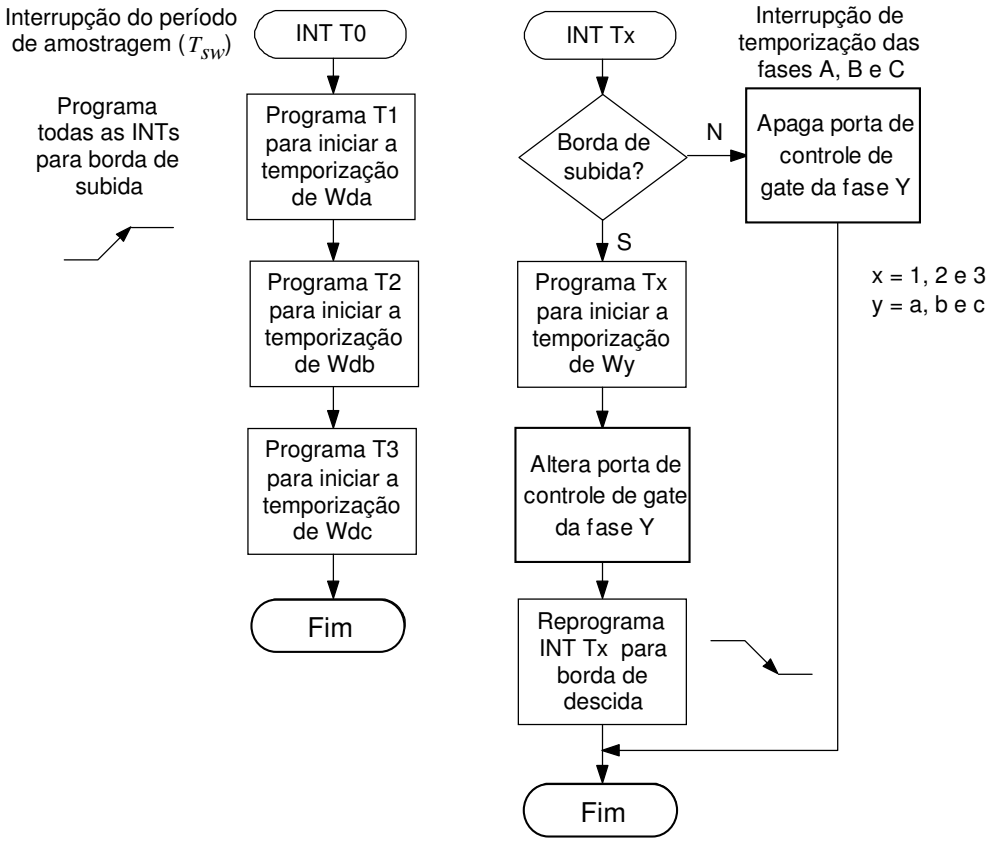


Figura 3.30 – Interrupções dos temporizadores para gerar o MLP nos pinos de saída do microcontrolador

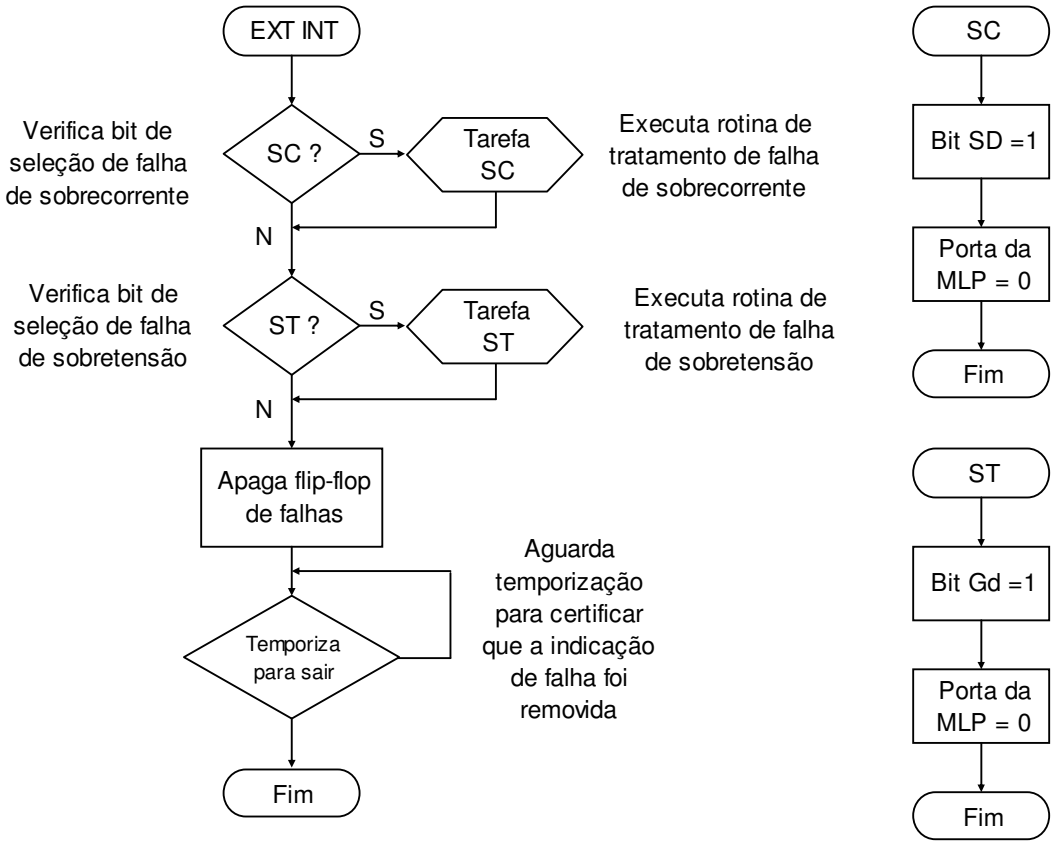


Figura 3.31 – Tarefas aperiódicas de tratamento de falhas, escalonadas via interrupção

3.7 Tempo morto

Um dos principais problemas encontrados em inversores de tensão baseados em moduladores por largura de pulsos, que operam em malha aberta, diz respeito à não linearidade do seu ganho de tensão, causado pelas características não ideais do estágio de potência do inversor. A não linearidade mais significativa não é inerente ao processo de modulação mas a que deve ser intencionalmente introduzida para se evitar a condução direta entre os transistores de uma mesma perna da ponte inversora [31], [32] e [33], o que colocaria em curto-circuito o barramento de corrente contínua, obviamente, com conseqüências catastróficas para o estágio de potência. Esta não linearidade é introduzida como um “tempo morto”, que é um pequeno atraso no sinal de controle da porta do transistor que está sendo ligado, fazendo que os dois transistores de uma perna da ponte estejam desligados ao mesmo tempo. Desta forma, garante-se que as

chaves de uma mesma perna nunca conduzirão simultaneamente. Este atraso introduz erros de amplitude e de fase na tensão de saída, provocando distorções e quedas de tensão na componente fundamental. Embora individualmente pequenos, quando acumulados em um ciclo completo o erro de tensão pode ser suficiente para distorcer o sinal MLP, provocando distorções na corrente do motor, podendo acarretar pulsações de torque. Esta distorção aumenta com o aumento da freqüência de chaveamento [31], introduzindo componentes harmônicas que, se não compensadas, podem causar instabilidades e perdas adicionais no motor.

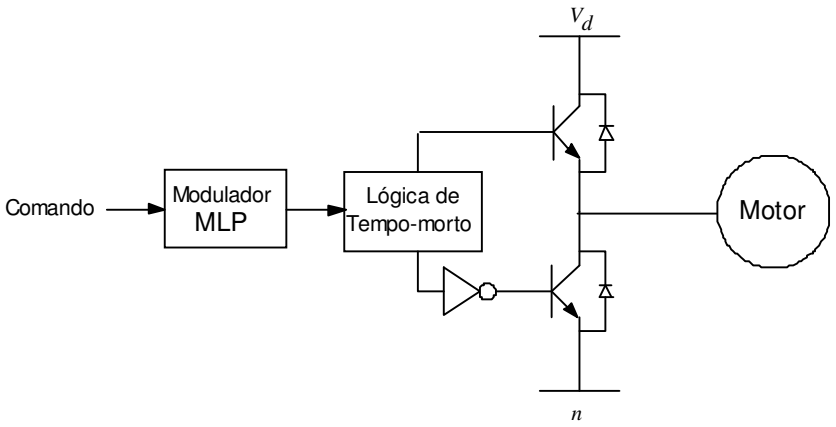


Figura 3.32 – Inclusão da lógica de tempo morto para uma perna da ponte

A figura 3.33 ilustra as formas de onda de saída do modulador com 50% de ciclo de trabalho das chaves, sendo que figura 3.33(a) representa a saída ideal, sem tempo morto e na 3.33(b), os sinais alterados pela inclusão deste (área sombreada).

Pode se observar que nem o transistor superior nem o inferior permanecem ligados durante 50% do ciclo de trabalho. A rigor a inclusão do tempo morto pode ser feita por *hardware* ou *software*, porém as implementações baseadas em *hardware* são mais complexas do ponto de vista da compensação do tempo morto e não da inclusão do tempo morto em si. Por outro lado, a literatura tem freqüentemente tratado das soluções baseadas em *software* [31] a [35].

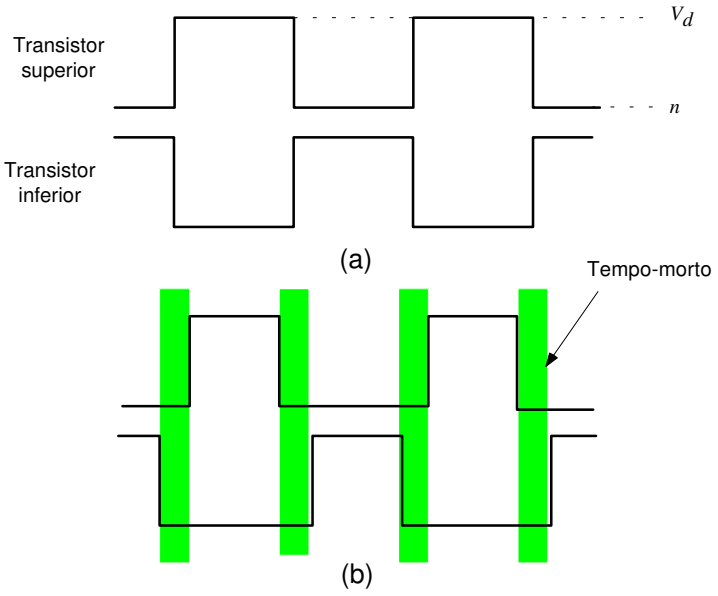


Figura 3.33 – Inclusão do tempo morto

3.7.1 Geração do tempo morto baseado em *hardware*

O circuito da figura 3.34 sugere como pode ser introduzido um tempo morto entre os sinais de controle de uma mesma perna da ponte [36] e [37], bem como a geração do sinal complementar. Para uma ponte trifásica devem ser utilizados três circuitos idênticos a este. Neste circuito emprega um inversor *Schmitt-Trigger* do tipo 74HC14. Utilizando-se diodos, diferentes atrasos podem ser gerados para a subida e descida do pulso apenas variando a constante de tempo RC.

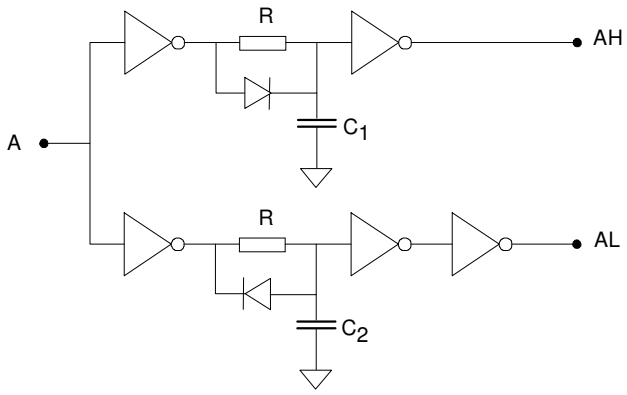


Figura 3.34 – Geração de tempo morto por *hardware*

De acordo com [37], o tempo morto (t_d) para este circuito pode ser determinado por:

$$t_d = -RC\ln(V_{th}/5) \quad [\text{s}] \tag{3.35}$$

Sendo V_{th} a tensão de limiar (*threshold*) da porta inversora tipo *Schimitt-Trigger*. As formas de onda da figura 3.35 representam a geração das tensões para os sinais de controle (AH e AL), para a perna da fase A.

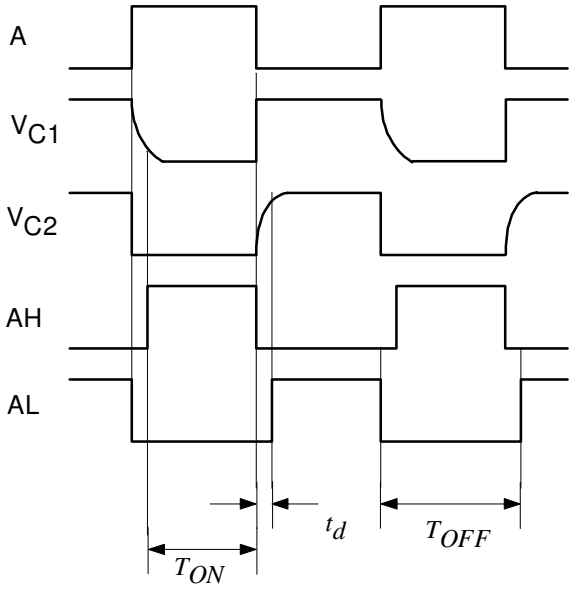


Figura 3.35 – Formas de onda do tempo morto (t_d) baseado em *hardware*

3.7.2 Geração do tempo morto baseado em *software*

Baseado na discussão anterior, a equação (3.17) pode ser re-escrita incluindo-se o tempo morto [33].

$$\delta_k = T_{ON} = T_{sw} \cdot m_a \cdot \text{sen}\,\omega_1 \cdot t_k - \text{sgn}(i_a) \cdot t_d \tag{3.36}$$

Sendo $sgn(i_a)$ a polaridade da corrente da fase "a", ou seja:

$$\text{sgn}(i_a) = \begin{cases} 1 : \text{quando } i_a > 0 \\ -1 : \text{quando } i_a < 0 \end{cases} \tag{3.37}$$

Combinando as expressões (3.9) e (3.36) pode-se determinar a tensão média por fase (V_a). O mesmo raciocínio deve ser feito para as demais fases.

$$V_a = \frac{V_d}{2} \cdot m_a \cdot \text{sen}\,\omega_1 \cdot t_k - \frac{\text{sgn}(i_a) \cdot t_d}{T_{sw}} \cdot \frac{V_d}{2} \tag{3.38}$$

A qual pode ser re-escrita por:

$$V_a = \frac{V_d}{2} \cdot m_a \cdot \operatorname{sen} \omega_1 \cdot t_k \pm \Delta V_e \tag{3.39}$$

Sendo ΔV_e o erro de tensão por período da portadora (T_{sw}), para mais ou para menos, dependendo da polaridade da corrente (i+=-ΔV_e e i-=+ΔV_e). Deve ser observado que o erro de tensão não depende da magnitude da corrente. Este método é adequado a aplicações de baixo custo, pois necessita apenas da polaridade da corrente de fase, eliminando conversores analógico para digital e sobrecarga de *software*. A polaridade pode ser determinada amostrando-se a corrente das três fases e transformando-as em sinais quadrados, por meio de comparadores de tensão.

Em aplicações de baixo custo é impraticável a monitoração em tempo real do erro introduzido pela distorção "volts-segundo" em cada ciclo sintetizado na saída do inversor, bem como a monitoração da distorção das correntes de cada fase do motor. Levando-se em consideração que a inclusão do tempo morto provoca menor instabilidade em acionamentos de motores fracionários [38] do que para motores integrais (maior que 1cv), bem como esta distorção é mais significativa para altas freqüências de chaveamento e altas freqüências de saída, optou-se por uma implementação sem compensação do tempo morto, tal como representada na figura 3.36.

Se os transistores da ponte inversora fossem perfeitos, para gerar as formas de onda do MLP bastaria calcular os tempos T_{ON} e T_{OFF}, ligar ou desligar o bit da porta do MCU, correspondente ao terminal de controle do transistor superior e complementar este nível lógico em uma outra porta para o controle do transistor inferior da perna considerada. Porém esse procedimento não é suficiente para garantir a operação segura do estágio de potência do inversor, de forma que deve ser empregada uma lógica de proteção contra condução cruzada, incluindo-se o tempo morto.

A figura 3.36 mostra a geração dos sinais de controle da porta que comandam os transistores da perna correspondente à fase "a", tanto para a geração do tempo de desligado (*notch*) 3.36(a), quanto para a geração de um pulso 3.36(b), normalmente somente a borda do sinal que está ligando o transistor

é atrasada, como pode ser observado nesta figura. O instante de referência para o início da temporização é o instante α_k, a partir do qual temporiza-se t_d para ligar a porta de controle que estava em nível lógico zero.

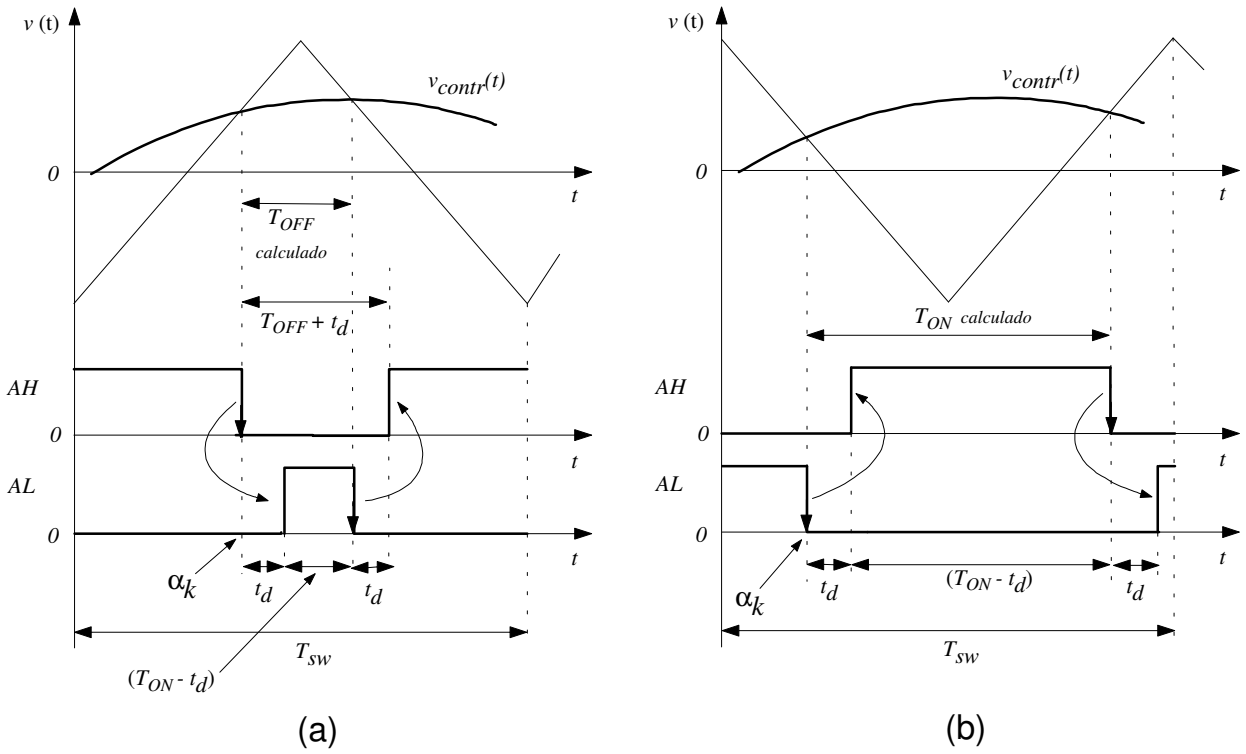


Figura 3.36 - Sinal de comando dos transistores com tempo morto, (a) *notch* e (b) pulso

A porta que estava no nível lógico um, é desligada no instante α_k sem nenhum atraso. Os tempos T_{ON} e T_{OFF}, previamente calculados, devem ser reduzidos de t_d. Após este tempo, um novo atraso t_d deve ser temporizado para garantir que a condução cruzada não ocorra.

Neste método deve-se fazer $t_d > t_f + t_r$, onde t_f e t_r são características de cada tipo de transistor. A figura 3.30 pode ser redesenhada incluindo-se o tempo morto na temporização do pulso de cada fase, figura 3.37.

De acordo com a referência [39] publicada em 1987, um tempo morto de 10 a 20 μs deveria ser introduzido entre os transistores de uma mesma perna para se evitar o curto na fonte de corrente contínua. Já para a referência [38] de 1997, para os transistores atuais (por exemplo, IGBT's operando até 20 kHz), ajustando-se t_d de 2 a 5 μs é suficiente para grande parte das aplicações. Pode-se notar que

após dez anos de desenvolvimento tecnológico das chaves semicondutoras o tempo morto pôde ser reduzido por um fator de quatro. Tipicamente, hoje se utiliza 1 μs. Se esta tendência persistir no futuro o tempo morto será bastante reduzido, ou seja, as chaves semicondutoras aproximam-se cada vez mais das características ideais apontadas no Apêndice B.

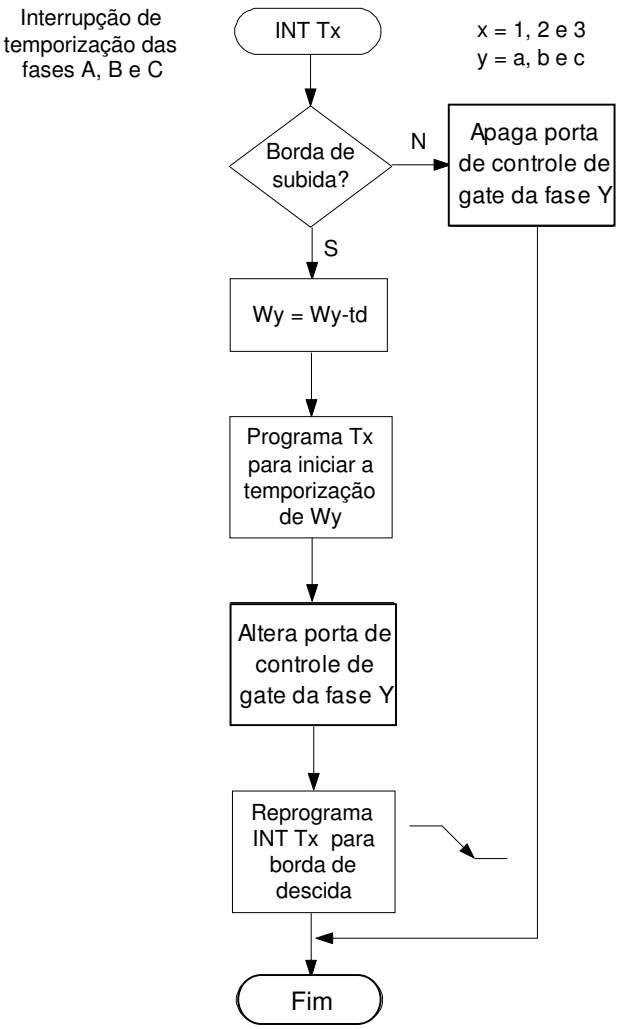


Figura 3.37 – Inclusão do tempo morto nas interrupções de T1, T2 e T3

3.8 Protótipo experimental

Para verificar a eficácia dos estimadores de desempenho que serão apresentados no Capítulo 4 foi desenvolvido um protótipo de laboratório com um *software* básico (*firmware*) que pudesse ser executado em um microcontrolador (MCU) desprovido de periféricos dedicados ao controle de motores elétricos e com

baixa velocidade de processamento. Essas limitações foram intencionalmente impostas para que se pudesse atingir o limite de escalonamento do processador e assim comparar os resultados experimentais e simulados. Desta forma os princípios teóricos tratados nas seções anteriores foram utilizados no Capítulo 4 com o *software* mostrado no Apêndice A e desenvolvido para o *hardware* aqui mostrado.

3.8.1 Sistema de desenvolvimento

O protótipo foi implementado em um MCU de 8 bits da Motorola, o MC68HC11A1, utilizando uma placa emuladora MC68HC11EVB a qual inclui diversas facilidades tais como: comunicação serial com PC, programa monitor que permite transferir os programas elaborados no PC para o HC11A1, funções de *debug* (*breakpoints, inspect*, etc.), possibilidade de conexão a uma placa de expansão genérica e outras. Foi desenvolvida uma placa de interface entre a placa emuladora e o estágio de potência, figura 3.38, a qual condiciona os sinais de controle e comanda o estágio de potência.

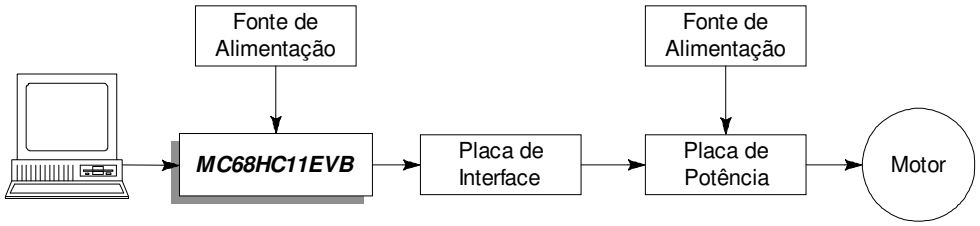


Figura 3.38 – Sistema de desenvolvimento

3.8.2 Placa emuladora

A família de microcontroladores 68HC11 da Motorola é suportada por uma ferramenta de desenvolvimento e *debugging*, MC68HC11EVB *Evaluation Board*, projetada para auxiliar no desenvolvimento de *softwares* para sistemas baseados nesta família de MCU's. De forma simplificada a figura 3.39 a representa. A referência [40] traz os detalhes necessários ao seu funcionamento.

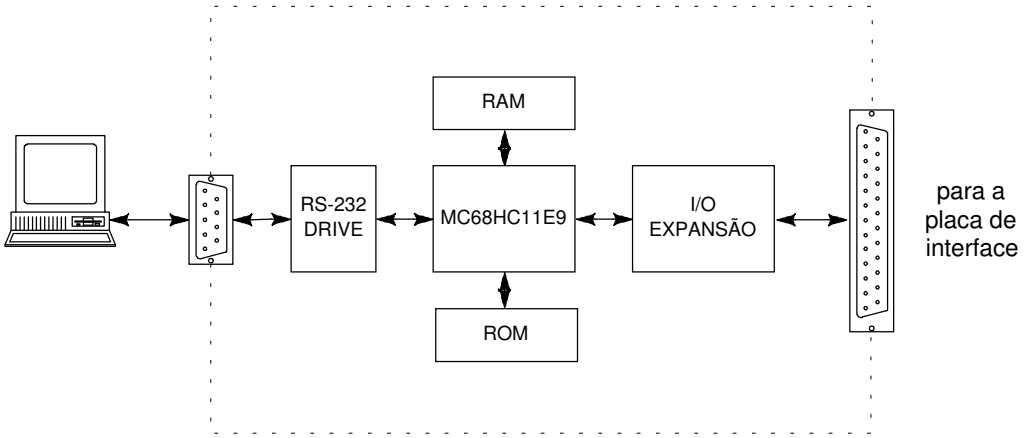


Figura 3.39 – Placa emuladora

3.8.3 Placa de interface

Como indicado na figura 3.38 esta placa faz a interface entre o emulador e a placa de potência e possui uma área de montagem de protótipos do tipo *wire-wrap*, onde foi montada uma interface homem-máquina (IHM) bastante simples, composta por chaves, as quais permitem variar a freqüência de saída para acima e para baixo da nominal, selecionar o sentido de rotação (avante e reverso, ou seja, para frente e para trás), bem como partir e parar o motor.

Como saída foi implementado um conversor digital para analógico (CDA) de 8 bits para se monitorar grandezas analógicas. Este circuito não é necessário na aplicação final (controle do motor).

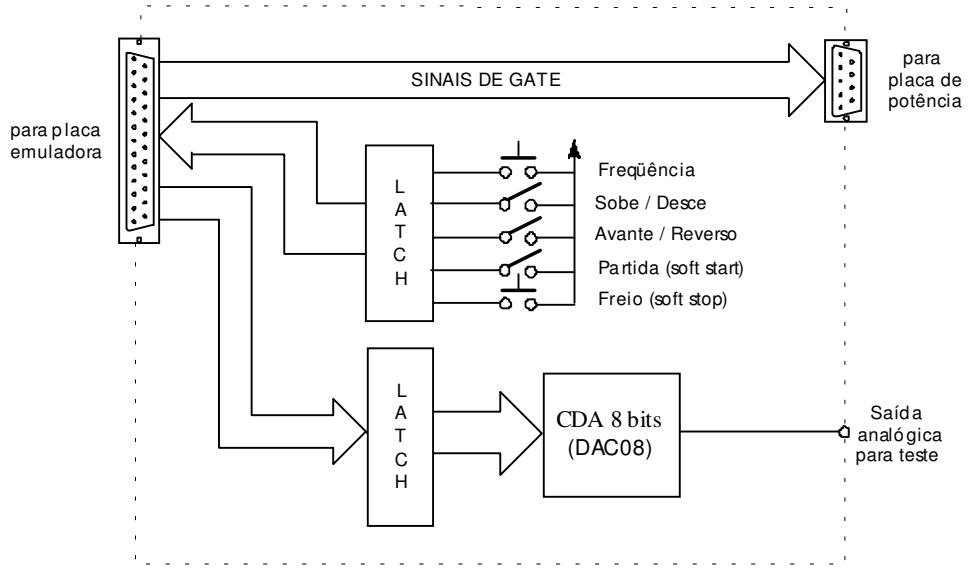


Figura 3.40 – Placa de interface

3.8.4 Placa de potência

A figura 3.41 mostra o diagrama de blocos da placa de potência a qual foi construída especialmente para esta aplicação e é responsável pela conversão dos sinais CA-CC-CA e pela alimentação do motor com freqüência e tensão variáveis.

Os sinais de controle de porta dos transistores da ponte inversora são recebidos da placa de interface, são pré-condicionados pelo circuito denominado *drive* de entrada e amplificados pelos *gate drives* que acionam os transistores.

Estes circuitos estão detalhados na seção seguinte.

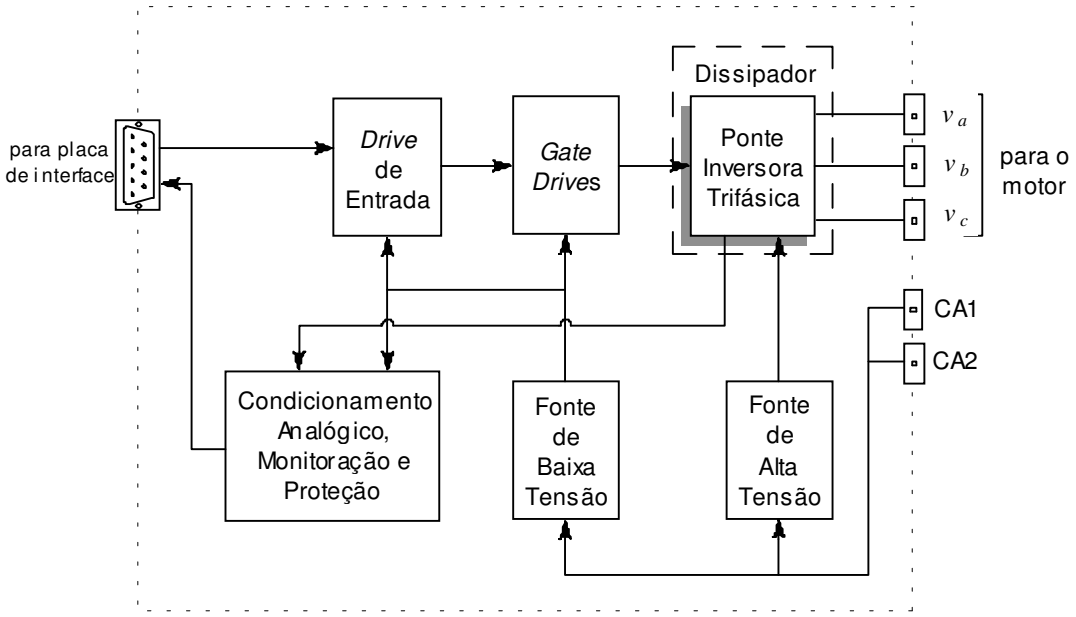


Figura 3.41 – Placa de potência

3.8.5 Detalhamento dos circuitos

Esta seção descreve os circuitos implementados no protótipo experimental e apresenta os dispositivos semicondutores, circuitos integrados e IGBT's, utilizados.

3.8.5.1 Microcontrolador

Como descrito, o protótipo experimental foi baseado no MCU MC68HC11A1 da Motorola, o qual possui as seguintes características [40]:

- 8k de memória interna ROM (versão A8 e B8);
- 512 bytes de memória interna EEPROM;
- 256 bytes de memória interna RAM;

- *Timer* de 16 bits de quatro estágios;
- Interfaces de comunicação serial síncrona e assíncrona;
- Oito canais de conversores analógico para digital (CAD) de 8 bits;
- Cinco portas de entradas e saídas digitais (I/O) de 8 bits.

O diagrama de blocos da figura 3.42 mostra a arquitetura interna deste MCU. Um fator limitante importante que deve ser mencionado é a sua baixa freqüência de operação de barramento (*bus clock*) de apenas 2,0 MHz, o que implica em um ciclo de máquina de 500 ns.

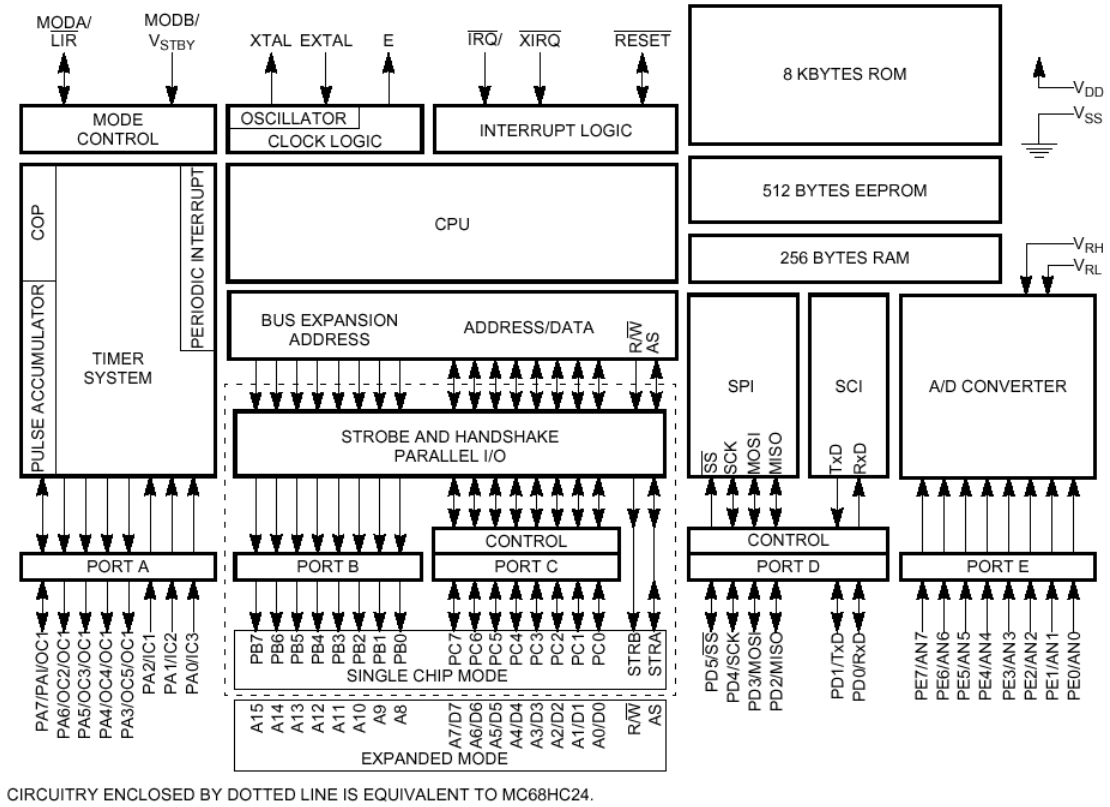


Figura 3.42 – Diagrama de blocos do MC68HC11A1

Tipicamente os MCU's da Motorola possuem um divisor por quatro no circuito gerador de *clock*, de tal forma que para se obter o *clock* interno de 2 MHz foi utilizado um cristal externo de 8 MHz. Esta característica do processador forçou a otimização do código em tempo de processamento e não na melhor solução em termos de utilização de memória.

3.8.5.2 Fonte de alimentação

Para a operação deste tipo de conversor são necessárias ao menos duas fontes de alimentação de corrente contínua (CC), uma com capacidade de fornecer potência para o motor (fonte de alta tensão, não regulada) e outra com capacidade de fornecer tensão regulada para os circuitos de controle e disparo dos transistores da ponte inversora (fonte de baixa tensão). As entradas CA1 e CA2 são conectadas à rede elétrica (220V), a qual é retificada e filtrada gerando uma tensão contínua representada por +/-HVDC, para alimentar a ponte inversora. A fonte de baixa tensão é derivada da de alta, obtendo-se +12Vcc, não regulado e a partir deste, +5Vcc regulado através de um regulador linear do tipo LM7805, para alimentar os circuitos de controle.

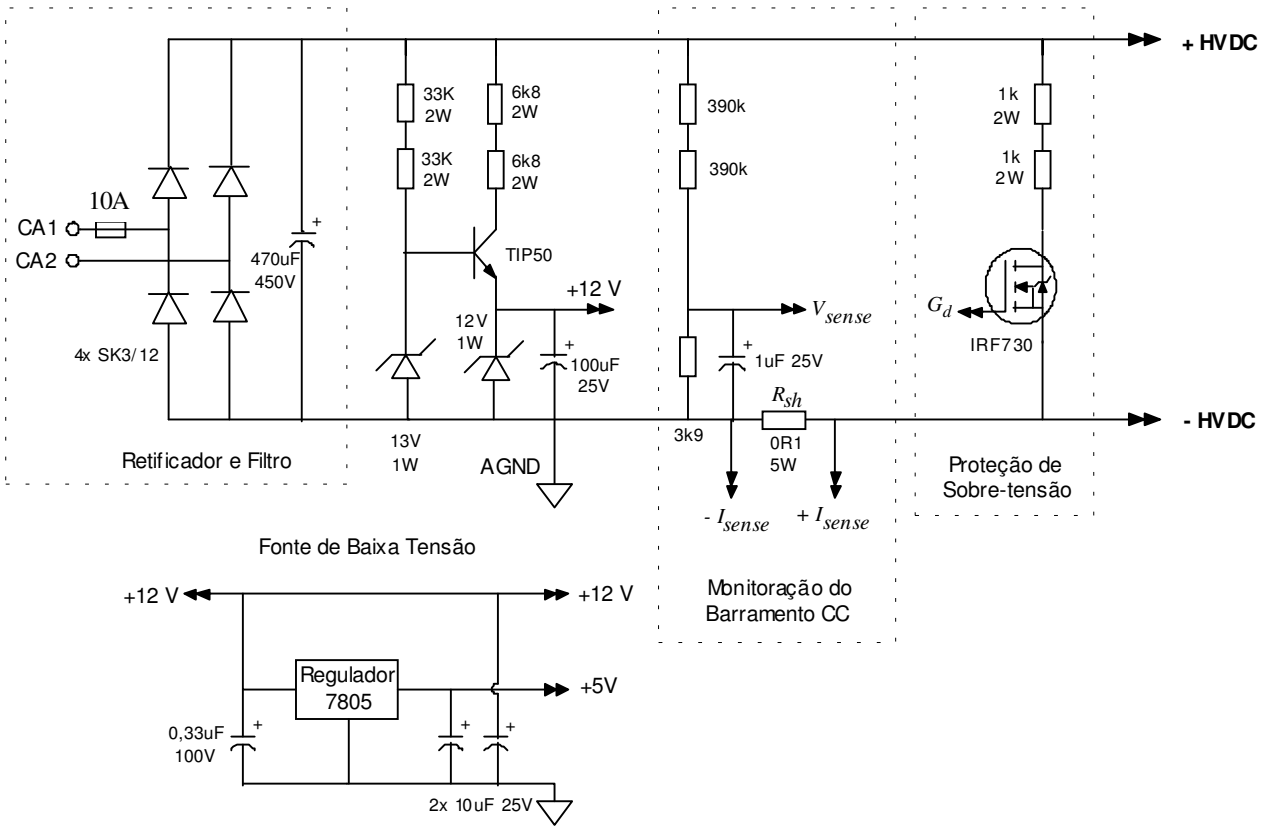


Figura 3.43 – Fonte de alimentação, circuitos de monitoração e proteção do barramento CC

3.8.5.3 Condicionamento dos sinais analógicos

Para a correta operação de qualquer circuito para acionamento de motores de corrente alternada, a ponte inversora deve ser monitorada com o intuito de controle e proteção. Normalmente em implementações de baixo custo monitora-se a corrente e a tensão do barramento de corrente contínua. As correntes alternadas de saída do inversor são monitoradas em aplicações mais sofisticadas ou de maior valor agregado. A literatura tem mostrado diversos métodos de medidas para a utilização e redução de sensores, tal como sugerido em [41] ou mesmo a ausência destes (*sensorless drives*), este último para aplicações comandadas por dispositivos com maior poder computacional, invariavelmente utilizando-se DSP's. A medição de tensão sobre o barramento de corrente contínua normalmente é feita por intermédio de um simples e barato divisor de tensão resistivo. Os dispositivos ideais para se medir as correntes alternadas e contínuas são sensores do tipo efeito Hall os quais possuem linearidade, precisão, não dissipam potência e são eletricamente isolados, porém são caros para aplicações de consumo. Uma maneira alternativa é a utilização de resistores *shunt*, o quais são baratos quando comparados com os dispositivos Hall, porém não são isolados e dissipam calor.

Na saída da fonte de alta tensão foram adicionados circuitos para monitoração da tensão e da corrente do barramento de corrente contínua e um circuito de proteção contra sobretensão também no barramento de corrente contínua (conhecido como *crowbar*). Todos os sinais monitorados foram condicionados analogicamente antes que o MCU possa fazer uso destes.

Um divisor de tensão faz a monitoração da tensão do barramento de corrente contínua, gerando o sinal V_{sense}, onde $V_{sense}\cong$1,55 V em 311 V, cujo condicionamento é feito como mostrado na figura 3.44 e a partir dele são gerados dois outros sinais: V_{BUS} que é limitado em 5 Vcc por um grampeador a diodos e filtrado, que fica disponível para conversão analógico para digital pelo MCU e o outro sinal gerado a partir de V_{sense} é o indicador de sobretensão (ST) que é obtido pela comparação de V_{BUS} com uma referência fixa de 2,0 Vcc. A partir da comparação realizada pelo LM393 a ocorrência de uma sobretensão será

detectada (ST=1) se a tensão do barramento de corrente contínua ultrapassar 30% da tensão nominal.

Um resistor *shunt* (0,1Ω/5W) é conectado em série com o barramento de corrente contínua para que a corrente contínua seja monitorada. O circuito da figura 3.45 mostra o condicionamento analógico, onde novamente são gerados dois sinais: o I_{BUS} é o sinal de tensão proporcional à corrente contínua do barramento. A partir de I_{BUS} é gerado o indicador de sobrecorrente (SC) que é obtido pela comparação de I_{BUS} com uma referência fixa de 5 Vcc. A partir desta comparação uma sobrecorrente será detectada (SC=1) se a corrente do barramento ultrapassar o limite pré-estabelecido pelo potenciômetro P1, o qual deve ser ajustado em função da corrente nominal do motor. Este potenciômetro permite o ajuste da detecção de sobrecorrente de 0 a 4 p.u., onde 1 p.u. de corrente equivale a 2,0 A. Deve ser dispensada especial atenção ao *firmware* para que não ocorra falsa detecção de sobrecorrente na partida do motor.

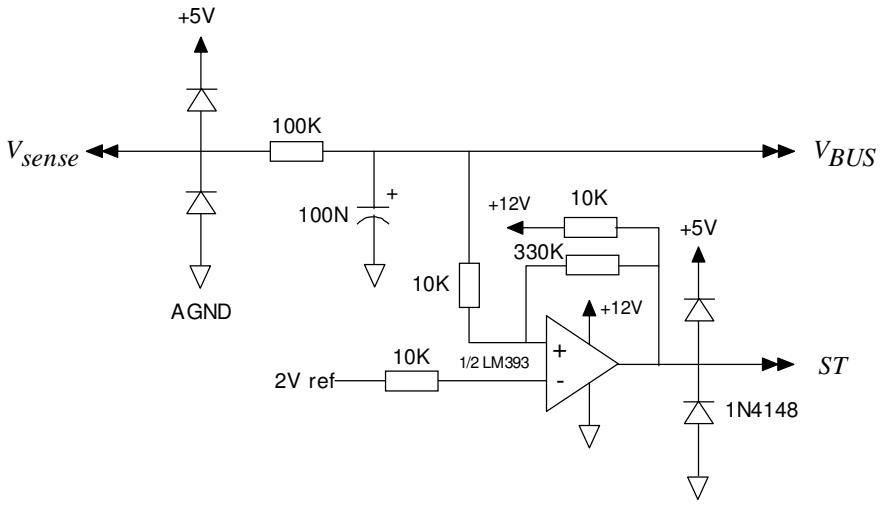


Figura 3.44 – Condicionamento analógico da tensão contínua

Os sinais analógicos de corrente e tensão estão disponíveis para serem amostrados e convertidos pelo conversor analógico para digital interno ao HC11A1, nos canais AN0 e AN1. Os indicadores de sobretensão e de sobrecorrente são lidos pelo MCU nas portas de entrada PC7 e PC6

respectivamente, os quais ficam retidos por *flip-flops* do tipo D (74LS74), até que as falhas sejam sanadas, figura 3.46.

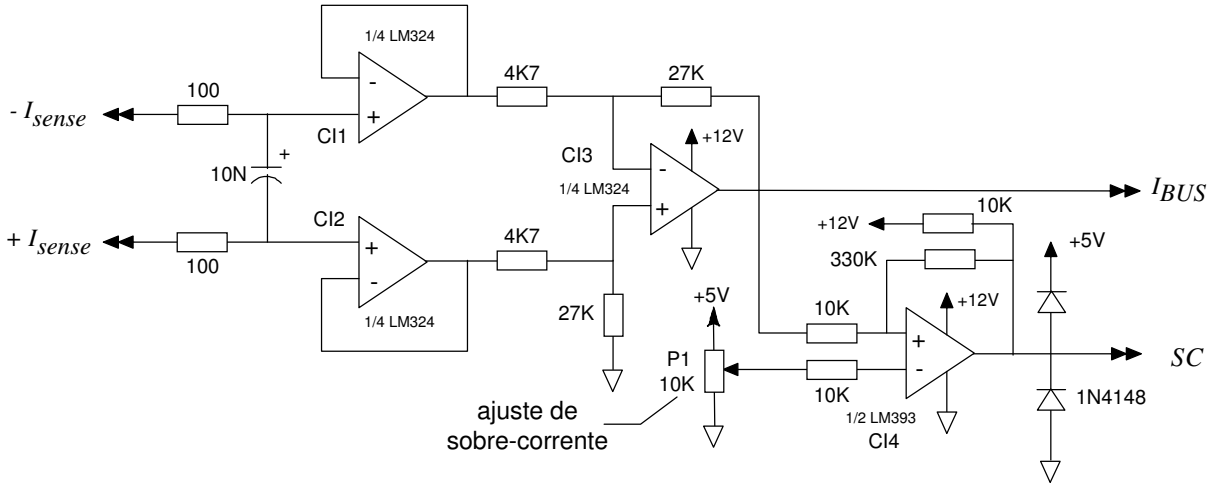


Figura 3.45 – Condicionamento analógico da corrente contínua

Na ocorrência de uma falha de *hardware*, uma interrupção externa (/XIRQ) é vetorizada e uma rotina aperiódica de tratamento de falhas é escalonada para ser executada interrompendo o fluxo corrente de execução, tal como indicado no fluxograma da figura 3.31. Assim que a falha for sanada, o *flip-flop* é "resetado" pelo *firmware*, através da porta de saída PC5. Na ocorrência de uma sobretensão, o sinal de controle da porta do transistor G_d na figura 3.43, é acionado para se colocar os resistores de descarga em paralelo com o barramento de corrente contínua. Se um nível máximo pré-estabelecido for ultrapassado, os transistores da ponte inversora são desligados. Por outro lado, se for detectada uma sobrecorrente, duas ações podem ser tomadas: quando um primeiro nível de sobrecorrente for detectado pode-se reduzir a tensão aplicada na expectativa de uma redução de corrente, porém, se um segundo nível de sobrecorrente for detectado, vinculado ao indicador SC, os sinais de controle das portas dos transistores da ponte inversora (AH_C, AL_C, BH_C, BL_C, CH_C, CL_C) são suprimidos e um sinal de desabilitação (SD, *shutdown*) dos *gate drives* é gerado.

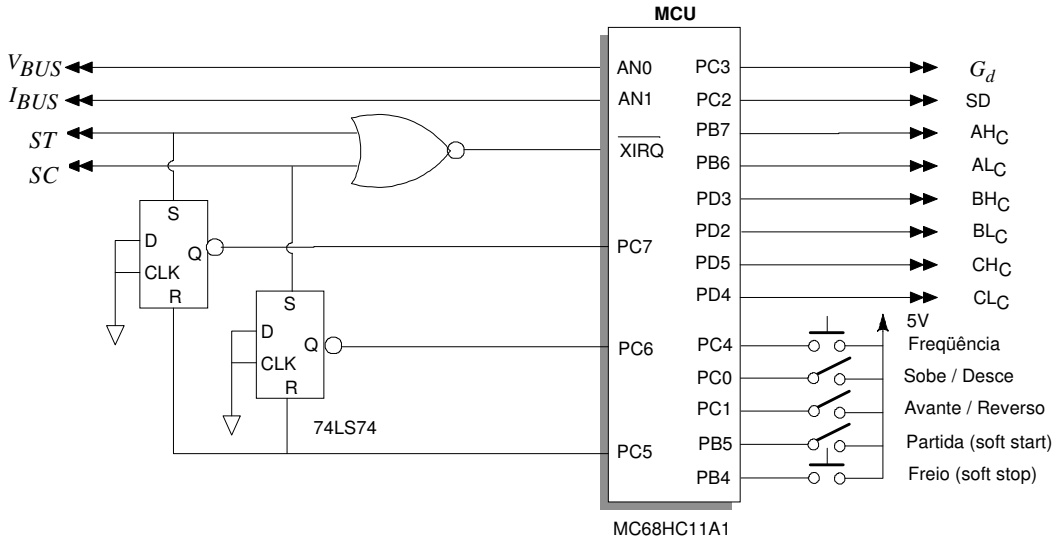


Figura 3.46 – Leitura dos sinais analógicos, IHM e disparo dos transistores pelo MCU

3.8.5.4 Acionador de porta dos transistores (*gate drive*)

Os circuitos de acionamento das portas dos transistores (*gate drives*) devem ser capazes de acionar os dois transistores de uma perna da ponte, isso significa que devem ser capazes de acionar a porta do transistor superior, cujo potencial não está referenciado ao terra do circuito de controle, tal como mostrado na figura 3.47. O circuito integrado utilizado foi o IR2112 *High and Low Side Driver*, fabricado pela *International Rectifier*.

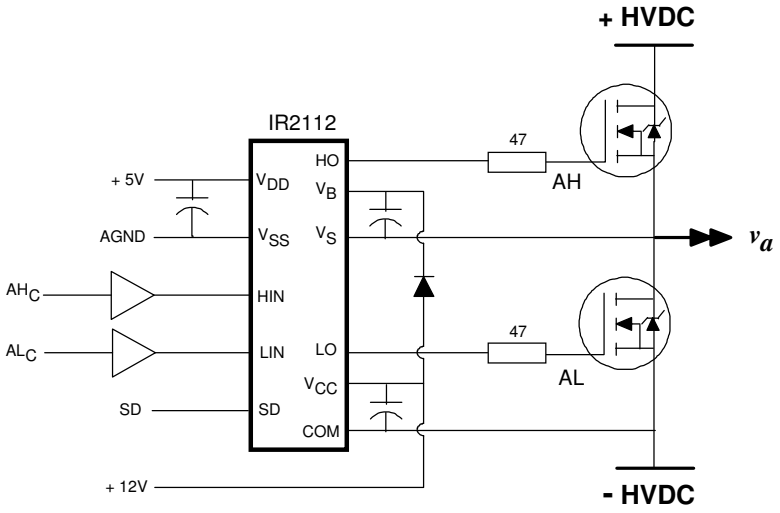


Figura 3.47 – *Gate drive* para a perna “a” da ponte inversora

Ao todo são utilizados três CIs, um para cada perna da ponte. O IR2112 possui as seguintes características [42]: (a) opera com tensões de barramento até 600 Vcc, (b) fornece tensão de porta de 10 a 20 Vcc, (c) compatível com tensão de controle de 5 a 20 Vcc, (d) possui saída em fase com a entrada, (e) fornece corrente de saída de +200 mA a -420 mA, (f) possui t_r = 125 ns e t_f = 105 ns, (g) possui atraso de propagação 30 ns, (h) possui capacidade de *drive* para MOSFET ou IGBT, (i) os *drives* superior e inferior são independentes, (j) o *drive* superior é flutuante, (k) sua entrada lógica é compatível com CMOS e TTL.

3.8.5.5 Ponte inversora

Este trabalho não explorou a otimização do estágio de potência visando sua redução de custo, bem como não foram consideradas topologias alternativas para a redução do número de chaves semicondutoras [1], [2] e [3]. Como o interesse residiu no desenvolvimento e nas medidas de desempenho do *firmware*, o circuito de potência foi deliberadamente super dimensionado para atender aos requisitos da carga sem correr riscos de falhas, porém sem esquecer os cuidados básicos de proteção dos transistores, tal como indicado em [43].

A ponte inversora trifásica, tal como descrita nas seções anteriores, é o circuito fundamental para se implementar o conversor CC-CA e é o estágio mais crítico do acionamento sendo o mais caro e bastante susceptível a falhas se não houver preocupação com sua proteção. Sem dúvida é o circuito que mais queima nos acionamentos transistorizados, sejam eles de aplicação industrial ou doméstica. Isso explica o cuidado que todos os fabricantes deste tipo de acionamento possuem com a monitoração e proteção da ponte inversora. A figura 3.48 é uma reapresentação da figura 3.17, porém substituindo os transistores genéricos pelos IGBT's, tal como implementado no protótipo.

Nesta figura os capacitores "CA", "CB" e "CC", de 0,1µF/630V, funcionam como *"snubbers"* de baixo custo contra transitórios de tensão e são indicados para circuitos de baixa potência [43]. Estes capacitores devem ser de filme de polipropileno de baixa perda e baixa indutância e devem ser conectados o mais próximo possível dos transistores.

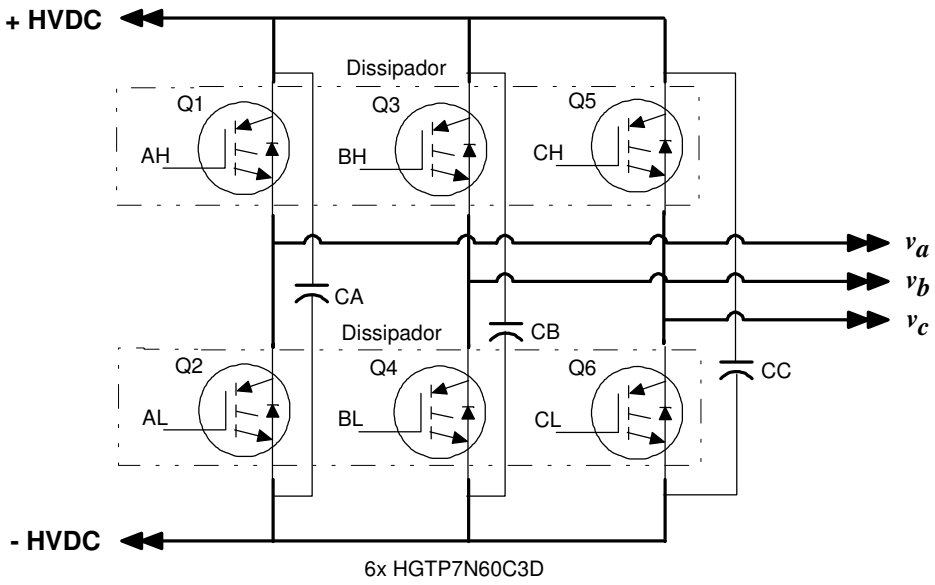


Figura 3.48 – Ponte inversora com IGBT's

O objetivo é reduzir a indutância do barramento de corrente contínua para minimizar oscilações parasitas, de modo a reduzir as perdas e a fadiga de chaveamento nos transistores.

O protótipo montado não prevê a monitoração e medição da corrente alternada de saída da ponte inversora. Isso só foi possível porque o erro introduzido pela inclusão do tempo-morto não foi compensado, tal como justificado na seção 3.7.2 e a proteção de sobrecorrente no lado alternado (saída) é feita indiretamente pela proteção de corrente do barramento de corrente contínua. A ponte inversora foi montada com transistores IGBT do tipo HGTP7N60C3D fabricados pela Intersil, que possuem diodos ultra-rápidos em anti-paralelo. O HGTP7N60C3D possui as seguintes características [44]: (a) corrente de coletor: 14 A para T_c = 25°C, 7A para T_c = 110°C, (b) suporta tensão coletor/emissor até 600Vcc, (c) suporta corrente média direta no diodo até 8A em T_c = 110°C, (d) potência dissipada máxima de 60W em T_c = 25°C, (e) suporta curto circuito durante 1µs com V_{GE} = 15V, ou 8µs com V_{GE} = 10V e (f) possui tempo de descida da corrente de 140 ns para T_j = 150°C.

O Apêndice B apresenta um resumo com comparação sobre chaves semicondutoras possíveis de serem utilizadas em pontes inversoras.

Capítulo 4

ESTIMATIVA DE DESEMPENHO DE PROCESSADORES

4.1 Introdução

Diante de uma aplicação que utilize técnicas de sistemas embarcados, o projetista do *software* tem de determinar o tamanho das memórias de programa e de dados necessárias para a aplicação e decidir como atender às restrições de tempo das diversas tarefas que compõem o *software*, para poder escolher com que processador trabalhará. A sua decisão influirá diretamente no desempenho e no custo do projeto. Tradicionalmente os projetistas se valem de suas experiências anteriores para as primeiras estimativas, o que muitas vezes resulta em um processo iterativo de convergência lenta e cara.

Na última década muito se escreveu sobre co-projeto, ou co-síntese, entre *hardware* e *software* para sistemas embarcados [45] a [48], cujo objetivo é o de eliminar o empirismo das primeiras decisões do projeto, fornecendo aos projetistas ferramentas de simulação e métricas para tomada de decisões. A computação embarcada é única, devido ao problema intrínseco do co-projeto entre *hardware* e *software*. Estes dois componentes devem ser projetados juntos para assegurar que a implementação não somente funcione apropriadamente, mas atinja os requisitos de desempenho, custo e confiabilidade estipulados para a aplicação. Enquanto as implementações de *hardware* possuem desempenho elevado, associados a uma baixa flexibilidade com relação a alterações, as porções do projeto que são executadas em *software* são bem mais flexíveis a alterações. Em aplicações destinadas a produção de larga escala, geralmente deseja-se um particionamento não uniforme entre *hardware* e *software*, isto é, uma porção elevada de *software* sendo executada no menor *hardware* possível. Entre as vantagens do *software* em relação ao *hardware*, destacam-se: menor custo, menor tempo de desenvolvimento e facilidade de alteração. A referência [49] cita métricas de qualidade geralmente utilizadas para caracterizar um projeto de

sistema embarcado. Dentre as mais importantes estão as métricas de custo e desempenho do *hardware/software* da implementação.

4.2 Definições

Para a discussão que se segue fazem-se necessárias algumas definições:

4.2.1 Sistema

Conjunto de recursos computacionais e tarefas que compõe o *software* de uma determinada aplicação, o qual interage com o seu ambiente, usualmente pela execução seqüencial das tarefas.

4.2.2 Sistema de tempo real crítico

São sistemas caracterizados por conter um conjunto de tarefas que possuem restrições de tempo de processamento, restrições de precedência e compartilhamento dos recursos computacionais. Tais sistemas não podem ter suas restrições temporais violadas sob o risco de causar mau funcionamento, ou até mesmo acidentes (falhas catastróficas).

4.2.3 Tarefa

De acordo com [50] e [51] tarefa é um módulo de programa que pode ser invocado pelo processador para executar uma função específica e que, em conjunto com as demais tarefas do sistema, contribui para o processamento global da aplicação. Comumente dois tipos de tarefas podem ser encontrados em sistemas de tempo real crítico: as periódicas e as aperiódicas.

Tarefas periódicas são as mais comuns e como o próprio nome diz, são invocadas pelo processador apenas uma vez dentro de um determinado período. Já as aperiódicas podem ser invocadas em qualquer instante e normalmente caracterizam-se por possuírem prazos severos de execução (*hard deadline*). Em sistemas embarcados geralmente estão associadas a módulos de *software* para tratamento de falhas.

4.2.4 Período

O período de execução de uma tarefa periódica é a taxa de repetição entre duas invocações sucessivas, figura 4.1.

4.2.5 Prazo

O prazo de execução de uma tarefa é o tempo de que ela dispõe para terminar sua execução. A tarefa cuja execução não pode ultrapassar seu prazo é dita possuir *hard deadline*. Já a tarefa onde a violação do seu prazo é aceitável é dita ter *soft deadline*.

4.2.6 Tempo de execução

É o tempo máximo que uma tarefa pode gastar para ser executada, contado a partir do momento em que ela é invocada pelo processador, figura 4.1.

4.2.7 Tempo de pronto

É o instante mais próximo que uma tarefa pode iniciar sua execução.

4.2.8 Prioridade

A prioridade de uma tarefa estabelece a urgência da sua execução. Por convenção [50], considera-se que quanto menor for o período da tarefa, maior será sua prioridade e vice-versa.

A tabela 4.1 e o diagrama de Gantt da figura 4.1 ilustram os conceitos de especificação de tarefas, para um conjunto de duas tarefas. No instante t = 0 as tarefas T_1 e T_2 estão prontas e começam a ser executadas. Ao terminar a tarefa T_1 no instante t = 4, a tarefa T_2 assume a CPU e permanece até o instante t = 10 quando é interrompida devido à invocação da segunda instância de T_1 que novamente é executada até o instante t = 14, quando então dá lugar a T_2 para finalizar seu processamento pendente.

Tabela 4.1 Especificação de tarefas

Tarefa	Tempo de execução	Período	Prioridade
T_1	4	10	1
T_2	8	20	2

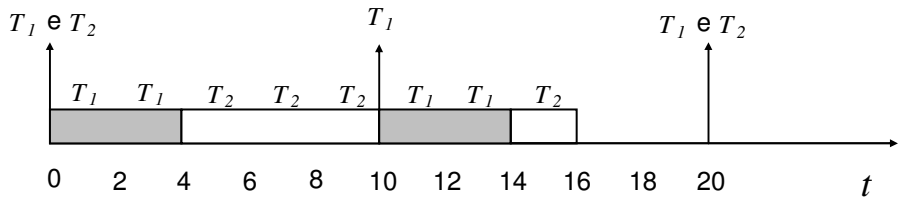


Figura 4.1 – Execução das tarefas no transcorrer do tempo

4.3 Métricas e estimativas de desempenho

Métricas são atributos que qualificam e quantificam um sistema de forma a permitir a comparação de sistemas afins. Como exemplo de métricas pode-se citar: custo, peso, volume, consumo de potência, tempo de execução, tamanho de memória entre outras. De acordo com [49] existem duas formas de se calcular uma métrica: através de uma implementação específica e através de uma estimativa. Este trabalho considera as seguintes métricas: tamanho da memória de programa, tempo de execução das rotinas do *software* e análise de escalonamento do conjunto de tarefas do *software*.

A implementação de uma aplicação específica, conseqüentemente com a construção do *hardware* e/ou *software*, proporciona resultados precisos dos valores de quaisquer métricas, no entanto mostra-se pouco prática do ponto de vista de obtenção da métrica, pois demanda tempo e recursos materiais para ser implementada.

Já uma estimativa, a qual é obtida através de uma implementação mais simples e rápida, baseada em um modelo do sistema real que não inclui diversos detalhes e otimizações, no entanto, geralmente fornece valores suficientemente precisos para se comparar métricas de diferentes sistemas. A velocidade de implementação e a precisão são parâmetros antagônicos e concorrentes em uma estimativa da métrica. Mas a velocidade na obtenção dos resultados implica na omissão de detalhes e conseqüentemente em imprecisões enquanto que, maior precisão implica na inclusão de detalhes com conseqüente aumento do tempo de implementação.

A seguir são descritas diversas técnicas que ajudam a estimar o desempenho de processadores digitais com o intuito de verificar se um determinado processador será capaz de executar todas as tarefas que compõe o

software do sistema, atendendo aos seus requisitos de prazos e capacidade de memória necessária à implementação. As técnicas aqui descritas são de uso geral, no entanto foram particularizadas para aplicações de controle digital de motores elétricos, onde são comparadas algumas aplicações, em diversos processadores comerciais. Uma delas é comprovada através de resultados experimentais.

4.3.1 Estimativa de *software*

Para uma dada implementação de *software* as tarefas que o compõem devem ser compiladas utilizando-se o conjunto de instruções do processador escolhido para a aplicação. Para o caso de sistemas com um único processador a execução das tarefas concorrentes devem ser intercaladas de forma a satisfazer as dependências de dados e restrições de tempo. De acordo com [49], dois modelos podem ser adotados para a estimativa de *software*: modelo específico de processador e modelo genérico de processador.

4.3.1.1 Estimativa com modelo específico de processador

Neste modelo, representado pela figura 4.2, o valor exato de uma métrica é computado pela compilação de cada tarefa utilizando-se o conjunto de instruções do processador escolhido e submetendo-as ao compilador específico do processador em questão. Das informações de tempo (*timing*), tais como o número de ciclos de máquina (*clock*) necessários para executar cada instrução e de tamanho (*size*), tal como o número de *bytes* requerido para cada instrução, associadas a cada processador específico, pode-se determinar precisamente o tamanho e o tempo de execução das tarefas compiladas. Por exemplo, se o *software* de uma determinada aplicação for implementado em um processador específico “A” (comercialmente disponível), ele deve ser escrito com o seu conjunto de instruções e utilizar o compilador da família deste dispositivo. Mesmo que sejam utilizadas linguagens de alto nível, tal como C, ainda existe a dependência do compilador que irá traduzir as instruções de alto nível para baixo nível (*assembly*), onde diferentes tamanhos de códigos serão obtidos para

diferentes processadores, acarretando em diferenças de tamanhos e *timing* para diferentes processadores.

Após a compilação, utilizando-se as informações de tempo e tamanho, determina-se o desempenho e o tamanho do *software* implementado. Se por outro lado for utilizado um processador “B”, de um outro fabricante, todo o processo deve ser refeito para este novo processador.

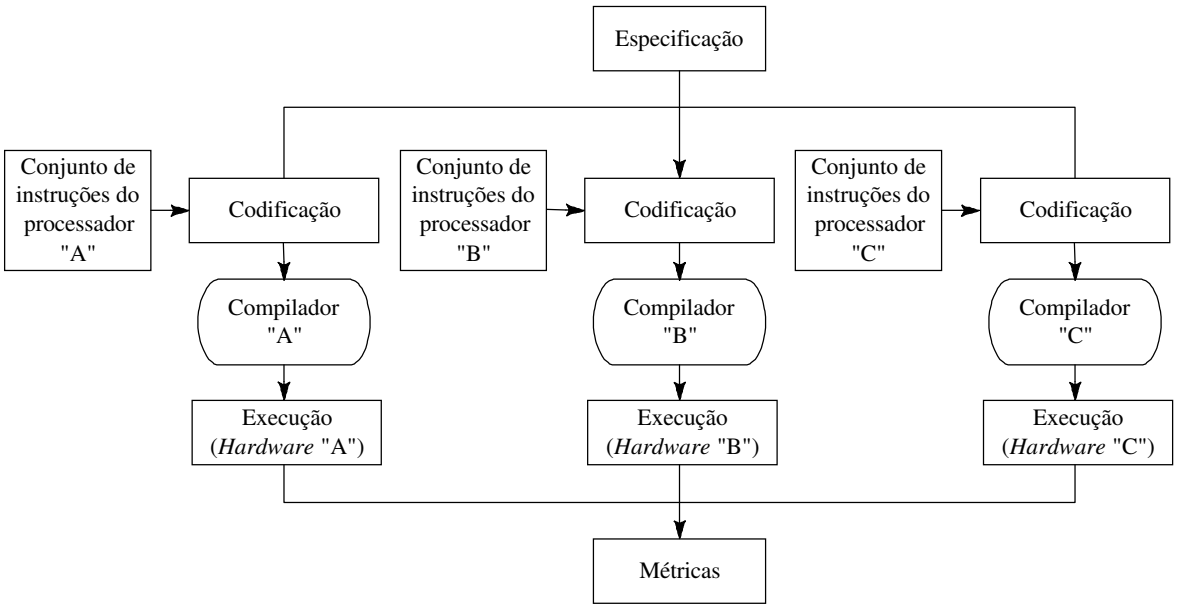


Figura 4.2 – Estimativa de *software* por modelo específico de processador

A principal vantagem deste método é que as estimativas são muito precisas, pois trata-se de uma implementação específica. Entretanto, do ponto de vista da estimativa, este método não é muito prático pois requer compiladores dedicados a cada processador de interesse, necessitando que o *software* seja escrito novamente para dispositivos de diferentes famílias ou fabricantes, o que muitas vezes não é uma tarefa fácil devido à baixa portabilidade dos *firmwares* quando se utilizam linguagens de baixo nível (*assembly*).

A figura 4.2 representa uma situação hipotética onde se deseja experimentar três processadores específicos, denominados por “A”, “B” e “C”, de três fabricantes diferentes, em uma mesma aplicação para predizer se os processadores estarão adequados a ela e qual deles será melhor utilizado por este *software*.

Partindo-se de uma especificação, codifica-se a aplicação (escreve-se o *software*) utilizando-se o conjunto de instruções de cada processador, neste caso o mesmo *software* deverá ser escrito três vezes, ou adaptado para cada processador específico. O segundo passo é compilar o conjunto de tarefas utilizando-se os respectivos compiladores. Como resultado das compilações obtêm-se os valores exatos das métricas, tais como o tamanho da memória de programa (números de *bytes*) da aplicação para cada processador testado e, executando-se o *software* em cada processador (para isso são necessárias diferentes plataformas de *hardware*) determinam-se os tempos de execução das aplicações. Se por ventura um processador não conseguir executar a aplicação devido a restrições temporais das tarefas, por exemplo um prazo muito curto, a aplicação deverá ser escrita novamente buscando-se a sua otimização. Após sucessivas tentativas pode-se concluir que este processador não se presta à aplicação em questão. Cabe ressaltar que este é um processo lento, trabalhoso e caro.

4.3.1.2 Estimativa com modelo genérico de processador

Ao invés de utilizar diferentes compiladores e plataformas de hardware, uma alternativa é a obtenção de estimativas de desempenho através de um modelo genérico de processador proposto em [52]. Neste método as tarefas são definidas utilizando-se um conjunto reduzido de instruções, denominado instruções genéricas (IG), as quais são divididas em cinco classes: (1) Instruções aritméticas/lógicas/relacionais (ALR); (2) Instruções de movimentação e armazenagem (MAR); (3) Instruções de desvio condicional (DEC); (4) Instruções de desvio incondicional (DEI) e (5) Instruções de chamada de subrotina (SUB).

Cada IG é definida por duas características básicas: número de bytes e número de ciclos de máquina da instrução.

A referência [49] estabelece um tamanho fixo de 3 bytes para todas IGs, no entanto esta simplificação acarreta um erro na estimativa do tamanho da memória de programa necessária para se implementar uma dada aplicação. Este trabalho utiliza um método alternativo de estimativa com modelo genérico de processador

para determinar o tamanho de cada IG, denominado Método do Tamanho Médio, semelhante ao descrito em [68], o qual está descrito a seguir.

4.3.1.3 Estimativa com modelo genérico de processador – Método do Tamanho Médio

Das folhas de dados fornecidas pelos fabricantes dos processadores para os quais se deseja simular uma aplicação, obtêm-se as informações de tamanho e do número de ciclos de máquina para todas as instruções do processador. Organizando-as por afinidade de função, divide-se o conjunto de instruções em cinco classes, tal como descrito acima. Os tamanhos e os números de ciclos de cada IG serão obtidos calculando-se as médias dos tamanhos e dos números de ciclos para cada classe e aproximando-as pelo inteiro imediatamente inferior, se a parte fracionária da média for menor que 0,5 ou, será aproximado pelo inteiro imediatamente superior, se a parte fracionária da média for maior ou igual a 0,5, expressões (4.1) e (4.2).

$$\text{Tamanho IG} = \begin{cases} \left\lfloor \dfrac{\sum \text{Número de bytes}}{\text{Número de instruções}} \right\rfloor & \text{se parte fracionária} < 0{,}5 \\ \left\lceil \dfrac{\sum \text{Número de bytes}}{\text{Número de instruções}} \right\rceil & \text{se parte fracionária} \geq 0{,}5 \end{cases} \tag{4.1}$$

$$\text{Núm. de ciclo IG} = \begin{cases} \left\lfloor \dfrac{\sum \text{Número de ciclos}}{\text{Número de instruções}} \right\rfloor & \text{se parte fracionária} < 0{,}5 \\ \left\lceil \dfrac{\sum \text{Número de ciclos}}{\text{Número de instruções}} \right\rceil & \text{se parte fracionária} \geq 0{,}5 \end{cases} \tag{4.2}$$

Sendo que o símbolo $\lfloor\ \rfloor$ representa a aproximação pelo inteiro imediatamente inferior e $\lceil\ \rceil$ representa a aproximação pelo inteiro imediatamente superior.

A tabela 4.3 que representa o arquivo de tecnologia, ou technology file, de acordo com a definição de [49], para diversos microcontroladores e DSPs exemplificando o processo de obtenção das características básicas das IGs. Faz parte ainda do arquivo de tecnologia o tempo do ciclo de máquina, geralmente

especificado em nano segundos, que depende do valor da freqüência do oscilador (clock) selecionado para a aplicação e do fator de divisão do barramento interno do processador. No caso do MC68HC11A1 o menor fator de divisão é 1 e a freqüência de clock máxima é de 2 MHz, portanto o menor valor para um ciclo de máquina será 500 ns. Em princípio, quanto menor for o ciclo de máquina, mais rápida será a execução da aplicação. Este processo deve ser repetido para todos os processadores de interesse.

Ainda na tabela 4.3 que mostra os arquivos de tecnologia para sete processadores específicos, os quais foram escolhidos em função das seguintes características: processadores lentos (COP8SGE7 com freqüência de clock de 1 MHz e MC68HC11A1), processadores rápidos (PIC17C752 com clock de 33 MHz, TMS320LC2402 com 40 MHz e DSP56800 com 40 MHz) e processadores dedicados ao controle de máquinas elétricas (ST92141 com 25 MHz, MC68HC08MR4 com 8,2 MHz, PIC17C752 e TMS320LC2402). Pode se observar que foram escolhidos processadores com diferentes características para ressaltar o impacto causado no desempenho destas máquinas e deve ser observado também que alguns se enquadram em mais de uma categoria. Desta forma, quando for aplicado o método aqui proposto, poder-se-á verificar (ou estimar) a diferença de desempenho de cada tipo de processador.

O processo de estimativa utilizando o modelo genérico de processador dá-se de acordo com a figura 4.3, cujo objetivo é a obtenção das métricas de comparação entre os processadores. De posse de uma especificação técnica da aplicação de interesse, elabora-se um pseudo-código utilizando o conjunto das IGs. O próximo passo é aplicar às IGs os arquivos de tecnologia dos processadores para os quais se deseja estimar o desempenho, associando a cada IG o seu respectivo tamanho e número de ciclos. A “compilação genérica” se dá pela soma dos números de bytes e dos ciclos de máquina. A tabela 4.4 exemplifica esta compilação utilizando um trecho do código da aplicação implementada no protótipo experimental e escrita para o MC68HC11A1, onde se observam as diferenças entre o avaliado e o experimental. O código completo é apresentado no Apêndice A.

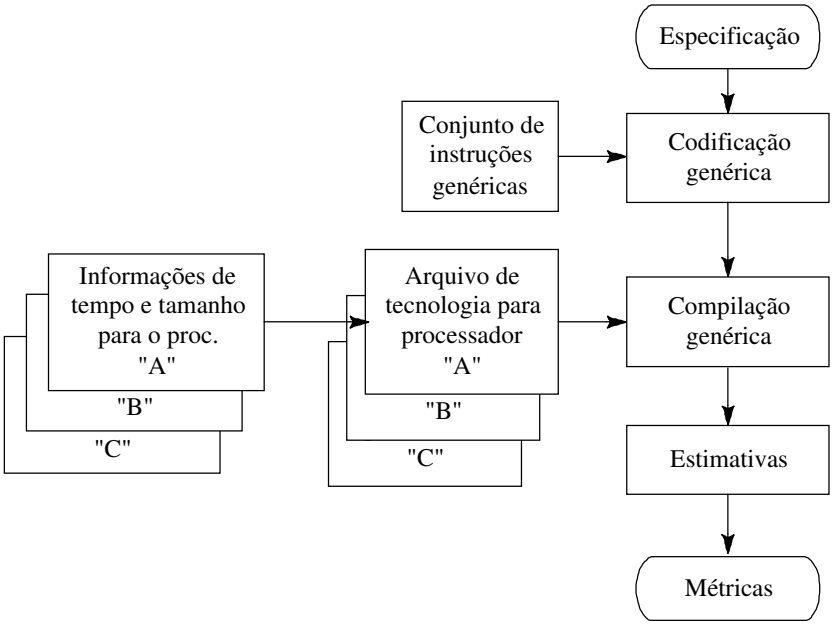


Figura 4.3 – Processo de compilação genérica

O código genérico permite a aplicação rápida de diferentes arquivos de tecnologia, tabelas 4.4(a) e 4.4(b), nas quais se observam as variações dos números de *bytes* e dos números de ciclos de máquina. A tabela 4.4(a) foi escrita considerando um trecho real do código escrito para o protótipo experimental. Já a tabela 4.4(b) foi obtida pelo método aqui proposto. A desvantagem deste método é o erro introduzido pelo conjunto das instruções genéricas, o qual representa a média do conjunto das instruções do processador específico.

Este erro pode ser observado na diferença entre os totais das tabelas 4.4(a) e 4.4(b), porém ele é baixo. As tabelas 4.5(a) e 4.5(b) exemplificam a compilação genérica para os MCUs MC68HC08MR4 e ST92141 respectivamente. Os resultados destas tabelas podem ser comparados com a tabela 4.4(b), de onde pode se predizer que para este trecho de programa o MR4 ocupará menos memória que o HC11A1 (13 *bytes* contra 25), já o ST92 ocupará mais (29 *bytes* contra 25).

Com relação ao número de ciclos, o MR4 necessitará de menos tempo para execução deste trecho (34 ciclos de máquina contra 47) e o ST92, necessitará de um tempo levemente inferior (45 ciclos contra 47).

Tabela 4.2 – Determinação das IG para o MC68HC11A1

PROC.	**MC68HC11A1**			GENÉRICO		
CLASSE	**INSTRUÇÃO ESPECÍFICA**	NUM. BYTES	NUM. CICLOS	**INSTRUÇÃO GENÉRICA**	NUM. BYTES	NUM. CICLOS
1	ADD 8 BIT	2	3	ALR	2	4
	ADD 16 BIT	3	5			
	SHIFT 8 BIT	2	4			
	DEC	2	4			
	INC	2	4			
	MULT 8 BIT	1	10			
	NEG	2	4			
	CLR	3	3			
	SUB 8 BIT	2	3			
	SUB 16 BIT	2	5			
	AND	2	3			
	COMP	2	6			
	OR	2	3			
	ROL	2	4			
	COMPARE 8 BIT	2	4			
2	LOAD	2	3	MAR	2	3
	PUSH	1	3			
	PULL	1	4			
	STORE	2	4			
	TRANSFER	1	3			
	EXCHANGE	2	3			
3	BRANCH 8 BIT	2	3	DEC	2	3
4	JUMP	2	2	DEI	2	2
5	RETURN SUB	1	6	SUB	1	9
	RETURN INT	1	12			

Tabela 4.3 – Arquivos de tecnologia para vários processadores

Processador	Fabricante	*Clock* Interno (MHz)	Ciclo de Máquina (ns)	Instruções Genéricas									
				Classe 1		Classe 2		Classe 3		Classe 4		Classe 5	
				Byte	Ciclos	Byte	Ciclos	Byte	Ciclos	Byte	Ciclos	Byte	Ciclos
COP8SGE7	NSC	1,0	1000	3	4	2	3	2	3	2	3	1	12
MC68HC11A1	Motorola	2,0	500	2	4	2	3	2	3	2	2	1	9
ST92141	ST	25,0	160	3	4	2	3	3	4	3	4	1	5
MC68HC08MR4	Motorola	8,2	120	1	2	1	2	1	2	1	6	1	10
PIC17C752	Microchip	33,0	120	1	1	1	1	1	1	1	2	1	2
TMS320LC2402	Texas	40,0	50	1	1	1	1	1	1	1	4	1	4
DSP56800	Motorola	40,0	30	1	1	1	1	1	1	1	2	1	2

Tabela 4.4 – Compilação genérica para MC68HC11A1 (a) experimental e (b) avaliado

Código Fonte do MC68HC11A1	Num. Bytes	Num. Ciclos
ldaa PORTC	2	3
bita #$40	2	3
bne DESLIGA_AH	2	3
anda #$bf	2	3
staa PORTC	2	3
ldab #$40	2	3
stab TFLG1	2	3
ldd TWA2	3	3
std TOC2	2	4
ldaa PORTC	2	3
ora #$80	2	2
staa PORTC	2	3
rti	1	12
Total	**26**	**48**

(a)

Codificação Genérica	Num. Bytes	Num. Ciclos
MAR	2	3
DEC	2	3
DEC	2	3
ALR	2	4
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
ALR	2	4
MAR	2	3
SUB	1	9
Total	**25**	**47**

(b)

Tabela 4.5 – Compilação genérica para (a) MC68HC08MR4 e (b) ST92141

Conversão para IG do MC68HC08MR4	Num. Bytes	Num. Ciclos
MAR	1	2
DEC	1	2
DEC	1	2
ALR	1	2
MAR	1	2
MAR	1	2
MAR	1	2
MAR	1	2
MAR	1	2
MAR	1	2
ALR	1	2
MAR	1	2
SUB	1	10
Total	**13**	**34**

(a)

Conversão para IG do ST92141	Num. Bytes	Num. Ciclos
MAR	2	3
DEC	3	4
DEC	3	4
ALR	3	4
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
MAR	2	3
ALR	3	4
MAR	2	3
SUB	1	5
Total	**29**	**45**

(b)

4.3.2 Estimativa do tempo de execução

O tempo de execução de um *software* é o tempo que a CPU necessita para executar a seqüência de instruções que compreende o programa, desde o instante da sua invocação até sua última instrução, antes que o laço fechado (*loop*) se reinicie, ou seja, é tempo para se executar uma vez o laço do programa, figura 4.4.

O tempo de execução pode ser estimado de duas formas: por simulação dinâmica ou por simulação estática [49].

A simulação dinâmica consiste em executar e armazenar na memória o número total de ciclos de máquina (*total_ciclos*) requeridos para a execução de um programa, que foi codificado para um determinado processador, de tal forma que sabendo se o tempo de um ciclo de máquina (T_{clk}), calcula-se precisamente o tempo de execução do programa usando a expressão 4.3.

$$exec\_time = total\_ciclos \cdot T_{clk} \tag{4.3}$$

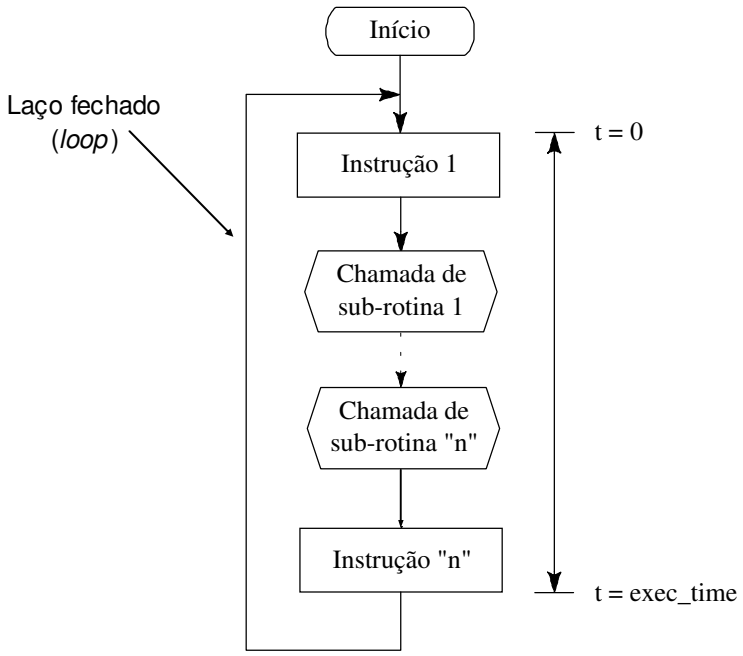


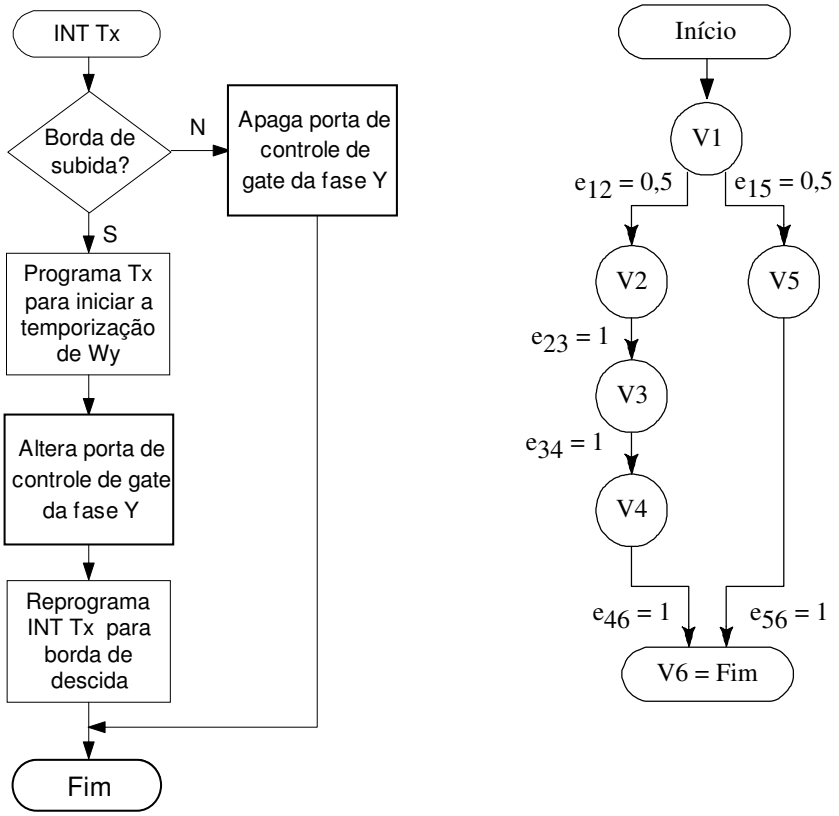
Figura 4.4 – Definição do tempo de execução

Para diferentes conjuntos de dados de entrada esta simulação pode oferecer diferentes números de ciclos de máquina, devido à dependência dos dados com relação aos desvios condicionais e *loops* do programa [49].

A estimativa estática, por outro lado, é insensível a possíveis variações nos dados de entrada e pode produzir resultados razoavelmente precisos se o número de iterações dos *loops* são conhecidos e se as probabilidades dos desvios condicionais possam ser previstas. A “probabilidade de um desvio condicional” é a

probabilidade de um desvio do *software* ser executado, durante a execução de uma única vez do laço do programa [49].

Para exemplificar a probabilidade de execução dos desvios condicionais, tome-se como exemplo um dos fluxogramas da figura 3.30, reproduzido na figura 4.5(a). Este fluxograma é representado em termos dos vértices V1 a V6, figura 4.5(b), que são as instruções do programa, onde para cada desvio condicional (neste caso apenas o vértice V1) é associado uma probabilidade de execução do desvio. Neste caso específico é de 50%, pois esta rotina é invocada a cada ciclo da MLP, de tal forma que ora está na borda de subida, ora está na borda de descida.



(a) (b)

Figura 4.5 – Probabilidade de execução dos vértices

A freqüência de execução, *freq*, de qualquer vértice Vj depende da probabilidade de execução, $prob(e_{ij})$, de todos os seus vértices predecessores imediatos, dada pela expressão 4.4.

$$freq(V_j) = \sum_{\substack{vertices \\ predecessores}} freq(V_i) \cdot prob(e_{ij}) \qquad (4.4)$$

Da figura 4.5(b) pode se escrever o seguinte conjunto de equações:

freq(início) = 1,0
freq(V1) = 1,0 x *freq*(início)
freq(V2) = 0,5 x *freq*(V1)
freq(V3) = 1,0 x *freq*(V2)
freq(V4) = 1,0 x *freq*(V3)
freq(V5) = 0,5 x *freq*(V1)
freq(V6=fim) = 1,0 x *freq*(V4) + 1,0 x *freq*(V5)

Resolvendo o sistema de equações:
freq(V1) = 1,0
freq(V2) = 0,5
freq(V3) = 0,5
freq(V4) = 0,5
freq(V5) = 0,5
freq(V6=fim) = 1,0

Uma vez conhecida a freqüência de execução de cada vértice do programa aplica-se o procedimento abaixo:

Se uma tarefa de *software* "B" é descrita por uma seqüência de n_j linhas de código, o tempo de execução desta tarefa dependerá do número de ciclos de máquina de cada instrução (*ciclos*) e do período de *clock* (T_{clk}) selecionado para a aplicação. Tal como na simulação dinâmica o tempo de execução (*exec_time*) em segundos, para esta tarefa, pode ser dado por:

$$exec\_time(B) = \sum_{n_j} ciclos(n_j) \cdot T_{clk} \qquad (4.5)$$

Como exemplificado acima, uma tarefa é constituída por comandos seqüenciais que possuem desvios e estruturas de iterações, tais como comandos de *loops, if* e *case*, que limitam a aplicação da expressão (4.5), fazendo-se necessário a inclusão da freqüência de execução dos vértices.

A probabilidade de ocorrer um desvio é a medida de sua freqüência de execução. Sob esta perspectiva, para se determinar corretamente o tempo de execução deve-se ponderar cada vértice do programa pela freqüência de execução, a qual define o número de vezes que um determinado bloco básico será executado durante uma única execução do fluxo do programa, então a expressão (4.5) pode ser re-escrita por:

$$exec\_time(B) = \sum_{n_j} ciclos(n_j) \cdot T_{clk} \cdot freq(n_j) \qquad (4.6)$$

Como exemplo desta estimativa pode-se ainda utilizar a figura 4.5(a), a qual foi codificada genericamente considerando-se o MCU MC68HC11A1 e usando o período de *clock* obtido na tabela 4.3 para este processador (T_{clk} = 500 ns), a tabela 4.6 mostra a estimativa do tempo de execução para o trecho de código considerado.

Tabela 4.6 – Exemplo de cálculo do tempo de execução

Rótulo	Instrução Genérica	Operando	Ciclos	Freq.	exec_time (µs)
INT T1:	DEC	COND,BORDA_DESCIDA	3	1,0	1,5
	MAR	CONSTANTE	3	0,5	0,75
	MAR	CONSTANTE	3	0,5	0,75
	MAR	CONSTANTE	3	0,5	0,75
	MAR	CONSTANTE	3	0,5	0,75
	DEI	FIM	2	0,5	0,5
BORDA_DESCIDA:	MAR	CONSTANTE	3	0,5	0,75
	MAR	CONSTANTE	3	0,5	0,75
FIM:	SUB		9	1,0	4,5
				Total:	**11,0**

A estimativa estática é mais rápida e utiliza o modelo genérico, ao passo que a simulação dinâmica utiliza a estimativa com modelo específico de processador, com as vantagens e desvantagens inerentes a cada método. Dada a facilidade de implementação e aos resultados suficientemente precisos para uma

primeira abordagem, este trabalho considera apenas o método de estimativa estática.

4.3.3 Estimativa estática do tamanho da memória de programa

O primeiro passo para se estimar estaticamente o tamanho da memória de programa necessário para uma dada aplicação é codificar e compilar o programa utilizando o conjunto de instruções genéricas. O tamanho, em *bytes*, de cada instrução genérica é especificado no "arquivo de tecnologia" [49] para cada processador de interesse. Baseado nestas informações, o tamanho da memória é calculado com sendo a soma em *bytes* de todas as instruções genéricas do programa. Então, se uma tarefa "B" for compilada para o conjunto de instruções genéricas "G" e, $instr\_size(g)$ representar o tamanho em *bytes* da instrução genérica "g", então o tamanho da memória requerido para esta tarefa pode ser dado por:

$$mem\_size(B) = \sum_{g \in G} instr\_size(g) \tag{4.7}$$

4.3.4 Análise de escalonamento

As aplicações de controle digital para acionamento de motores elétricos invariavelmente utilizam técnicas de sistemas de tempo real, onde várias tarefas do *software* concorrem pelo processador, tais como: execução das malhas de controle, disparo de transistores, TRIACs ou SCRs, sincronismo com a rede elétrica, medição de corrente e tensão, detecção e falhas, monitoração das realimentações e outras são executadas. Estas aplicações são caracterizadas por conterem tarefas que possuem restrições de tempo (prazos ou *deadlines*) para que tenham sua execução concluída, bem como, compartilham dos mesmos recursos computacionais. Tais aplicações não podem ter suas restrições temporais violadas sob o risco de causar mal funcionamento ou até mesmo acidentes dependendo da aplicação, o que permite classificá-las como sistemas de tempo real crítico.

O atendimento aos prazos das tarefas diz respeito ao modo com que os recursos do sistema são alocados em tempo de execução (durante a execução do

software), incluindo aí os recursos lógicos e físicos. Convencionalmente, a alocação de recursos é executada por algoritmos de escalonamento (*scheduler*) cujo propósito é intercalar as execuções das tarefas para que o sistema atinja seu objetivo pré-determinado. Assim, os algoritmos de escalonamento representam papel fundamental em sistemas de tempo real crítico.

De acordo com [50] os algoritmos de escalonamento de tempo real crítico podem ser divididos em duas classes: estáticos e dinâmicos. Os algoritmos estáticos determinam a seqüência de escalonamento das tarefas, atribuindo-lhes prioridades fixas antes que elas comecem a ser executadas (*off-line*). Tais algoritmos requerem antecipadamente um conhecimento completo das características das tarefas que fazem parte do sistema. Já os algoritmos dinâmicos determinam o escalonamento das tarefas conforme a evolução do sistema, atribuindo-lhes prioridades que podem mudar com o tempo. Comparando-se os dois tipos de algoritmos conclui-se que apesar dos estáticos imporem baixa sobrecarga computacional para o processador, eles são inflexíveis e não conseguem se adaptar às alterações do ambiente, sendo inapropriados para aplicações cujo comportamento não seja previsível. Por outro lado, os algoritmos do tipo dinâmico apresentam uma maior sobrecarga computacional devido às constantes mudanças de prioridades das tarefas, em compensação, fornecem maior flexibilidade ao sistema.

Diversos algoritmos de escalonamento para tarefas periódicas e aperiódicas têm sido desenvolvidos [54] a [57], entretanto todos se baseiam no algoritmo proposto por Liu e Layland [58] que usa o esquema de escalonamento da taxa monotônica, cujo princípio se baseia em um teste de escalonabilidade para um dado conjunto de tarefas periódicas.

O termo “taxa monotônica” (ou monótona) deriva de um método que estabelece prioridades a um conjunto de tarefas, assegurando que as prioridades sejam estabelecidas de forma monotônica, baseadas nos períodos das tarefas, tal que, quanto menor for o período maior será a prioridade.

Nas aplicações do capítulo 5 considerou se escalonadores estáticos, já que o comportamento de todas as tarefas periódicas podem ser previstos, de forma que suas prioridades podem ser estabelecidas monotonicamente. Este argumento

não vale para as tarefas aperiódicas, com é caso das tarefas de detecção de falhas, que são escalonadas via interrupção de *hardware*.

4.3.4.1 Análise da taxa monotônica

O esquema de escalonamento da taxa monotônica (Algoritmo da Taxa Monotônica, conhecido pela sigla RMA) desenvolvido por Lui e Layland [58] determina a escalonabilidade de um conjunto de tarefas periódicas e independentes, o qual assegura uma condição suficiente para garantir a não violação dos prazos do conjunto de tarefas que compõe o sistema. Este esquema se mostra ótimo entre os esquemas de escalonamento baseados em prioridades fixas ou estáticas [59]. Para cada uma das tarefas é estabelecida uma prioridade baseada no período da tarefa, tal que, quanto menor o período, maior a prioridade.

O escalonamento de tarefas para aplicações de tempo real crítico, baseado na teoria monotônica, tem sido bastante utilizado pela indústria e pela Academia, quer seja em sua forma original proposta em [58] ou derivações desta, tal como proposto em [57]. A referência [60] cita exemplos de aplicações do RMA tais como: um sistema de sonar dos submarinos da marinha americana; foi adotado pela NASA para aplicações na estação espacial *Freedom*; é utilizado pela Agência Espacial Européia em sistemas operacionais de tempo-real crítico; utilizado pela IBM; pela Universidade Carnegie Mellon, entre outras.

O algoritmo de escalonamento da taxa monotônica pode ser dividido em duas partes: a primeira decide se um conjunto de tarefas é escalonável (teste de escalonabilidade) e a segunda determina as prioridades estáticas baseadas nos períodos. Este algoritmo impõe algumas restrições ao sistema [54]: (a) o conjunto de tarefas é fixo; (b) todas as tarefas são periódicas; (c) todas as tarefas possuem prazo igual ao período; (d) o ambiente de processamento é centralizado (um único processador) e, (e) uma instância (execução) de uma tarefa deve ser concluída antes que a instância subseqüente seja escalonada pelo processador (as tarefas não são re-entrantes).

O teste de escalonabilidade baseado no RMA pode ser enunciado da seguinte forma: seja C_i o tempo de execução (computação) da *i-ésima* tarefa do

sistema e T_i o seu período. Define-se U_i como sendo a utilização da CPU pela *i-ésima* tarefa:

$$U_i = \frac{C_i}{T_i} \tag{4.8}$$

De acordo com [54] e [56] uma condição necessária mas não suficiente para que cada tarefa do sistema seja escalonavel, é dada por:

$$C_i \leq D_i \leq T_i \tag{4.9}$$

Sendo D_i o prazo (*deadline*) da *i-ésima* tarefa. De acordo com as restrições impostas acima, o prazo deve ser igual ao período, então a tarefa cumprirá seu prazo se $C_i \leq T_i$.

Define-se $U(n)$ como sendo o fator de utilização da CPU para um sistema composto por "n" tarefas e é dado por:

$$U(n) = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{4.10}$$

Liu e Layland determinaram um limite de utilização para o processador $LU(n)$, com sendo:

$$LU(n) = n \cdot \left(2^{\frac{1}{n}} - 1 \right) \tag{4.11}$$

A referência [58] estabelece que este limite converge para 69%, conforme o número de tarefas tende ao infinito, figura 4.6 e que se a inequação (4.12) for satisfeita, todas as tarefas do sistema cumprirão seus prazos. Esta inequação é conhecida como teste de escalonabilidade do algoritmo taxa monotônica.

$$U(n) \leq LU(n) \tag{4.12}$$

Embora o fator de utilização de 69% seja baixo, ele é pessimista e representa o pior caso possível. Através de análise estocástica [58] para um conjunto de tarefas periódicas geradas aleatoriamente e escalonadas pelo RMA, chegou-se à conclusão que a média do fator de utilização é muito melhor do que o pior caso.

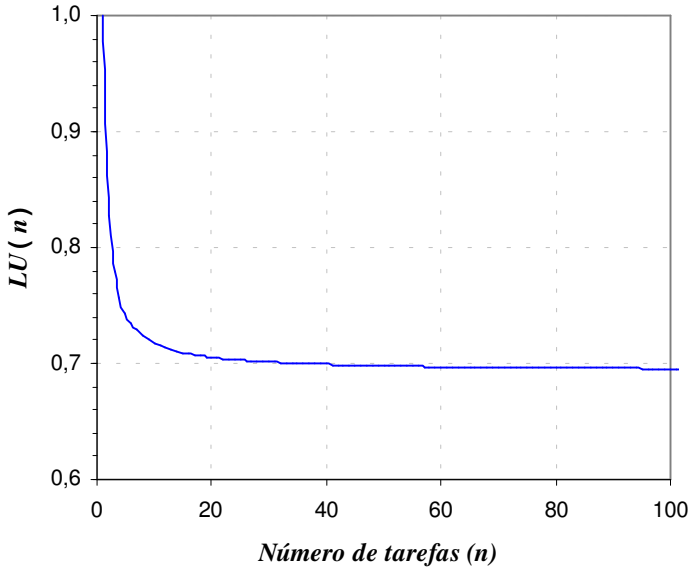


Figura 4.6 – LU x número de tarefas

Concluiu-se que uma boa aproximação seria 88%, com alguns sistemas atingindo até 99% [50] e [61], de tal forma que se pode dizer que, para um número elevado de tarefas, se $U(n) \leq 69\%$ com certeza o sistema será escalonável pelo RMA, se $U(n) > 100\%$ o sistema terá problemas de escalonamento, porém se $U(n)$ estiver entre 69% e 100% isso não significa que as tarefas não cumprirão seus respectivos prazos, nesta faixa o RMA é insuficiente para determinar a escalonabilidade e pode indicar falsamente que o sistema não é escalonável [54] e [62]. Para um número baixo de tarefas o limite de 69% pode ser superado, tal como indicado na figura 4.6, no entanto é importante ressaltar que o RMA estabelece uma condição suficiente mas não necessária de escalonamento, de tal forma que se $U(n) > LU(n)$ um método suficiente e necessário deve ser utilizado, tal como mencionado em [50] e [63] onde é apresentado um outro teste de escalonabilidade, denominado Teste Exato.

4.3.4.2 Teste Exato

O objetivo deste teste é determinar se uma tarefa irá concluir sua execução antes do seu prazo. Para tanto os pontos de escalonamento são analisados.

Seja um conjunto de n tarefas periódicas e independentes, organizadas em ordem de prioridade, irá cumprir todos os seus prazos se e somente se:

$$\min_{0<t<T_n}\left[\sum_{i=1}^{n}\left(\frac{C_i}{t}\right)\cdot\left\lceil\frac{t}{T_i}\right\rceil\right]\leq 1 \qquad (4.13)$$

Sendo t instantes de tempo múltiplos dos períodos das tarefas (chamados pontos de escalonamentos), até o valor do maior período do conjunto de tarefas (T_n) e $\lceil\ \rceil$ representa a aproximação pelo inteiro imediatamente superior.

Caso todas as tarefas pertencentes ao conjunto considerado atendam este requisito, então o conjunto será escalonável.

Para exemplificar este método considera-se o conjunto de tarefas caracterizadas abaixo:

Tabela 4.7 – Conjunto hipotético de tarefas

i	C_i [μs]	T_i [μs]	$U(i)$
1	40	100	0,4
2	40	150	0,267
3	100	350	0,286

No caso acima o fator de utilização total é 0,953, que é maior do que $3.(2^{1/3}-1) = 0{,}779$, portanto não é escalonável pelo RMA. Aplicando o Teste Exato tem-se:

Os pontos de escalonamentos são: 100, 150, 200, 300 e 350. Deve ser observado que os pontos de escalonamento 200 e 300 são múltiplos dos períodos 100 e 150 μs, portanto devem ser considerados. Então:

t =100 $\Rightarrow$ C1/100 ⌈100/100⌉ + C2/100 ⌈100/150⌉ + C3/100 ⌈100/350⌉ = 1,8

t =150 $\Rightarrow$ C1/150 ⌈150/100⌉ + C2/150 ⌈150/150⌉ + C3/150 ⌈150/350⌉ = 1,46

t =200 $\Rightarrow$ C1/200 ⌈200/100⌉ + C2/200 ⌈200/150⌉ + C3/200 ⌈200/350⌉ = 1,3

t =300 $\Rightarrow$ C1/300 ⌈300/100⌉ + C2/300 ⌈300/150⌉ + C3/300 ⌈300/350⌉ = 1,0

t =350 $\Rightarrow$ C1/350 ⌈350/100⌉ + C2/350 ⌈350/150⌉ + C3/350 ⌈350/350⌉ = 1,08

Como o valor mínimo dos resultados acima é igual a 1,0 o conjunto de tarefas satisfaz o Teste Exato e portanto é escalonável.

O Capítulo 5 utilizará as técnicas aqui descritas para simular o desempenho de diversos processadores quando submetidos ao pseudo-código obtido a partir da compilação genérica do código apresentado no Apêndice A.

Capítulo 5

AVALIAÇÃO DE DESEMPENHO DO PROCESSADOR

5.1 Introdução

Para ilustrar o emprego dos estimadores descritos no capítulo anterior foram realizadas duas avaliações estáticas de *software* considerando aplicações de controle de motor, utilizando moduladores em largura de pulso, através das técnicas escalar em malha aberta e em malha fechada baseada em vetores espaciais.

Para a modulação escalar, um protótipo experimental foi construído utilizando-se o MCU da Motorola MC68HC11A1, com o objetivo de comprovar a validade da avaliação e estabelecer seus erros. Essa avaliação foi baseada nos algoritmos apresentados no Capítulo 3 e no *firmware* do Apêndice A. Deve ser observado que apenas as tarefas periódicas foram avaliadas e consideradas no Apêndice A.

A técnica baseada em vetores espaciais não foi implementada em laboratório e foi baseada em algoritmos disponíveis na literatura tal com descrito em [64].

A partir destas aplicações foram desenvolvidos dois *softwares* padrões, os quais foram compilados com as instruções genéricas e sobre os quais foi aplicado o método descrito no item 4.3.1.3. Partindo-se dos modelos genéricos foram determinados: (a) o tempo de execução de cada tarefa, (b) o tamanho da memória de programa necessária para a aplicação e (c) a análise de escolamento das tarefas. Estas duas aplicações foram avaliadas com os dados dos processadores (arquivos de tecnologia) indicados na tabela 4.3.

Como será visto a seguir nem todos os processadores suportam as aplicações testadas, bem como alguns estarão super dimensionados. Este é o objetivo desta metodologia, ou seja, predizer qual processador é o mais adequado

a uma determinada aplicação (carga de *software*), uma vez que este trabalho se preocupa com soluções de baixo custo e geralmente, em eletrônica de consumo o processador é um dos componentes mais caros da aplicação, daí a preocupação em dimensioná-lo adequadamente.

Conforme indicado no Capítulo 4, este trabalho utilizou dados dos seguintes processadores: MC68HC11A1, MC68HC08MR4 e DSP56800 da Motorola (atualmente Freescale); ST92141 da STMicroelectronics; PIC17C752 da Microchip; COP8SGE7 da National Semiconductors e TMS320LC2402 da Texas Instruments. Para eles foram construídas as tabelas 5.1 a 5.14 baseadas no *firmware* do Apêndice A, usando um controle escalar em malha aberta. As tabelas 5.16 a 5.20 foram escritas para os processadores MC68HC08MR4, DSP56800, ST92141, PIC17C752 e TMS320LC2402, usando um controle baseado em vetores espaciais em malha fechada de velocidade.

5.2 Avaliação do controle escalar em malha aberta

Basicamente esta aplicação pode ser implementada por cinco tarefas periódicas e independentes, designadas por: (1) interface homem-máquina (IHM) e laço (*loop*) principal (leitura e interpretação das chaves de comando e escalonamento estático das tarefas); (2) controlador V/Hz com *voltage boost* (calcula a tensão a ser aplicada aos terminais da máquina a partir do comando de freqüência); (3) Duty-Cycle (calcula T_{ON} e T_{OFF} para as três fases do modulador); (4) Tempo Morto (insere o tempo morto nas três fases do modulador) e (5) PWM (controla os temporizadores – *timers* – do processador para gerar os sinais MLP). O diagrama de Gantt da figura 5.1 sugere o escalonamento da $k\text{-}ésima$ instância destas tarefas, onde P1(k) a P5(k) são os tempos de pronto (instante em que as tarefas estão prontas para serem executadas). Estas tarefas foram avaliadas para todos os processadores da tabela 4.3.

De acordo com o Algoritmo da Taxa Monotônica (RMA), o limite de utilização para este conjunto de tarefas é LU(5) = $5.(2^{1/5}-1)$ = 74,3%. Deve-se observar que os dispositivos Motorola MR4 e 56800, o Texas TMS320 e o ST92 possuem periféricos dedicados ao controle de motores, entretanto a redução de código proporcionada por estes periféricos não foi considerada nestas avaliações

por motivos de padronização (portabilidade) do *software* utilizado, ou seja, tais periféricos não foram considerados na análise. Desta forma pode-se esperar que os resultados para tais processadores possam ser melhorados.

Duas situações foram consideradas: baixas e altas freqüências de saída do inversor. Para tanto, de acordo com a tabela 3.3, dois períodos de chaveamento foram considerados: 397µs para 14Hz na saída e 556µs para 120Hz na saída. Estes períodos foram utilizados para avaliar a capacidade de escalonamento do conjunto de tarefas da aplicação em questão, para todos os processadores considerados na tabela 4.3. Após a conversão das rotinas do Apêndice A, utilizando-se as IG's e a compilação genérica (obtenção do número de *bytes* e número de ciclos de máquina), obteve-se os dados da tabela 5.1 para o HC11A1.

O teste de escalonabilidade baseado no RMA impõe que o fator de utilização da CPU deve ser menor ou igual ao limite de utilização da CPU, expressão (4.12), pode-se observar na tabela 5.2 que tanto para baixas, quanto para altas freqüências o processador MC68HC11A1 não atende ao RMA. No entanto, este resultado indica falsamente que o conjunto não é escalonável em toda a faixa de freqüência considerada. Aplicando-se o Teste Exato verifica-se que de fato para baixas freqüências de saída o conjunto de tarefas realmente não é escalonável, porém para altas ele o é, dai a importância de se verificar os resultados do RMA através do Teste Exato, uma vez que o RMA é pessimista.

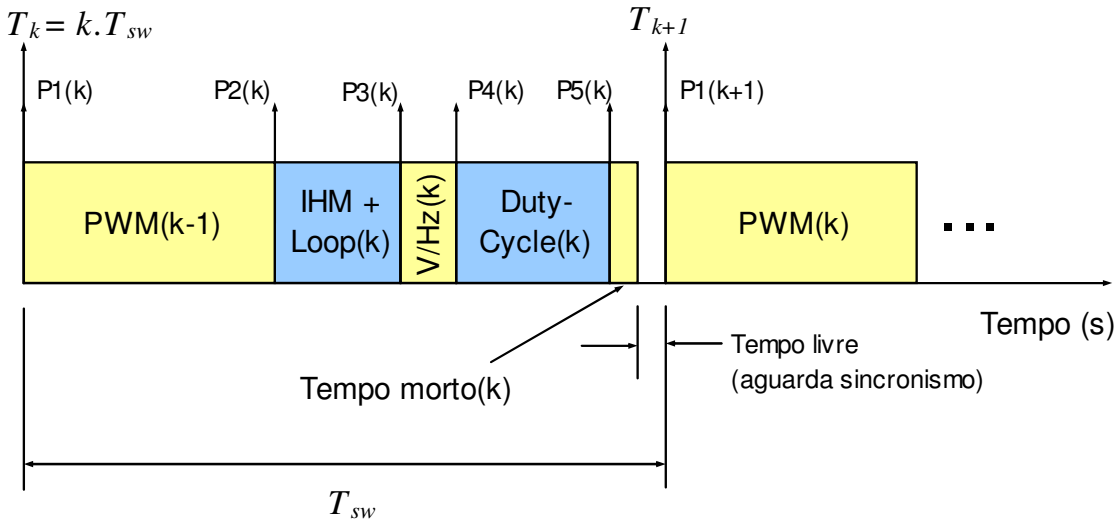


Figura 5.1 – Escalonamento das tarefas do controle escalar em malha aberta

Da figura 5.1 pode-se constatar que esta aplicação foi implementada de forma que as cinco tarefas devam ser executadas dentro de um período de chaveamento do MLP (T_{sw}), então todos os períodos das tarefas serão iguais, porém com diferentes tempos de pronto, P1(k) a P5(k). Formalmente, o RMA considera o pior caso no qual todas as tarefas estão prontas para iniciar o seu processamento no mesmo instante. No caso específico desta aplicação, a situação é mais branda, ou seja, nem todas as tarefas estão prontas no instante "zero", isso é um facilitador no momento da escolha de qual tarefa será escalonada pelo processador.

Tabela 5.1 – Tarefas do controle escalar em malha aberta para o Motorola HC11A1

Proc.: MOTOROLA HC11A1 / **TAREFAS**	**IG**	**QTDDE**	**BYTES**	**CICLOS**	**TOTAL BYTES**	**TOTAL CICLOS**	**TOTAL TEMPO (s)**
PWM (3 FASES)	ALR	19	2	4	38	76	3,8E-05
	MAR	51	2	3	102	153	7,7E-05
	DEC	3	2	3	6	9	4,5E-06
	DEI	0	2	2	0	0	0,0E+00
	SUB	1	1	9	1	9	4,5E-06
TEMPO MORTO	ALR	3	2	4	6	12	6,0E-06
	MAR	3	2	3	6	9	4,5E-06
	DEC	9	2	3	18	27	1,4E-05
	DEI	0	2	2	0	0	0,0E+00
	SUB	0	1	9	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	2	4	18	36	1,8E-05
	MAR	12	2	3	24	36	1,8E-05
	DEC	4	2	3	8	12	6,0E-06
	DEI	0	2	2	0	0	0,0E+00
	SUB	1	1	9	1	9	4,5E-06
DUTY CYCLE (3 FASES)	ALR	55	2	4	110	220	1,1E-04
	MAR	45	2	3	90	135	6,8E-05
	DEC	20	2	3	40	60	3,0E-05
	DEI	0	2	2	0	0	0,0E+00
	SUB	1	1	9	1	9	4,5E-06
IHM / LOOP PRINCIPAL	ALR	22	2	4	44	88	4,4E-05
	MAR	26	2	3	52	78	3,9E-05
	DEC	13	2	3	26	39	2,0E-05
	DEI	0	2	2	0	0	0,0E+00
	SUB	1	1	9	1	9	4,5E-06

PARCIAL	**BYTES**	**CICLOS**	**TEMPO (us)**
PWM (3 FASES)	147	247	123,50
TEMPO MORTO	30	48	24,00
CONTROLADOR V/HZ	51	93	46,50
DUTY CYCLE (3 FASES)	241	424	212,00
IHM / LOOP PRINCIPAL	123	214	107,00
TOTAL	**592**	**1026**	**513,00**

Tabela 5.2 – Fator de utilização para o controle escalar em malha aberta, para 3 fases de saída, Motorola HC11A1, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	397	123,5	0,31
TEMPO MORTO	397	24,0	0,06
CONTROLADOR V/HZ	397	46,5	0,12
DUTY CYCLE (3 FASES)	397	212,0	0,53
IHM / LOOP PRINCIPAL	397	107,0	0,27
Total		**513,0**	**1,29**

(a)

Controle escalar em malha aberta em altas freqüências	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	556	123,5	0,22
TEMPO MORTO	556	24,0	0,04
CONTROLADOR V/HZ	556	46,5	0,08
DUTY CYCLE (3 FASES)	556	212,0	0,38
IHM / LOOP PRINCIPAL	556	107,0	0,19
Total		**513,0**	**0,92**

(b)

Para o caso especial no qual todos os períodos são iguais, os pontos de escalonamento do Teste Exato resumem-se a $t = T_{sw}$, então de acordo com a expressão (4.13) pode-se verificar a escalonabilidade do conjunto de tarefas para altas e baixas freqüências, como se vê abaixo.

Para baixas freqüências de saída do inversor, t = 397 μs, então:

$C_1/397 \lceil 397/397 \rceil + C_2/397 \lceil 397/397 \rceil + C_3/397 \lceil 397/397 \rceil + C_4/397 \lceil 397/397 \rceil + C_5/397 \lceil 397/397 \rceil = 105,5/397 + 210,5/397 + 45/397 + 122/397 + 24/397 = 1,28$

Como o resultado acima é maior do que um, então pelo Teste Exato o conjunto de tarefas não é escalonável para o período considerado, coincidindo com o resultado do RMA.

Para altas freqüências de saída do inversor, t = 556 μs, então:

$C_1/556 \lceil 556/556 \rceil + C_2/556 \lceil 556/556 \rceil + C_3/556 \lceil 556/556 \rceil + C_4/556 \lceil 556/556 \rceil + C_5/556 \lceil 556/556 \rceil = 105,5/556 + 210,5/556 + 45/556 + 122/556 + 24/556 = 0,91$

Como o resultado acima é menor do que um, então pelo Teste Exato o conjunto de tarefas é escalonável para o período considerado, divergindo do resultado do RMA. Deve-se observar que, no caso especial no qual todos os períodos são iguais, para se aplicar o Teste Exato basta verificar se o fator de utilização da CPU (U) é menor ou igual a um, ou seja, o cálculo acima não precisaria ser feito, bastaria verificar a soma dos fatores de utilização da CPU nas tabelas 5.2 (a) e (b).

Tabela 5.3 – Tarefas do controle escalar em malha aberta para Motorola MR4

Proc.: MOTOROLA MR4 / TAREFAS	IG	QTDDE	BYTES	CICLOS	TOTAL BYTES	TOTAL CICLOS	TOTAL TEMPO (s)
PWM (3 FASES)	ALR	19	1	2	19	38	4,6E-06
	MAR	51	1	2	51	102	1,2E-05
	DEC	3	1	2	3	6	7,2E-07
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	10	1	10	1,2E-06
TEMPO MORTO	ALR	3	1	2	3	6	7,2E-07
	MAR	3	1	2	3	6	7,2E-07
	DEC	9	1	2	9	18	2,2E-06
	DEI	0	1	4	0	0	0,0E+00
	SUB	0	1	10	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	1	2	9	18	2,2E-06
	MAR	12	1	2	12	24	2,9E-06
	DEC	4	1	2	4	8	9,6E-07
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	10	1	10	1,2E-06
DUTY CYCLE (3 FASES)	ALR	55	1	2	55	110	1,3E-05
	MAR	45	1	2	45	90	1,1E-05
	DEC	20	1	2	20	40	4,8E-06
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	10	1	10	1,2E-06
IHM / LOOP PRINCIPAL	ALR	22	1	2	22	44	5,3E-06
	MAR	26	1	2	26	52	6,2E-06
	DEC	13	1	2	13	26	3,1E-06
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	10	1	10	1,2E-06

PARCIAL	BYTES	CICLOS	TEMPO (us)
PWM (3 FASES)	74	156	18,72
TEMPO MORTO	15	30	3,60
CONTROLADOR V/HZ	26	60	7,20
DUTY CYCLE (3 FASES)	121	250	30,00
IHM / LOOP PRINCIPAL	62	132	15,84
TOTAL	**298**	**628**	**75,36**

Tabela 5.4 – Fator de utilização para o controle escalar em malha aberta Motorola MR4, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: Motorola MR4		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	397	18,7	0,05
TEMPO MORTO	397	3,6	0,01
CONTROLADOR V/HZ	397	7,2	0,02
DUTY CYCLE (3 FASES)	397	30,0	0,08
IHM / LOOP PRINCIPAL	397	15,8	0,04
Total		**75,4**	**0,19**

(a)

Controle escalar em malha aberta em altas freqüências	Processador: Motorola MR4		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	556	18,7	0,03
TEMPO MORTO	556	3,6	0,01
CONTROLADOR V/HZ	556	7,2	0,01
DUTY CYCLE (3 FASES)	556	30,0	0,05
IHM / LOOP PRINCIPAL	556	15,8	0,03
Total		**75,4**	**0,14**

(b)

Tabela 5.5 – Tarefas do controle escalar em malha aberta para Motorola DSP56800

Proc.: MOTOROLA 56800 / TAREFAS	IG	QTDDE	BYTES	CICLOS	TOTAL BYTES	TOTAL CICLOS	TOTAL TEMPO (s)
PWM (3 FASES)	ALR	19	1	1	19	19	5,7E-07
	MAR	51	1	1	51	51	1,5E-06
	DEC	3	1	1	3	3	9,0E-08
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	6,0E-08
TEMPO MORTO	ALR	3	1	1	3	3	9,0E-08
	MAR	3	1	1	3	3	9,0E-08
	DEC	9	1	1	9	9	2,7E-07
	DEI	0	1	2	0	0	0,0E+00
	SUB	0	1	2	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	1	1	9	9	2,7E-07
	MAR	12	1	1	12	12	3,6E-07
	DEC	4	1	1	4	4	1,2E-07
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	6,0E-08
DUTY CYCLE (3 FASES)	ALR	55	1	1	55	55	1,7E-06
	MAR	45	1	1	45	45	1,4E-06
	DEC	20	1	1	20	20	6,0E-07
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	6,0E-08
IHM / LOOP PRINCIPAL	ALR	22	1	1	22	22	6,6E-07
	MAR	26	1	1	26	26	7,8E-07
	DEC	13	1	1	13	13	3,9E-07
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	6,0E-08

PARCIAL	BYTES	CICLOS	TEMPO (us)
PWM (3 FASES)	74	75	2,25
TEMPO MORTO	15	15	0,45
CONTROLADOR V/HZ	26	27	0,81
DUTY CYCLE (3 FASES)	121	122	3,66
IHM / LOOP PRINCIPAL	62	63	1,89
TOTAL	**298**	**302**	**9,06**

Tabela 5.6 – Fator de utilização para o controle escalar em malha aberta Motorola DSP56800, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: Motorola 56800		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	397	2,3	0,0057
TEMPO MORTO	397	0,5	0,0011
CONTROLADOR V/HZ	397	0,8	0,0020
DUTY CYCLE (3 FASES)	397	3,7	0,0092
IHM / LOOP PRINCIPAL	397	1,9	0,0048
Total		**9,1**	**0,0228**

(a)

Controle escalar em malha aberta em altas freqüências	Processador: Motorola 56800		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	556	2,3	0,0040
TEMPO MORTO	556	0,5	0,0008
CONTROLADOR V/HZ	556	0,8	0,0015
DUTY CYCLE (3 FASES)	556	3,7	0,0066
IHM / LOOP PRINCIPAL	556	1,9	0,0034
Total		**9,1**	**0,0163**

(b)

Tabela 5.7 – Tarefas do controle escalar em malha aberta para NSC COP8

Proc.: NSC COP8 / TAREFAS	IG	QTDDE	BYTES	CICLOS	TOTAL BYTES	TOTAL CICLOS	TOTAL TEMPO (s)
PWM (3 FASES)	ALR	19	3	4	57	76	7,6E-05
	MAR	51	2	3	102	153	1,5E-04
	DEC	3	2	3	6	9	9,0E-06
	DEI	0	2	3	0	0	0,0E+00
	SUB	1	1	12	1	12	1,2E-05
TEMPO MORTO	ALR	3	3	4	9	12	1,2E-05
	MAR	3	2	3	6	9	9,0E-06
	DEC	9	2	3	18	27	2,7E-05
	DEI	0	2	3	0	0	0,0E+00
	SUB	0	1	12	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	3	4	27	36	3,6E-05
	MAR	12	2	3	24	36	3,6E-05
	DEC	4	2	3	8	12	1,2E-05
	DEI	0	2	3	0	0	0,0E+00
	SUB	1	1	12	1	12	1,2E-05
DUTY CYCLE (3 FASES)	ALR	55	3	4	165	220	2,2E-04
	MAR	45	2	3	90	135	1,4E-04
	DEC	20	2	3	40	60	6,0E-05
	DEI	0	2	3	0	0	0,0E+00
	SUB	1	1	12	1	12	1,2E-05
IHM / LOOP PRINCIPAL	ALR	22	3	4	66	88	8,8E-05
	MAR	26	2	3	52	78	7,8E-05
	DEC	13	2	3	26	39	3,9E-05
	DEI	0	2	3	0	0	0,0E+00
	SUB	1	1	12	1	12	1,2E-05

PARCIAL	BYTES	CICLOS	TEMPO (us)
PWM (3 FASES)	166	250	250,00
TEMPO MORTO	33	48	48,00
CONTROLADOR V/HZ	60	96	96,00
DUTY CYCLE (3 FASES)	296	427	427,00
IHM / LOOP PRINCIPAL	145	217	217,00
TOTAL	**700**	**1038**	**1038,00**

Tabela 5.8 – Fator de utilização para o controle escalar em malha aberta NSC COP8, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: COP8		
	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
Tarefas			
PWM (3 FASES)	397	250,0	0,63
TEMPO MORTO	397	48,0	0,12
CONTROLADOR V/HZ	397	96,0	0,24
DUTY CYCLE (3 FASES)	397	427,0	1,08
IHM / LOOP PRINCIPAL	397	217,0	0,55
Total		**1038,0**	**2,61**

(a)

Controle escalar em malha aberta em altas freqüências	Processador: COP8		
	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
Tarefas			
PWM (3 FASES)	556	250,0	0,45
TEMPO MORTO	556	48,0	0,09
CONTROLADOR V/HZ	556	96,0	0,17
DUTY CYCLE (3 FASES)	556	427,0	0,77
IHM / LOOP PRINCIPAL	556	217,0	0,39
Total		**1038,0**	**1,87**

(b)

Tabela 5.9 – Tarefas do controle escalar em malha aberta para Texas TMS320C24

Proc.: TEXAS TMS320C TAREFAS	IG	QTDDE	BYTES	CICLOS	TOTAL BYTES	TOTAL CICLOS	TOTAL TEMPO (s)
PWM (3 FASES)	ALR	19	1	1	19	19	9,5E-07
	MAR	51	1	1	51	51	2,6E-06
	DEC	3	1	1	3	3	1,5E-07
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	4	1	4	2,0E-07
TEMPO MORTO	ALR	3	1	1	3	3	1,5E-07
	MAR	3	1	1	3	3	1,5E-07
	DEC	9	1	1	9	9	4,5E-07
	DEI	0	1	4	0	0	0,0E+00
	SUB	0	1	4	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	1	1	9	9	4,5E-07
	MAR	12	1	1	12	12	6,0E-07
	DEC	4	1	1	4	4	2,0E-07
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	4	1	4	2,0E-07
DUTY CYCLE (3 FASES)	ALR	55	1	1	55	55	2,8E-06
	MAR	45	1	1	45	45	2,3E-06
	DEC	20	1	1	20	20	1,0E-06
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	4	1	4	2,0E-07
IHM / LOOP PRINCIPAL	ALR	22	1	1	22	22	1,1E-06
	MAR	26	1	1	26	26	1,3E-06
	DEC	13	1	1	13	13	6,5E-07
	DEI	0	1	4	0	0	0,0E+00
	SUB	1	1	4	1	4	2,0E-07

PARCIAL	BYTES	CICLOS	TEMPO (us)
PWM (3 FASES)	74	77	3,85
TEMPO MORTO	15	15	0,75
CONTROLADOR V/HZ	26	29	1,45
DUTY CYCLE (3 FASES)	121	124	6,20
IHM / LOOP PRINCIPAL	62	65	3,25
TOTAL	**298**	**310**	**15,50**

Tabela 5.10 – Fator de utilização para o controle
escalar em malha aberta TMS320C, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: TMS320C		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	397	3,9	0,0097
TEMPO MORTO	397	0,8	0,0019
CONTROLADOR V/HZ	397	1,5	0,0037
DUTY CYCLE (3 FASES)	397	6,2	0,0156
IHM / LOOP PRINCIPAL	397	3,3	0,0082
Total		**15,5**	**0,0390**

(a)

Controle escalar em malha aberta em baixas freqüências	Processador: TMS320C		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	556	3,9	0,0069
TEMPO MORTO	556	0,8	0,0013
CONTROLADOR V/HZ	556	1,5	0,0026
DUTY CYCLE (3 FASES)	556	6,2	0,0112
IHM / LOOP PRINCIPAL	556	3,3	0,0058
Total		**15,5**	**0,0279**

(b)

Tabela 5.11 – Tarefas do controle escalar em malha aberta para ST92141

Proc.: ST92141 TAREFAS	IG	QTDDE	BYTES	CICLOS	TOTAL BYTES	TOTAL CICLOS	TOTAL TEMPO (s)
PWM (3 FASES)	ALR	19	3	4	57	76	1,2E-05
	MAR	51	2	3	102	153	2,4E-05
	DEC	3	3	4	9	12	1,9E-06
	DEI	0	3	4	0	0	0,0E+00
	SUB	1	1	5	1	5	8,0E-07
TEMPO MORTO	ALR	3	3	4	9	12	1,9E-06
	MAR	3	2	3	6	9	1,4E-06
	DEC	9	3	4	27	36	5,8E-06
	DEI	0	3	4	0	0	0,0E+00
	SUB	0	1	5	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	3	4	27	36	5,8E-06
	MAR	12	2	3	24	36	5,8E-06
	DEC	4	3	4	12	16	2,6E-06
	DEI	0	3	4	0	0	0,0E+00
	SUB	1	1	5	1	5	8,0E-07
DUTY CYCLE (3 FASES)	ALR	55	3	4	165	220	3,5E-05
	MAR	45	2	3	90	135	2,2E-05
	DEC	20	3	4	60	80	1,3E-05
	DEI	0	3	4	0	0	0,0E+00
	SUB	1	1	5	1	5	8,0E-07
IHM / LOOP PRINCIPAL	ALR	22	3	4	66	88	1,4E-05
	MAR	26	2	3	52	78	1,2E-05
	DEC	13	3	4	39	52	8,3E-06
	DEI	0	3	4	0	0	0,0E+00
	SUB	1	1	5	1	5	8,0E-07

PARCIAL	BYTES	CICLOS	TEMPO (us)
PWM (3 FASES)	169	246	39,36
TEMPO MORTO	42	57	9,12
CONTROLADOR V/HZ	64	93	14,88
DUTY CYCLE (3 FASES)	316	440	70,40
IHM / LOOP PRINCIPAL	158	223	35,68
TOTAL	**749**	**1059**	**169,44**

Tabela 5.12 – Fator de utilização para o controle escalar em malha aberta ST92141, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: ST92141		
	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
Tarefas			
PWM (3 FASES)	397	39,4	0,10
TEMPO MORTO	397	9,1	0,02
CONTROLADOR V/HZ	397	14,9	0,04
DUTY CYCLE (3 FASES)	397	70,4	0,18
IHM / LOOP PRINCIPAL	397	35,7	0,09
Total		**169,4**	**0,43**

(a)

Controle escalar em malha aberta em baixas freqüências	Processador: ST92141		
	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
Tarefas			
PWM (3 FASES)	556	39,4	0,07
TEMPO MORTO	556	9,1	0,02
CONTROLADOR V/HZ	556	14,9	0,03
DUTY CYCLE (3 FASES)	556	70,4	0,13
IHM / LOOP PRINCIPAL	556	35,7	0,06
Total		**169,4**	**0,30**

(b)

Tabela 5.13 – Tarefas do controle escalar em malha aberta para PIC17C752

Proc.: PIC 17C572 TAREFAS	IG	QTDDE	BYTES	CICLOS	TOTAL BYTES	TOTAL CICLOS	TOTAL TEMPO (s)
PWM (3 FASES)	ALR	19	1	1	19	19	2,3E-06
	MAR	51	1	1	51	51	6,1E-06
	DEC	3	1	1	3	3	3,6E-07
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	2,4E-07
TEMPO MORTO	ALR	3	1	1	3	3	3,6E-07
	MAR	3	1	1	3	3	3,6E-07
	DEC	9	1	1	9	9	1,1E-06
	DEI	0	1	2	0	0	0,0E+00
	SUB	0	1	2	0	0	0,0E+00
CONTROLADOR V/HZ	ALR	9	1	1	9	9	1,1E-06
	MAR	12	1	1	12	12	1,4E-06
	DEC	4	1	1	4	4	4,8E-07
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	2,4E-07
DUTY CYCLE (3 FASES)	ALR	55	1	1	55	55	6,6E-06
	MAR	45	1	1	45	45	5,4E-06
	DEC	20	1	1	20	20	2,4E-06
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	2,4E-07
IHM / LOOP PRINCIPAL	ALR	22	1	1	22	22	2,6E-06
	MAR	26	1	1	26	26	3,1E-06
	DEC	13	1	1	13	13	1,6E-06
	DEI	0	1	2	0	0	0,0E+00
	SUB	1	1	2	1	2	2,4E-07

PARCIAL	BYTES	CICLOS	TEMPO (us)
PWM (3 FASES)	74	75	9,00
TEMPO MORTO	15	15	1,80
CONTROLADOR V/HZ	26	27	3,24
DUTY CYCLE (3 FASES)	121	122	14,64
IHM / LOOP PRINCIPAL	62	63	7,56
TOTAL	**298**	**302**	**36,24**

Tabela 5.14 – Fator de utilização para o controle
escalar em malha aberta PIC 17C572, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências	Processador: PIC 17C572		
	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
Tarefas			
PWM (3 FASES)	397	9,0	0,0227
TEMPO MORTO	397	1,8	0,0045
CONTROLADOR V/HZ	397	3,2	0,0082
DUTY CYCLE (3 FASES)	397	14,6	0,0369
IHM / LOOP PRINCIPAL	397	7,6	0,0190
Total		**36,2**	**0,0913**

(a)

Controle escalar em malha aberta em baixas freqüências	Processador: PIC 17C572		
	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
Tarefas			
PWM (3 FASES)	556	9,0	0,0162
TEMPO MORTO	556	1,8	0,0032
CONTROLADOR V/HZ	556	3,2	0,0058
DUTY CYCLE (3 FASES)	556	14,6	0,0263
IHM / LOOP PRINCIPAL	556	7,6	0,0136
Total		**36,2**	**0,0652**

(b)

Os gráficos abaixo mostram os resultados obtidos para os diversos processadores considerados nas tabelas anteriores.

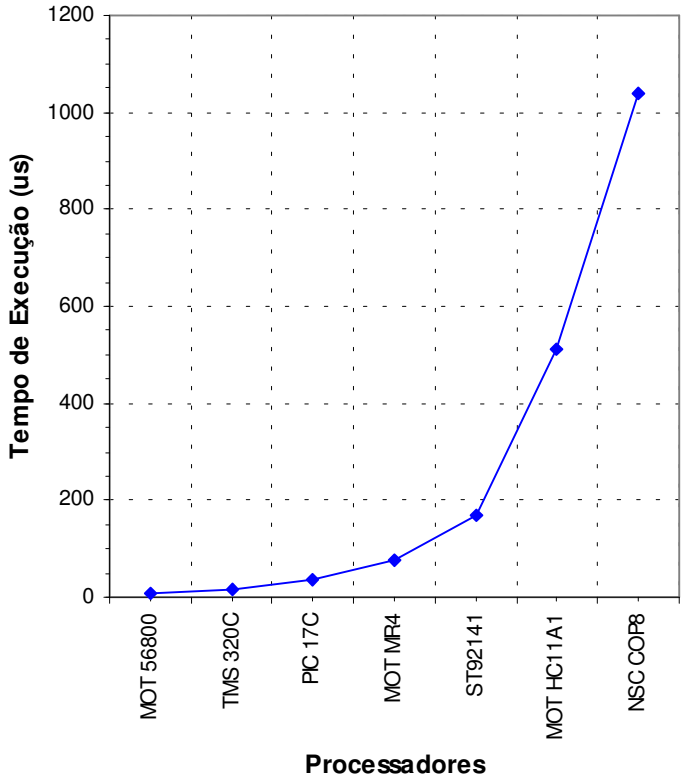


Figura 5.2 – Avaliação do tempo de execução de cada processador, para a execução do controle escalar em malha aberta

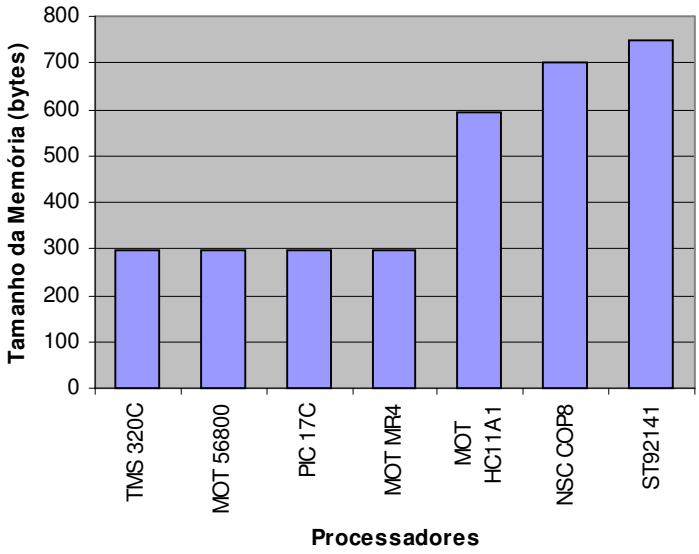


Figura 5.3 – Avaliação do tamanho da memória de programa necessária para implementar o controle escalar em malha aberta, para cada processador

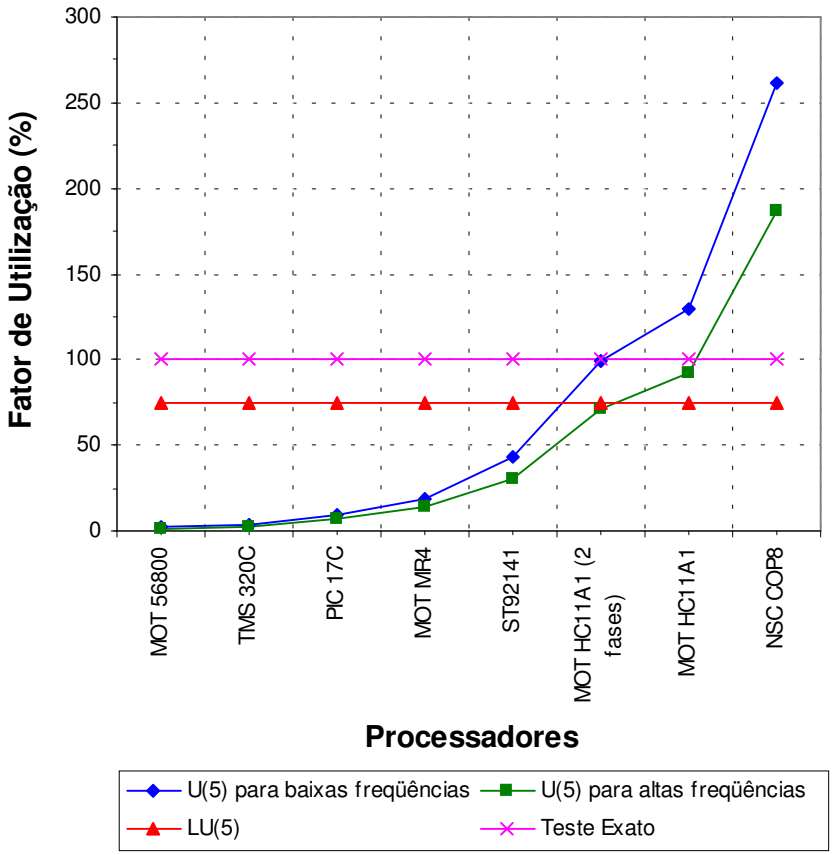


Figura 5.4 – Avaliação do fator de utilização de cada processador, para a execução do controle escalar em malha aberta

Os processadores 56800, TMS320, PIC17C, MOT MR4 e ST92 podem, de acordo com a avaliação, ser utilizados nesta aplicação pois fator de utilização da CPU é inferior ao limite imposto pelo RMA. O Motorola MC68HC11A1 pode ser utilizado para sintetizar altas freqüências nas três fases de saída do inversor, como garante a análise pelo Teste Exato. Já o MCU COP8 não pode ser utilizado nem em baixas nem em altas freqüências, pois o seu fator de utilização excede até mesmo o limite imposto pelo Teste Exato que é de 100%.

Como não foi possível executar o *software* de controle escalar com suas 5 tarefas periódicas, no MCU MC68HC11A1, para toda a faixa de freqüência de saída do inversor (3 Hz a 120 Hz), eliminou-se o processamento das rotinas de uma das fases do motor. Isso significa que o tempo necessário para executar as tarefas “PWM”, “Tempo Morto” e “Duty Cycle” foram reduzidos em um terço. Desta forma o acionamento trifásico tornou-se um acionamento “bifásico”. O fator

de utilização da CPU para executar este novo *software* está indicado na figura 5.4 por “MOT HC11A1 (2 fases)”. Agora o referido processador conseguiu escalonar todas as tarefas, tanto em baixas freqüências pelo Teste Exato, quanto em altas pelo RMA. A tabela 5.15 fornece os dados comparativos.

A figura 5.4 mostra um detalhe bastante importante deste método que é a visualização gráfica do fator de utilização dos diferentes processadores avaliados. A partir deste gráfico pode-se escolher o melhor processador para a aplicação. Os dispositivos que não atendem ao Teste Exato devem ser prontamente descartados. Os processadores que mostram um fator de utilização muito baixo, não são uma escolha sensata, pois estão super dimensionados para a aplicação. A melhor escolha fica por conta dos dispositivos marginais, ou seja aqueles que estão próximos dos limites impostos pelo RMA ou Teste Exato. No caso da figura 5.4, uma boa escolha para o início do projeto poderia ser o ST92141.

Tabela 5.15 – Fator de utilização para o controle escalar em malha aberta, para duas fases de saída, MOT HC11A1 (a) baixas e (b) altas freqüências

Controle ecalar em malha aberta em baixas freqüências, duas fases de saída	**Processador: Motorola A1**		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	397	82,3	0,21
TEMPO MORTO	397	16,0	0,04
CONTROLADOR V/HZ	397	46,5	0,12
DUTY CYCLE (2 FASES)	397	141,3	0,36
IHM / LOOP PRINCIPAL	397	107,0	0,27
Total		**393,1**	**0,99**

(a)

Controle ecalar em malha aberta em altas freqüências, duas fases de saída	**Processador: Motorola A1**		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	556	82,3	0,15
TEMPO MORTO	556	16,0	0,03
CONTROLADOR V/HZ	556	46,5	0,08
DUTY CYCLE (2 FASES)	556	141,3	0,25
IHM / LOOP PRINCIPAL	556	107,0	0,19
Total		**393,1**	**0,71**

(b)

Como será visto no Capítulo 6 o resultado desta avaliação foi comprovado experimentalmente, uma vez que é possível sintetizar os sinais MLP para alimentar as três fases do motor apenas para altas freqüências de saída e em baixas apenas para duas delas. Isso se deve ao fato de que para a sintetização das baixas freqüências exige-se um número maior de iterações para se obter um ciclo da senóide, ver figura 3.20, sendo comum a elevação da freqüência de chaveamento para executar mais rapidamente estas iterações, tal como mostrado na figura 3.26.

5.3 Avaliação do controle escalar em malha fechada

A estratégia de controle escalar em malha fechada baseada em vetores espaciais não foi implementada no protótipo experimental, uma vez que o processador utilizado não suportou nem mesmo a técnica em malha aberta, entretanto ela foi avaliada para fins de comparação. De forma bastante simplificada esta aplicação pode ser implementada através de sete tarefas periódicas de acordo com [64]. Esta referência é um *Application Note* da Texas Instruments que apresenta uma codificação para tal técnica de controle, usando modulação em largura de pulso por vetores espaciais representada na figura 5.5. Apenas uma situação foi considerada nesta avaliação: tempo de malha (*loop*) de 41,6 μs (período das tarefas), o que equivale a uma taxa de varredura do programa de 24 kHz. As tabelas 5.16 a 5.20 resumem a avaliação desta aplicação para alguns dos processadores da tabela 4.3.

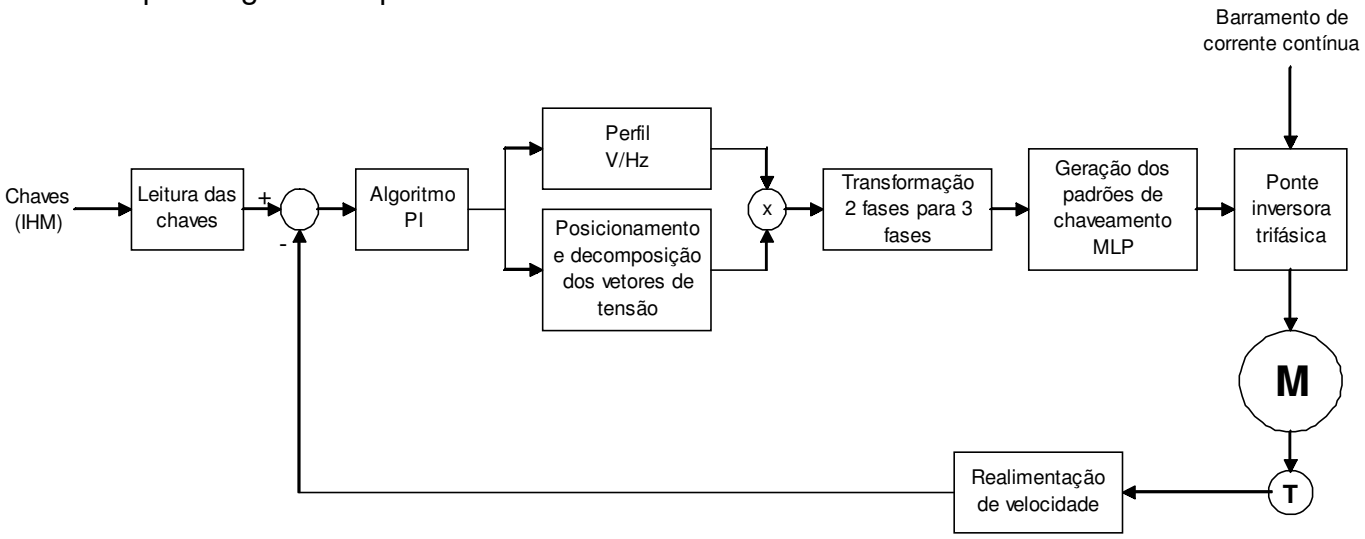


Figura 5.5 – Diagrama de blocos do controle proposto em [64]

Tabela 5.16 - Tarefas do controle escalar em malha fechada para TMS320C

Processador: TMS320 Controle escalar malha fechada **Tarefas**	Período da Tarefa (us)	Número Total de Bytes	Número Total de Ciclos	Tempo de Execução (us)	Fator de Utilização da CPU
Leitura e tratamento das chaves	41,6	28	32	1,6	0,04
Algorítmo PI	41,6	26	28	1,4	0,03
Perfil V/Hz	41,6	24	24	1,2	0,03
Pos. e dec. dos vetores de tensão	41,6	29	33	1,65	0,04
Transformação 2/3 fases	41,6	49	53	2,65	0,06
Geração dos padrões PWM	41,6	49	43	2,15	0,05
Realimentação de velocidade	41,6	98	114	5,7	0,14
Total		**303**	**327**	**16,35**	**0,39**

Tabela 5.17 - Tarefas do controle escalar em malha fechada para Motorola MR4

Processador: MOTOROLA MR4	Período	Número	Número	Tempo de	Fator de
Controle escalar malha fechada	da Tarefa	Total de	Total de	Execução	Utilização
Tarefas	(us)	Bytes	Ciclos	(us)	da CPU
Leitura e tratamento das chaves	41,6	58	67	8	0,19
Algorítmo PI	41,6	54	59	7	0,17
Perfil V/Hz	41,6	50	50	6	0,14
Pos. e dec. dos vetores de tensão	41,6	60	69	8,25	0,20
Transformação 2/3 fases	41,6	101	111	13,25	0,32
Geração dos padrões PWM	41,6	101	90	10,75	0,26
Realimentação de velocidade	41,6	203	239	28,5	0,69
Total		**627**	**685**	**81,75**	**1,97**

Tabela 5.18 - Tarefas do controle escalar em malha fechada para Motorola DSP56800

Processador: MOTOROLA 56800	Período	Número	Número	Tempo de	Fator de
Controle escalar malha fechada	da Tarefa	Total de	Total de	Execução	Utilização
Tarefas	(us)	Bytes	Ciclos	(us)	da CPU
Leitura e tratamento das chaves	41,6	25	54	1,6	0,04
Algorítmo PI	41,6	23	47	1,4	0,03
Perfil V/Hz	41,6	21	40	1,2	0,03
Pos. e dec. dos vetores de tensão	41,6	26	55	1,65	0,04
Transformação 2/3 fases	41,6	43	89	2,65	0,06
Geração dos padrões PWM	41,6	43	72	2,15	0,05
Realimentação de velocidade	41,6	87	191	5,7	0,14
Total		**268**	**548**	**16,35**	**0,39**

Tabela 5.19 - Tarefas do controle escalar em malha fechada para ST92141

Processador: ST92141	Período	Número	Número	Tempo de	Fator de
Controle escalar malha fechada	da Tarefa	Total de	Total de	Execução	Utilização
Tarefas	(us)	Bytes	Ciclos	(us)	da CPU
Leitura e tratamento das chaves	41,6	55	258	41,33	0,99
Algorítmo PI	41,6	51	226	36,16	0,87
Perfil V/Hz	41,6	47	194	31,00	0,75
Pos. e dec. dos vetores de tensão	41,6	57	266	42,62	1,02
Transformação 2/3 fases	41,6	96	428	68,45	1,65
Geração dos padrões PWM	41,6	96	347	55,53	1,33
Realimentação de velocidade	41,6	193	920	147,23	3,54
Total		**595**	**2639**	**422,32**	**10,15**

Tabela 5.20 - Tarefas do controle escalar em malha fechada para PIC 17C752

Processador: PIC 17C752 Controle escalar malha fechada **Tarefas**	Período da Tarefa (us)	Número Total de Bytes	Número Total de Ciclos	Tempo de Execução (us)	Fator de Utilização da CPU
Leitura e tratamento das chaves	41,6	25	28	3,35	0,08
Algorítmo PI	41,6	23	24	2,93	0,07
Perfil V/Hz	41,6	21	21	2,51	0,06
Pos. e dec. dos vetores de tensão	41,6	26	29	3,46	0,08
Transformação 2/3 fases	41,6	44	46	5,55	0,13
Geração dos padrões PWM	41,6	44	37	4,50	0,11
Realimentação de velocidade	41,6	88	99	11,94	0,29
Total		**271**	**284**	**34,24**	**0,82**

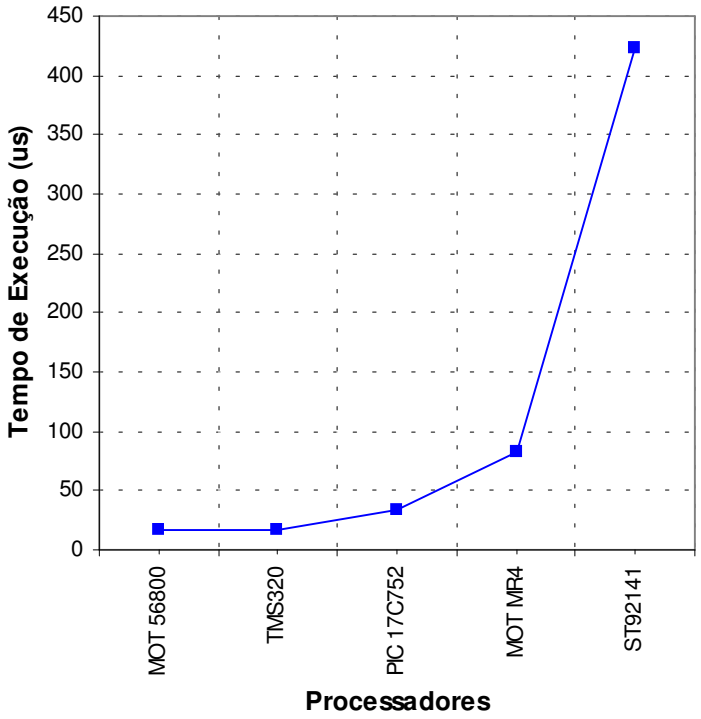


Figura 5.6 – Avaliação do tempo de execução de cada processador, para a execução do controle escalar em malha fechada

De acordo com o RMA o limite de utilização para este conjunto de tarefas é $LU(7) = 7.(2^{1/7}-1)$. 100% = 72,8%. A figura 5.8 mostra a avaliação dos fatores de utilização de cada processador para a aplicação escalar em malha fechada, bem como os limites impostos pelo RMA e Teste Exato.

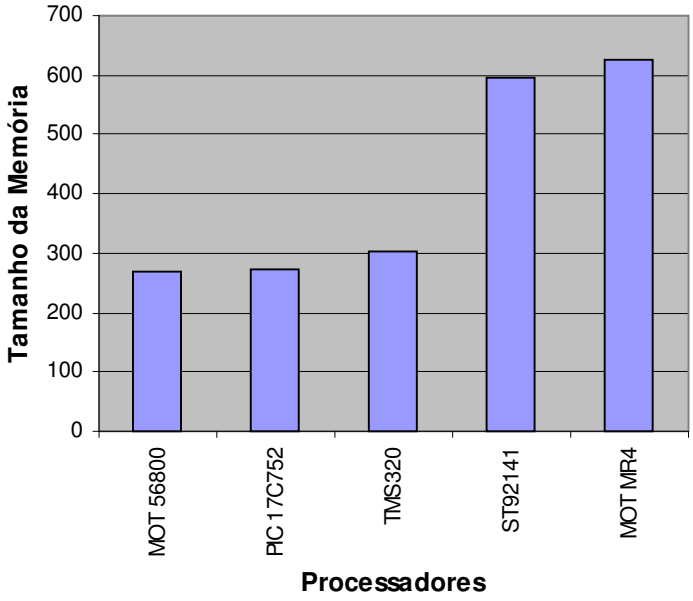


Figura 5.7 – Avaliação do tamanho da memória de programa necessária para implementar o controle escalar em malha fechada, para cada processador

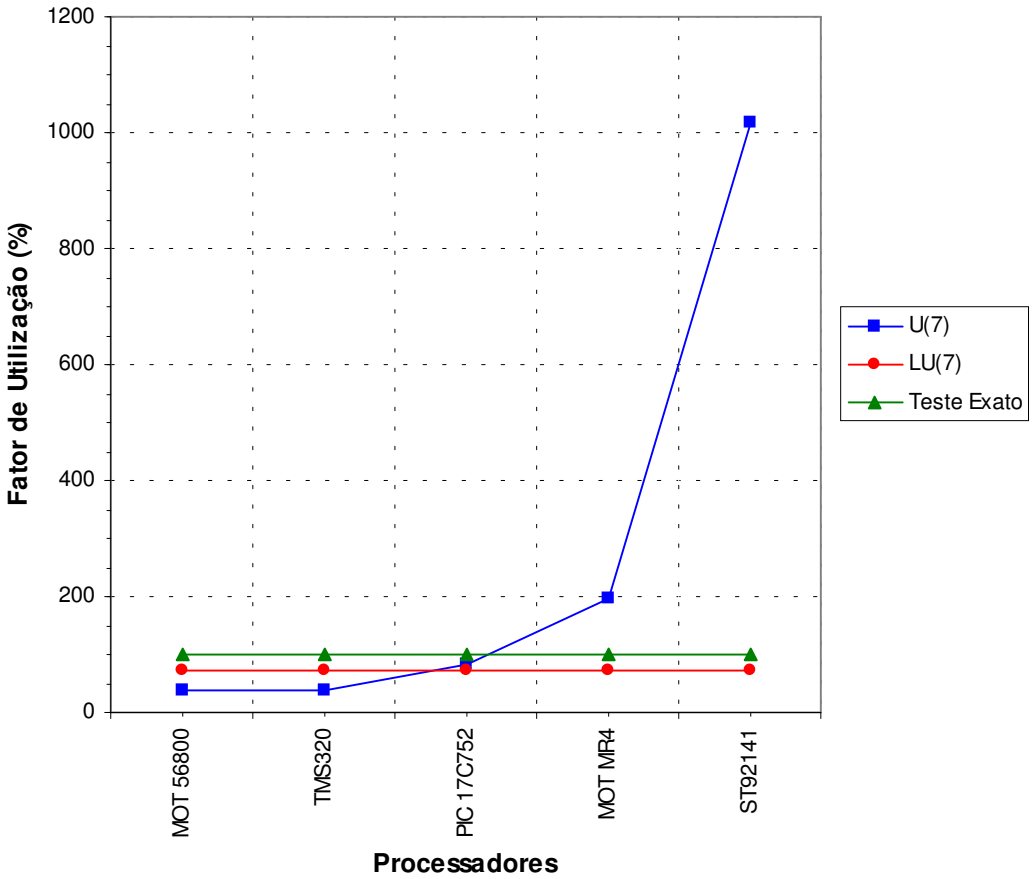


Figura 5.8 – Avaliação do fator de utilização de cada processador, para a execução do controle escalar em malha fechada

Observa-se que para esta aplicação apenas os DSP´s da Texas e da Motorola, bem como o PIC da Microchip, que é um processador tipo RISC (*Reduced Instruction Set Computer*) e portanto, rápido, podem ser utilizados, pois são os únicos dentre os analisados que atendem ao RMA e, ao Teste Exato para o caso do PIC, o qual tem sido citado na literatura [65] e [66] como opção de baixo custo para este tipo de aplicação. Este resultado já era esperado uma vez que estes dispositivos possuem elevada capacidade computacional. Conclui-se com isso que o uso do controle escalar em malha fechada baseado em vetores espaciais, apesar de possuir desempenho dinâmico superior ao controle escalar em malha aberta, ainda torna-se difícil de ser utilizado em aplicações de consumo devido ao elevado custo do processador necessário a sua implementação.

O capítulo 6 apresentará os resultados experimentais, comparando-os com os dados das avaliações.

Capítulo 6

RESULTADOS EXPERIMENTAIS

6.1 Introdução

Implementações de *software* básicos destinadas a aplicações de controle geralmente são otimizadas de acordo com as duas premissas: (a) desempenho - que normalmente está relacionado a um baixo tempo de resposta da malha de controle e a precisão de atuação no processo ou, (b) tamanho da memória necessária para implementar a aplicação - que diz respeito ao tamanho, geralmente em *bytes*, do programa. O ideal seria obter um mesmo código otimizado de acordo com as duas premissas, porém isso nem sempre é possível e por vezes elas são mutuamente exclusivas.

Para o desenvolvimento que se mostrou no Capítulo 3 optou-se por implementar o método da amostragem regular simétrica e, por motivos de capacidade computacional do microcontrolador utilizado, buscou-se um método com menor demanda de tempo de processamento, ou seja, o *firmware* foi otimizado em desempenho. O *hardware* do protótipo foi detalhado no Capítulo 3 e no Apêndice A está descrito o *firmware* desenvolvido para esta aplicação.

Neste capítulo será feita uma comparação entre os resultados obtidos nas avaliações do Capítulo 5 e os resultados experimentais obtidos no protótipo.

6.2 Modulação regular simétrica

Tal como descrito em 3.5.3, a modulação regular simétrica foi a técnica escolhida para a geração dos sinais MLP, cujas formas de ondas a seguir ilustram a execução das tarefas do *firmware*. Na figura 6.1, no canal 2 do osciloscópio (traço superior) está representado o sinal MLP para uma das fases do motor e no canal 1 (traço inferior), está representado o sinal senoidal que serve como referência para a sintetização do MLP, o qual foi obtido através da saída analógica de teste do *hardware* implementado, tal como ilustrado na figura 3.40. Deve ser

observado que este sinal coincide com o especificado na tabela 3.3, ou seja, para gerar uma freqüência de 60 Hz na saída do inversor utiliza-se o índice de modulação em freqüência (m_f) igual a 40 e uma freqüência de chaveamento (f_{sw}) igual a 2400 Hz, no caso prático utilizou-se 2440 Hz.

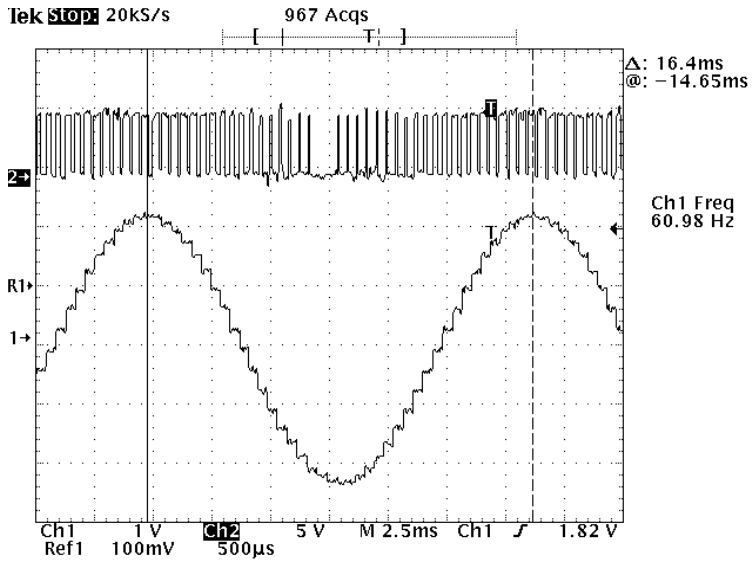


Figura 6.1 – Modulação regular simétrica para f_s = 60 Hz e m_f = 40

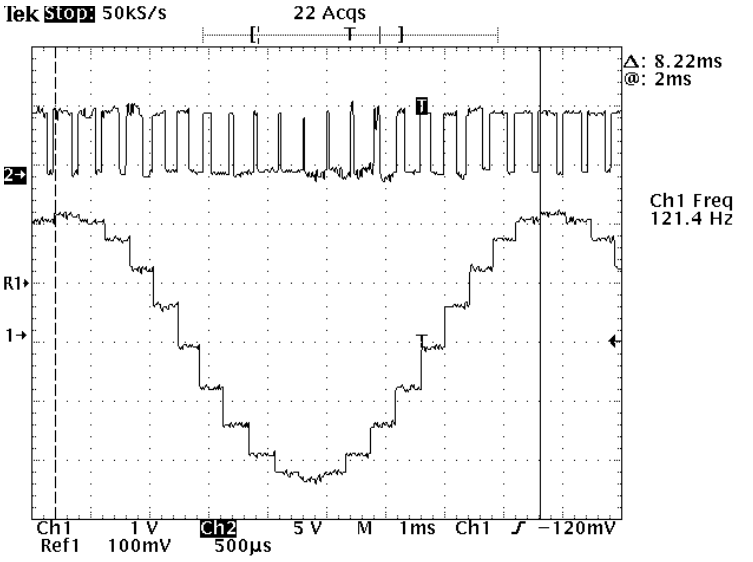


Figura 6.2 – Modulação regular simétrica para f_s = 121 Hz e m_f = 20

De forma semelhante à figura 6.1, a figura 6.2 mostra a modulação regular simétrica para uma freqüência de saída de aproximadamente 120 Hz, operando em uma freqüência de chaveamento de 2000 Hz.

Observa-se que existe uma pequena defasagem entre o sinal MLP e a senóide de referência, pois como a tarefa de *software* prioritária é a geração da MLP, esta é escalonada primeiro e na seqüência é escalonada a tarefa de controle do CDA. Este detalhe é evidenciado na figura 6.3, onde é mostrada uma ampliação da figura 6.1 utilizando-se uma base de tempo diferente. Ainda na figura 6.3, o intervalo de tempo entre os dois marcadores do osciloscópio (linhas verticais) é o período de chaveamento (T_{sw}), tempo durante o qual são temporizados os instantes de chave ligada (T_{ON}) e desligada (T_{OFF}), bem como são executadas todas as tarefas indicadas na tabela 5.1.

A faixa de variação obtida para o período de chaveamento do MLP foi de 410 µs a 900 µs o que equivale a uma variação na freqüência de chaveamento de 2,4 kHz a 1,1 kHz, respectivamente, valores estes muito próximos daqueles previstos na tabela 3.3 e representados graficamente na figura 3.27.

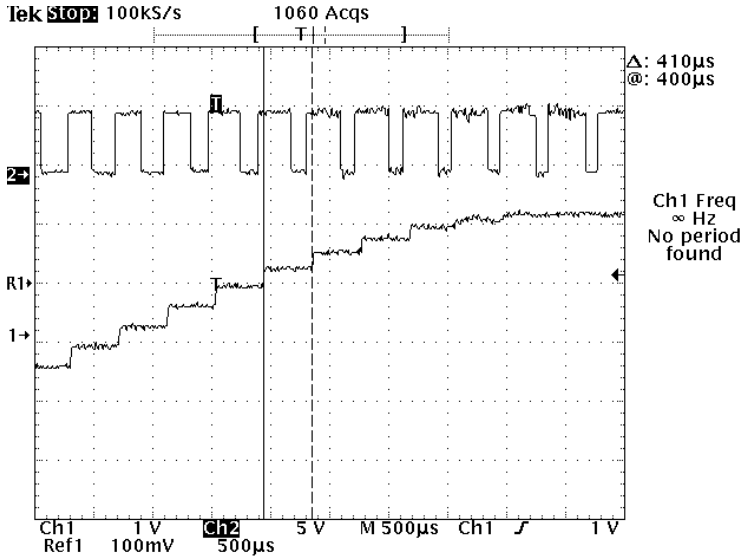


Figura 6.3 – Detalhe do período de chaveamento (T_{sw})

A limitação da baixa freqüência de chaveamento deve-se única e exclusivamente a baixa velocidade de processamento do MCU, o qual foi incapaz

de realizar todas as tarefas em períodos mais curtos do que os aqui indicados. Inversores comerciais podem operar acima de 10 kHz, buscando um compromisso entre a redução da distorção harmônica na saída do conversor, redução do ruído audível e minimização das perdas de chaveamento dos transistores da ponte inversora.

A figura 6.4 mostra a maneira como foram medidos os tempos de execução das diversas tarefas do *firmware*, onde o canal 1 monitora o tempo das interrupções da rotina principal e o tempo de processamento ocioso (livre). Quando o sinal está no nível alto o programa está executando as interrupções para temporização do período da MLP (tarefa "PWM"), quando está em nível baixo ele está executando o cálculo da MLP (tarefa "Duty Cycle"), *loop* principal (tarefa "IHM / *Loop* Principal"), V/Hz e tempo morto e, quando está no nível intermediário (porta digital em alta impedância) ele está ocioso esperando a próxima interrupção principal (sincronismo com T_{sw}).

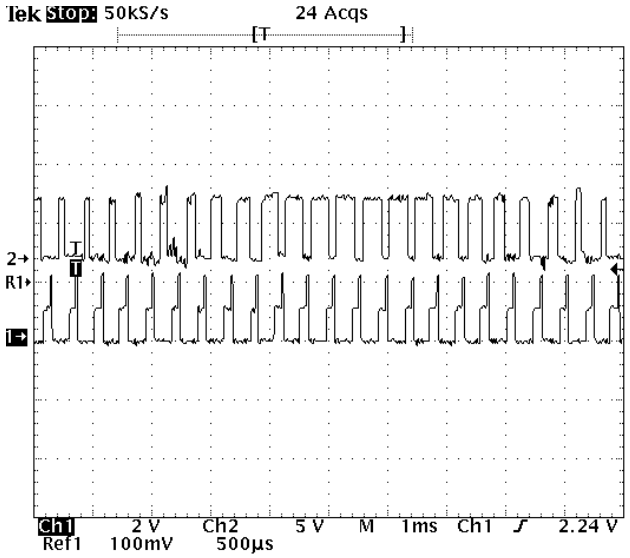


Figura 6.4 – Monitoração do tempo de execução das tarefas

Os sinais das figuras 6.4 e 6.5 foram obtidos em uma das portas de saída digital disponível no conector de interface da placa emuladora, figura 3.39.

A execução das rotinas "Duty Cycle" e da "IHM / *Loop* Principal" consumiram 300 µs do tempo de processamento do MCU, contra 319 µs

determinado pela avaliação. A rotina “PWM” consumiu 38 μs para temporizar cada uma das fases de saída, portanto para as três fases de saída serão 114 μs. Pela avaliação, tabela 5.1, seriam necessários 123,5 μs para as três fases.

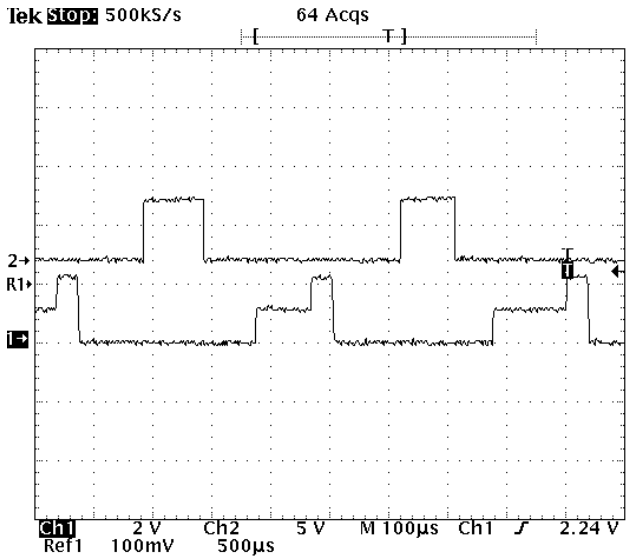


Figura 6.5 – Detalhe da monitoração do tempo de execução

6.3 Tempo-morto

As figuras 6.6 e 6.7 foram obtidas através do algoritmo de geração de tempo morto (tarefa “Tempo Morto”) e ilustram a inclusão do tempo morto para uma perna da ponte inversora, sendo possível verificar a defasagem entre a condução do transistor superior e do inferior para se evitar o problema da condução cruzada. No detalhe da figura 6.7 pode-se observar que o tempo morto foi ajustado em 10,5 μs.

Pela avaliação, tabela 5.1, para inserir o tempo-morto antes e depois do pulso seriam necessários 24 μs do tempo de processamento do MCU, ou seja, um tempo morto igual a 12 μs, pois na avaliação foram computados os tempos de inclusão do t_d antes e depois do pulso da MLP, tal como indicado na figura 3.36.

Transistores rápidos, que possuem alta velocidade de chaveamento, permitem um ajuste menor deste tempo, gerando uma menor distorção harmônica na componente fundamental da corrente do motor, porém, o MCU utilizado no

protótipo não permitiu tal redução devido a sua baixa velocidade de processamento.

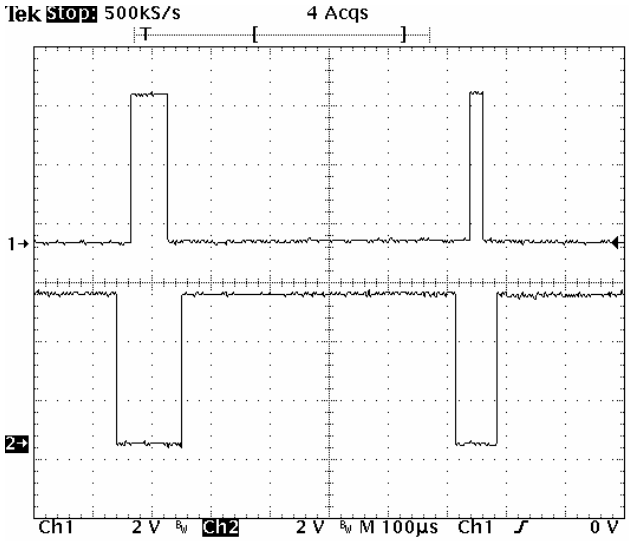


Figura 6.6 – Sinais de comando de porta do IGBT com inclusão do tempo-morto

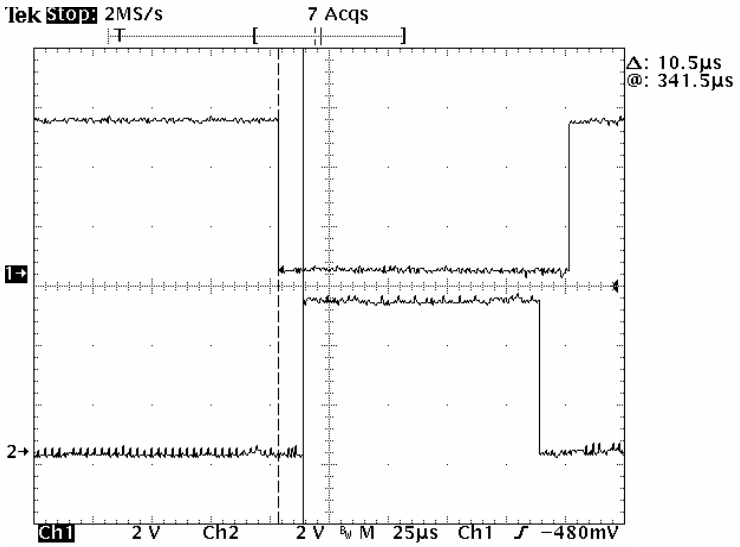


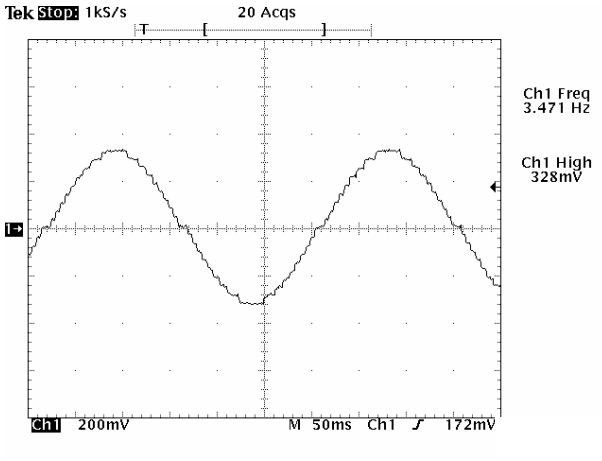
Figura 6.7 – Detalhe da inclusão do tempo-morto

6.4 Controle V/Hz

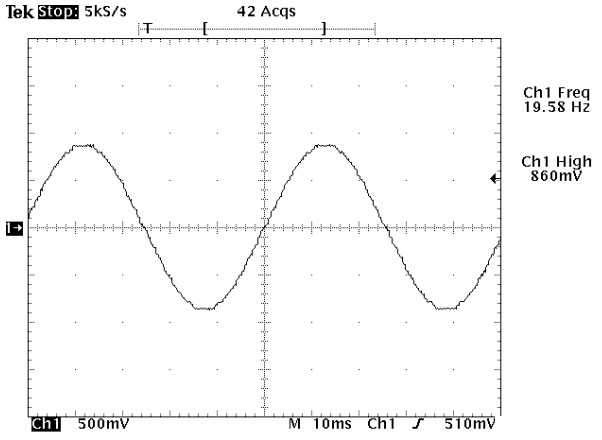
Utilizando-se a saída analógica para teste do esquema da figura 3.40 foi possível verificar a variação da freqüência de saída do conversor e ao mesmo

tempo a de tensão. As figuras 6.8(a) até 6.8(e) ilustram o ensaio do controlador V/Hz sendo f_s* e V_s* as referências de freqüência e tensão indicadas na figura 3.2.

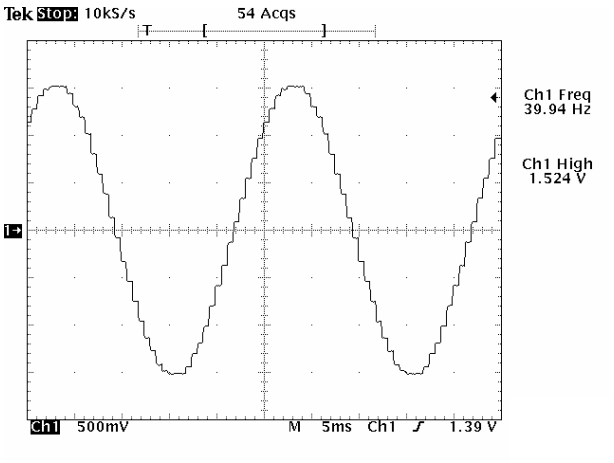
A figura 6.9 resume a operação do controlador V/Hz mostrando a relação entre freqüência e a tensão de estator, onde 1 p.u. é igual à tensão nominal do estator e corresponde a 2,0 V na saída do conversor digital para analógico (CDA) da figura 3.40.



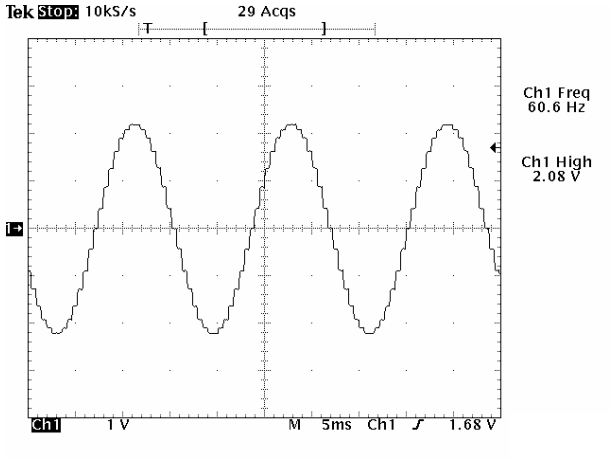
(a) f_s^* = 3,4 Hz e V_s^* = 328 mV



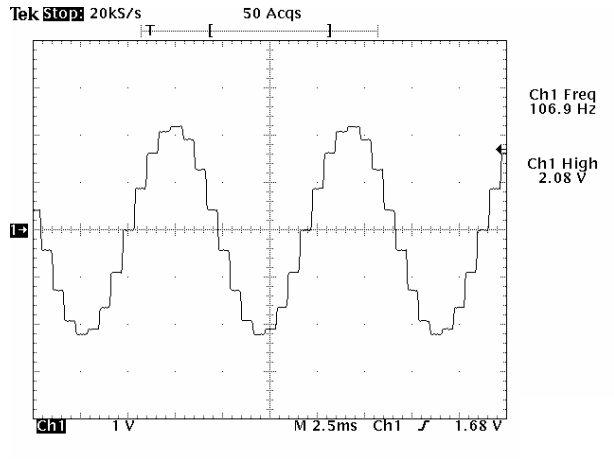
(b) f_s^* = 19,5 Hz e V_s^* = 860 mV



(c) f_s^* = 40 Hz e V_s^* = 1,52 V



(d) f_s^* = 60,6 Hz e V_s^* = 2,0 V



(e) f_s^* = 107 Hz e V_s^* = 2,0 V

Figura 6.8 (a) a (e) – Ensaio do controlador V/Hz

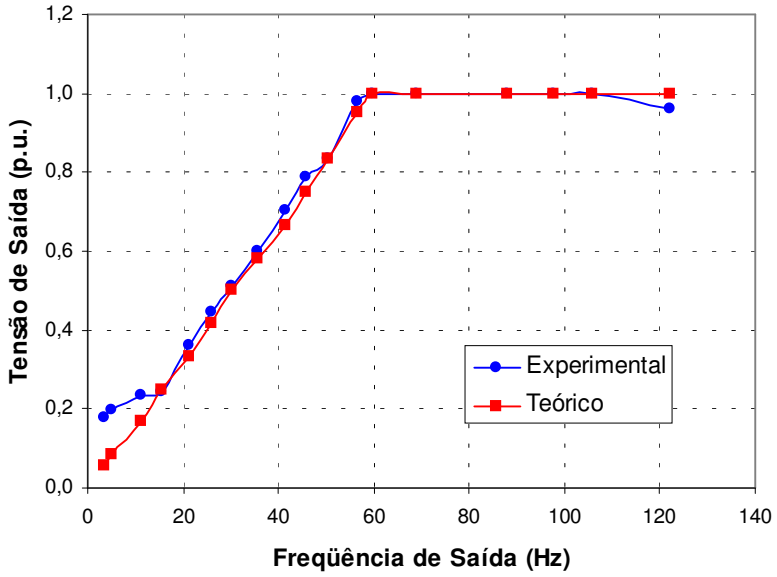


Figura 6.9 – Característica V/Hz experimental e teórica

6.5 Controle V/Hz compensado com *voltage boost*

Utilizando-se novamente a saída analógica para teste do esquema da figura 3.40 e o mesmo procedimento do item 6.4 foi possível verificar a variação de freqüência de saída do inversor e ao mesmo tempo a de tensão, porém agora impondo por *software* uma saturação na tensão mínima da máquina, de 0,23 p.u. até 15 Hz, tal como sugerido na figura 3.4. Estes valores podem ser alterados por parâmetros no *software*. O gráfico da figura 6.10 ilustra o resultado do ensaio do controlador V/Hz compensado com *voltage boost*. No protótipo este controlador foi processado em 58 µs (tarefa “Controlador V/Hz”) e pela avaliação obteve-se 46,5 µs, tal como indicado na tabela 5.1.

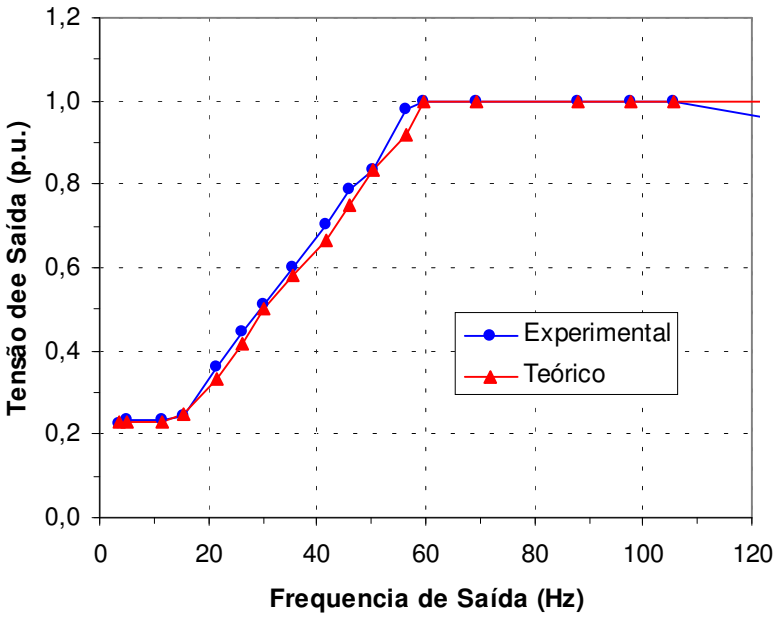


Figura 6.10 – Característica V/Hz com *voltage boost* experimental e teórica

6.6 Tamanho da memória de programa

A implementação do *firmware* apresentado no Apêndice A consumiu 834 *bytes* da memória de programa do MCU, sendo 180 *bytes* referentes a três rotinas de multiplicação que não foram consideradas na avaliação, então, para a comparação com a avaliação deve-se subtrair: 834 -180 = 654 *bytes* para as cinco tarefas periódicas. No código implementado deve-se acrescentar ainda a tabela do seno, com 360 *bytes* e as tabelas do índice de modulação em freqüência por faixa de freqüência de saída, que totalizam 150 *bytes*.

Na avaliação, tabela 5.1, foi indicado que seriam necessários 592 *bytes* para a implementação das cinco tarefas periódicas.

Desta forma, desconsiderando-se as rotinas de multiplicação, as tabelas do seno e as do índice de modulação em freqüência, as quais são iguais tanto para a avaliação quanto para o código real e comparando-se a implementação experimental com a avaliação, observa-se uma diferença de 62 *bytes* a mais no código do protótipo experimental, esta diferença se deve à simplificação do conjunto das instruções genéricas.

6.7 Comparação: avaliado versus experimental

Tal como foi dito anteriormente, a escolha do MC68HC11A1 deveu-se a dois motivos: a) disponibilidade da ferramenta de desenvolvimento no laboratório e, b) pelo fato de ser um MCU barato e acessível para uso em aplicações domésticas. Porém, como antecipado pela avaliação, constatou-se que este MCU não foi capaz de gerar os sinais MLP para as três fases de saída, em toda a faixa de freqüência de interesse. A tabela 6.1 resume os tempos de execução apresentados ao longo deste capítulo comparando a avaliação com o protótipo experimental e mostrando as divergências entre ambos, já a tabela 6.2 compara o tamanho da memória de programa necessária para a implementação de cada tarefa.

Tabela 6.1 – Tempo de execução avaliado x experimental

Tarefas	**Tempo de Execução (μs)**		**Erro (%)**
	Simulado	**Experimental**	
MLP (3 fases)	123,5	114,0	- 8,3
Tempo morto	24,0	21,0	- 14,3
Controlador V/Hz	46,5	58,0	+ 19,8
Duty-cycle (3 fases)	212,0	205,0	- 3,4
IHM/loop principal	107,0	95,0	- 12,6
Total	**513,0**	**493,0**	**- 4,0**

Tabela 6.2 – Tamanho da memória avaliado x experimental

Tarefas	**Tamanho da memória (*byte*)**		**Erro (%)**
	Simulado	*Experimental*	
MLP (3 fases)	147	161	+ 8,7
Tempo morto	30	25	- 20,0
Controlador V/Hz	51	80	- 36,2
Duty-cycle (3 fases)	241	259	+ 6,9
IHM/loop principal	123	129	+ 4,6
Total	**592**	**654**	**+ 9,5**

Sendo o erro percentual calculado por:

$$Erro\,(\%) = \frac{Experimental - Avaliado}{Experimental} \cdot 100\% \qquad (6.1)$$

As tabelas 6.3 a 6.6 mostram os fatores de utilização da CPU para o controle do motor com três fases habilitadas e para duas fases habilitadas, relatando os resultados da avaliação e experimentais.

Tabela 6.3 – Fator de utilização avaliado para 3 fases, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências, três fases de saída, avaliada	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	397	123,5	0,31
TEMPO MORTO	397	24,0	0,06
CONTROLADOR V/HZ	397	46,5	0,12
DUTY CYCLE (3 FASES)	397	212,0	0,53
IHM / LOOP PRINCIPAL	397	107,0	0,27
Total		**513,0**	**1,29**

(a)

Controle escalar em malha aberta em altas freqüências, três fases de saída, avaliada	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (3 FASES)	556	123,5	0,22
TEMPO MORTO	556	24,0	0,04
CONTROLADOR V/HZ	556	46,5	0,08
DUTY CYCLE (3 FASES)	556	212,0	0,38
IHM / LOOP PRINCIPAL	556	107,0	0,19
Total		**513,0**	**0,92**

(b)

Tabela 6.4 – Fator de utilização avaliado para 2 fases, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências, duas fases de saída, avaliada	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	397	82,3	0,21
TEMPO MORTO	397	16,0	0,04
CONTROLADOR V/HZ	397	46,5	0,12
DUTY CYCLE (2 FASES)	397	141,3	0,36
IHM / LOOP PRINCIPAL	397	107,0	0,27
Total		**393,1**	**0,99**

(a)

Controle escalar em malha aberta em altas freqüências, duas fases de saída, avaliada	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	556	82,3	0,15
TEMPO MORTO	556	16,0	0,03
CONTROLADOR V/HZ	556	46,5	0,08
DUTY CYCLE (2 FASES)	556	141,3	0,25
IHM / LOOP PRINCIPAL	556	107,0	0,19
Total		**393,1**	**0,71**

(b)

Tabela 6.5 – Fator de utilização experimental para 3 fases, apenas para altas freqüências

Controle escalar em malha aberta em altas freqüências, três fases de saída, experimental	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	580	114,0	0,20
TEMPO MORTO	580	21,0	0,04
CONTROLADOR V/HZ	580	58,0	0,10
DUTY CYCLE (2 FASES)	580	205,0	0,35
IHM / LOOP PRINCIPAL	580	95,0	0,16
Total		**493,0**	**0,85**

Tabela 6.6 – Fator de utilização experimental para 2 fases, (a) baixas e (b) altas freqüências

Controle escalar em malha aberta em baixas freqüências, duas fases de saída, experimental	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	410	76,0	0,2
TEMPO MORTO	410	14,0	0,03
CONTROLADOR V/HZ	410	38,7	0,09
DUTY CYCLE (2 FASES)	410	136,7	0,33
IHM / LOOP PRINCIPAL	410	63,3	0,15
Total		**328,7**	**0,80**

(a)

Controle escalar em malha aberta em altas freqüências, duas fases de saída, experimental	Processador: Motorola A1		
Tarefas	Período da Tarefa (us)	Tempo de Execução (us)	Fator de utilização da CPU
PWM (2 FASES)	580	76,0	0,13
TEMPO MORTO	580	14,0	0,02
CONTROLADOR V/HZ	580	38,7	0,07
DUTY CYCLE (2 FASES)	580	136,7	0,24
IHM / LOOP PRINCIPAL	580	63,3	0,11
Total		**328,7**	**0,57**

(b)

Os fatores de utilização das tabelas 6.3 a 6.6 estão representados graficamente na figura 6.11, sendo as condições de “A” até “F” definidas abaixo. Nesta figura, a condição “A” deve ser comparada com a “B”, a “C” com a “D” e a “E” com a “F”. Dois limites foram incluídos, o limite de utilização para 5 tarefas, LU(5) = 74% de acordo com o RMA e o limite imposto pelo Teste Exato (100%).

A = Avaliado / 3 fases / alta freqüência
B = Experimental / 3 fases / alta freqüência
C = Avaliado / 2 fases / baixa freqüência
D = Experimental / 2 fases / baixa freqüência
E = Avaliado / 2 fases / alta freqüência
F = Experimental / 2 fases / alta freqüência

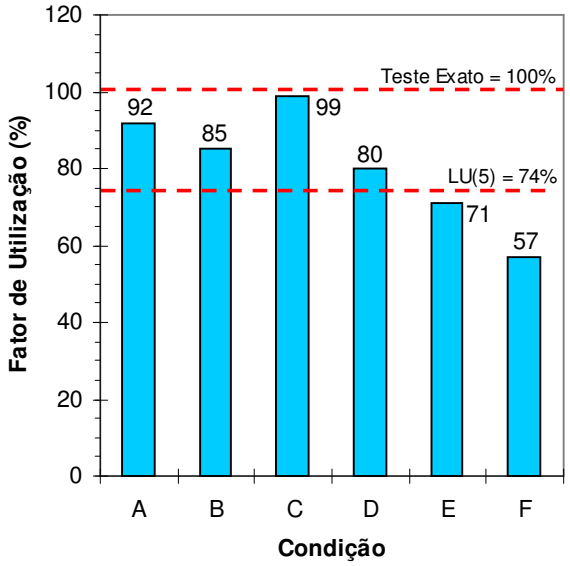


Figura 6.11 – Fatores de utilização para o Motorola HC11A1, para as diversas condições de operação

Deve ser observado que o fator de utilização da tabela 6.3(a) não está representado na figura 6.11, bem como não existem os dados para a condição “experimental / 3 fases / baixas freqüências” na tabela 6.5. Pois conforme previsto na tabela 6.3(a), não foi possível na prática escalonar todas as tarefas para esta condição de operação.

Como previsto nas avaliações, as condições “B”, “D” e “F” foram escalonadas e executadas pelo MCU MC68HC11A1 no protótipo experimental, lembrando que as condições “D” e “F” não se aplicam ao acionamento de motores trifásicos, pois uma das fases de saída foi intencionalmente desabilitada para se testar a capacidade de escalonamento do MCU utilizado.

Quando o resultado da avaliação for marginal, isto é, muito próximo dos limites impostos pelos métodos apresentados neste trabalho, o mais recomendado seria escolher um processador de maior capacidade computacional, pois como mostrado neste Capítulo, existe um erro associado à avaliação.

O capítulo 7 apresenta as conclusões e sugere novos trabalhos.

Capítulo 7

CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

O uso do motor de indução trifásico com inversor de freqüência exige uma técnica de controle de velocidade adequada, invariavelmente empregando microprocessadores digitais. Entre as diversas técnicas de acionamento de motores de indução trifásicos com velocidade variável, encontram-se as técnicas escalar, as vetoriais, o controle direto de torque e de velocidade e outras.

A técnica mais simples computacionalmente de realizar o controle de velocidade variável de motor de indução trifásico é a técnica de controle escalar que, embora sendo a pior do ponto de vista de desempenho, ela é a mais barata para uso em controle de motores de indução trifásicos em aplicações domésticas onde o motor monofásico predomina atualmente por razões simplesmente de custo.

O microprocessador, um dos elementos mais caros do sistema de controle do motor, passa a ser a figura central na otimização do custo do acionamento em velocidade variável. Torna-se então necessário utilizar-se a técnica mais simples do ponto de vista computacional, aliada ao menor e mais barato microprocessador existente no mercado capaz de suportar o software desenvolvido para tal.

A procura do microprocessador mais barato que se ajustasse perfeitamente à aplicação, ou seja, aquele que não resultasse em desperdício de capacidade de processamento tanto do ponto de vista de tempo como de memória, tornou-se o tema central deste trabalho.

Com esse microprocessador, usando a técnica de controle escalar, o custo do sistema trifásico, apesar de ainda não ser competitivo com o monofásico, pode já começar a ser viável, dependendo apenas da iniciativa dos fabricantes em querer adotá-lo.

A procura do microprocessador ideal para a aplicação exigiu um estudo demorado da bibliografia existente sobre o assunto que, apesar de ser ainda

pequena e não explorada nesta área de pesquisa, permitiu que se pudesse desenvolver um processo de identificá-lo entre os diversos disponíveis no mercado.

Essa metodologia de procura foi descrita no trabalho e constituiu uma contribuição para o estudo do problema de escolha de um microprocessador ótimo para uma dada aplicação de controle de motor, dentro de um universo muito pequeno de trabalhos desenvolvidos até o momento.

Baseado nisso, este trabalho propõe uma particularização do método genérico de avaliação, o qual possibilita o dimensionamento da capacidade computacional de processadores digitais, de forma relativamente fácil e rápida, com um erro aceitável para uma primeira estimativa. A idéia é que tais dispositivos possam ser escolhidos de modo a atender aos requisitos de uma dada aplicação de acionamento de máquinas elétricas, com folga mínima possível dos seus recursos computacionais. O objetivo é que este método venha contribuir para que a métrica de custo de um projeto de acionamento seja alcançada.

Este trabalho foi realizado com o apoio de um desenvolvimento experimental de controle escalar de velocidade de motor de indução trifásico realizado em laboratório e apresentado com todos os seus detalhes, normalmente omitidos em trabalhos desta natureza. Dada a dificuldade de encontrar e filtrar informações para a realização da parte experimental deste trabalho, entende o autor que esta também seja uma contribuição para pesquisadores iniciantes na área de controle envolvendo eletrônica de potência.

Como conclusão, aplicando-se o método descrito neste trabalho é possível predizer se um dado processador poderá ou não ser utilizado em uma aplicação de controle de máquinas elétricas. Esta é uma resposta extremamente importante quando se está iniciando um novo projeto e não há certeza se o processador de interesse conseguirá executar a carga de *software* imaginada para aplicação. Este método também poderá ser utilizado como ferramenta de auxílio na escolha de um, entre vários, processadores candidatos para um novo projeto.

Como sugestão para trabalhos futuros, pode-se refinar o método aqui proposto no sentido de reduzir os erros entre a avaliação e o experimental e seria interessante a inclusão de recursos para otimização do código genérico,

decorrente da utilização de processadores com periféricos dedicados ao controle de máquinas elétricas, pois estes periféricos reduzem a carga computacional do processador, fato este que deveria ser levado em conta na avaliação do processador.

Apêndice A

ROTINAS DO *SOFTWARE* BÁSICO

O código abaixo é a implementação em linguagem *assembly* do modulador MLP senoidal, controlador V/Hz com "*voltage boost*" e IHM (Interface Homem Máquina), que implementa o controle escalar descrito no Capítulo 3.

```
;  COMENTARIOS
;  update VER. 3
;* Tabelas de freqüências recalculadas
;* PORTB = senoide p/ o DAC
;* PORTC = fase A nos bits 6 e 7
;* PORTD = fase B nos bits 2 e 3 e fase C nos bits 4 e 5

;  update VER. 2
;* Controlador volts/hertz implementado com "voltage boost".
;* Para efeitos de visualizacao, o valor minimo e' atingido por volta de 5 Hz, e o maximo,
;  aproximadamente 60 Hz.
;* Para mudar a inclinacao da reta, modificar o valor da constante de tempo K_VHZ.
;* Para modificar o valor minimo basta alterar a constante VMIN_VHZ.
;* A modificacao de frequencia e´feita com o auxilio das chaves 1 e 3 (primeira e terceira
;* chaves), sendo que a primeira indica se deve aumentar ou diminuir e a outra indica se
;* deve mudar ou nao a frequencia.

;  update VER. 1
;* Limitadores de frequencias extremas (min = 3,4 Hz e max = 122 Hz) problema existente em
;* versoes anteriores, ja' corrigidos.
;* Apenas duas fases rodando (A e B).
;  O programa nao tem condicoes de rodar as 3 fases e o controlador V/Hz simultaneamente
;  pois o tempo de processamento do MCU e' muito longo e executa uma divisao fracionaria)
;  Se necessario, as tabelas de constantes p/ mudancas de freq. devem ser recalculadas.
;* Por motivo de falha na placa emuladora , nao e' possivel utilizar os 6 bits do PORTD
;  (as saidas D0 e D1 nao mudam de estado). Nesse caso, sobram apenas 4 outras saidas,
;  insuficuentes para representar as 3 fases. Entao, no PORTB temos a senoide de referencia
;  obtida da Fase X, e no PORTD temos D1 e D0 reservados p/ Fase A, e D3 e D2 funcionando
;* com a Fase B. E' possivel colocar a fase C nas saidas D5 e D4.
;* Dead time foi reduzido ao minimo.
;* Controlador V/Hz implementado e funcionando O.K. (s/ voltage boost), a tensao sobe ate'
;* aprox. 60Hz, qdo estabiliza no valor maximo.
;* Identificada uma falha de operacao entre as freqüências de 5 Hz e 7 Hz.
;* Do modo como foram definidas as constantes, nao e' possivel sintetizar frequencias nesse
;  intervalo. Sera' necessaria uma intervencao nao somente na tabela, mas tambem no codigo
;* do programa para permitir que constantes maiores sejam utilizadas.
;* Posteriormente elaborar o controlador V/Hz.
;* Para tanto, teremos que recalcular as tabelas utilizadas pelo programa:
;* TAB_MF_FREQ_DIM, TAB_MF_FREQ_AUM, TAB_FREQ_DIM, TAB_FREQ_AUM.
;* Programa baseado no arq: final21r.asm
;* Elaborado o Dead-time, utilizando saidas no PORTB.
;* Aprimorar o controle de frequencia, de modo que a freq. consiga ser a menor
;  possivel (ideal = 0.6 Hz). Utilizar interpolacao linear quando necessario.

;  DEFINICOES
PORTB equ $1004              ; DAC
PORTC equ $1003              ; Chaves + botao
PORTD equ $1008              ; Saidas PWM das fases (com alguns problemas)
TW_MIN equ #90               ; menor tempo de acionamento dos transistores (* 500 ns)
TW_MAX_EQU equ #90           ; maior tempo de acionamento (tempo maximo - TW_MAX_EQU)
TCNT  equ $100e              ; Contador livre 16 bits
TOC1  equ $1016              ; Comparador do Timer 1
```

```
TOC2  equ $1018             ; Comparador do Timer 2
TOC3  equ $101a             ; Comparador do Timer 3
TOC4  equ $101c             ; Comparador do Timer 4
TMSK1 equ $1022             ; Mascara de interrupcoes
TFLG1 equ $1023             ; Flags de interrupcoes
JTOC1 equ $00df             ; 'Vetor' de int. do kit p/ Timer 1
JTOC2 equ $00dc             ; 'Vetor' de int. do kit p/ Timer 2
JTOC3 equ $00d9             ; 'Vetor' de int. do kit p/ Timer 3
JTOC4 equ $00d6             ; 'Vetor' de int. do kit p/ Timer 4
DDRD  equ $1009             ; Programa direcao do PORT D
DDRC  equ $1007             ; programa direcao do PORT C
K_VHZ     equ   #46
VMIN_VHZ  equ   #29         ; 127 * 0.225 = VMIN_VHZ (29 = $1d)
                           ;(valor maximo da saida) * (tensao minima, em %) = constante

org $0000
DELTA_A:                    ; indica quantos graus passam a cada Tc
ds #2
TDELAY:
ds #1
AMPLITUDE:                  ; amplitude da senoide
ds #1
TEMP_DELTA:                 ; armazena futuro valor de DELTA, p/ nao perder valor atual
ds #2
PONT_TAB1:                  ; ponteiro p/ tab. seno da fase A
ds #2
PONT_TAB2:                  ; ponteiro p/ tab. seno da fase B
ds #2
PONT_TAB3:                  ; ponteiro p/ tab. seno da fase C
ds #2
FREQ:                       ; intervalo entre int. principais - Tc
ds #2
TEMP_FREQ:                  ; armazena futuro valor de FREQ, p/ nao mudar FREQ no meio do
processo
ds #2
TWA:                        ; tempo ligado da fase A
ds #2
TDA:                        ; metade do tempo desligado da fase A
ds #2
TWA2:                       ; TDA + TWA + TCNT
ds #2
TWB:                        ; tempo ligado da fase B
ds #2
TDB:                        ; metade do tempo desligado da fase B
ds #2
TWB2:                       ; TDB + TWB + TCNT
ds #2
TWC:                        ; tempo ligado da fase C
ds #2
TDC:                        ; metade do tempo desligado da fase C
ds #2
TWC2:                       ; TDC + TWC + TCNT
ds #2
MUDA_FASE:                  ; indica que terminou a int. principal
ds #1
FREQ_MUL:                   ; FREQ transformada em 8 bits (atualmente, e' somente a parte
ds #1                       ; superior de FREQ, mas pode ser arredondada)
CONT:                       ; temporario p/ TCNT na int. principal
ds #2
TW_MAX:                     ; constante calculada em funcao da freq p/ saber o tempo
maximo
ds #2                       ; de acionamento dos transistores

org $c000
jmp #RESET
INT_SENO:
ldaa #$f0                   ; apaga int. pendentes
staa TFLG1
ldd TCNT                    ; salva TCNT c/ o valor
std CONT                    ; do inicio da interrupcao
addd TDA                    ; programa timer da fase A
std TOC2
ldd CONT                    ; programa timer da fase B
```

```
addd TDB
std TOC3
ldd CONT                    ; programa timer da fase C
addd TDC
std TOC4
addd TWC                    ; calcula tempo de desligamento fase C
std TWC2
ldd TOC2                    ; calcula tempo de desligamento fase A
addd TWA
std TWA2
ldd TOC3                    ; calcula tempo de desligamento fase B
addd TWB
std TWB2
ldd CONT                    ; programa prox. chamada da interrupcao principal
addd FREQ
std TOC1
ldaa  #$ff                  ; indica que a interrupcao foi vetorizada
staa MUDA_FASE
rti

INT_FASEA:
ldaa PORTC                  ; le a situacao atual dos transistores da FASE A
bita #$40                   ; Caso o transistor superior (AH) esteja ligado
bne DESLIGA_AH              ; pule para DESLIGA_AH
anda #$bf                   ; caso contrario, desligue somente o transistor inferior
staa PORTC                  ; da FASE A
ldab #$40                   ; apaga interrupcoes pendentes e permite que a funcao
stab TFLG1                  ; seja escalonada da proxima vez
ldd TWA2                    ; programa o timer informando quando a interrupcao
std TOC2                    ; sera escalonada
ldaa PORTC                  ; ligue somente o transistor superior da FASE A
ora #$80
staa PORTC
rti                         ; fim da rotina

DESLIGA_AH:
anda #$7f                   ; desligua somente o transistor superior da FASE A
staa PORTC
ldab #$40                   ; apaga interrupcoes pendentes e permite que a funcao
stab TFLG1                  ; seja escalonada da proxima vez

DELAY_TEMP_A:
brn DELAY_TEMP_A
brn DELAY_TEMP_A
brn DELAY_TEMP_A
ora #$40                    ; ligua somente o transistor inferior da FASE A
staa PORTC
rti                         ; fim da rotina

INT_FASEB:
ldaa PORTD                  ; le a situacao atual dos transistores da FASE B
bita #$08                   ; Caso o transistor superior (BH) esteja ligado
bne DESLIGA_BH              ; pule para DESLIGA_BH
anda #$fb                   ; caso contrario, desligue somente o transistor inferior
staa PORTD                  ; da FASE B
ldab #$20                   ; apaga interrupcoes pendentes e permite que a funcao
stab TFLG1                  ; seja escalonada sa proxima vez
ldd TWB2                    ; programa o timer informando quando a interrupcao
std TOC3                    ; sera escalonada
ldaa PORTD                  ; ligua somente o transistor superior da FASE B
ora #$08
staa PORTD
rti                         ; fim da rotina

DESLIGA_BH:
anda #$f7                   ; desliga somente o transistor superior da FASE B
staa PORTD
ldab #$20                   ; apaga interrupcoes pendentes e permite que a funcao
stab TFLG1                  ; seja escalonada da proxima vez

DELAY_TEMP_B:
brn DELAY_TEMP_B
brn DELAY_TEMP_B
```

```
brn DELAY_TEMP_B
ora #$04                    ; ligua somente o transistor inferior da FASE B
staa PORTD
rti                         ; fim da rotina

INT_FASEC:
ldaa PORTD                  ; le a situacao atual dos transistores da FASE C
bita #$20                   ; Caso o transistor superior (CH) esteja ligado
bne DESLIGA_CH              ; pule para DESLIGA_CH
anda #$ef                   ; caso contrario, desligue somente o transistor inferior
staa PORTD                  ; da FASE C
ldab #$10                   ; apaga interrupcoes pendentes e permite que a funcao
stab TFLG1                  ; seja escalonada da proxima vez
ldd TWC2                    ; programa o timer informando quando a interrupcao
std TOC4                    ; sera escalonada
ldaa PORTD                  ; ligue somente o transistor superior da FASE C
ora #$20
staa PORTD
rti                         ; fim da rotina

DESLIGA_CH:
anda #$df                   ; desliga somente o transistor superior da FASE C
staa PORTD
ldab #$10                   ; apaga interrupcoes pendentes e permite que a funcao
stab TFLG1                  ; seja escalonada da proxima vez

DELAY_TEMP_C:
brn DELAY_TEMP_C
brn DELAY_TEMP_C
brn DELAY_TEMP_C
ora #$10                    ; ligua somente o transistor inferior da FASE C
staa PORTD
rti                         ; fim da rotina

RESET:                      ; inicio do MAIN
lds #$00bf                  ;inicializa Stack Pointer
ldaa #$01
staa TDELAY                 ;inicializa tempo morto
ldaa #$7f
staa AMPLITUDE
ldd #12
std DELTA_A
std TEMP_DELTA
ldd #1500
std FREQ
std TEMP_FREQ
ldd #TABELA
std PONT_TAB1
ldd #TABELA+120
std PONT_TAB2
ldd #TABELA+120+120
std PONT_TAB3
ldd #1111/4
std TDA
ldd #1111/2
std TWA
ldd #1111/4
std TDB
ldd #1111/2
std TWB
ldd #1111/4
std TDC
ldd #1111/2
std TWC
ldaa #$3f                   ; inicializa direcao do PORTD
staa DDRD
ldaa #$c0
staa DDRC                   ; inicializa direcao do port C
ldaa #$0                    ; zera PORTD
staa PORTD
staa PORTB
ldaa #$7e                   ; jmp p/ rotina de interrupcao
staa JTOC1
```

```
LDX #INT_SENO
stx JTOC1+1
ldaa #$7e                    ; jmp p/ rotina de interrupcao
staa JTOC2
LDX #INT_FASEA
stx JTOC2+1
ldaa #$7e                    ; jmp p/ rotina de interrupcao
staa JTOC3
LDX #INT_FASEB
stx JTOC3+1
ldaa #$7e                    ; jmp p/ rotina de interrupcao
staa JTOC4
LDX #INT_FASEC
stx JTOC4+1
ldd TCNT                     ; inicializa Counter 1 - principal
addd FREQ
std TOC1
addd #$100                   ; inicializa Counter 2 - Fase A, de modo que
std TOC2                     ; nunca sera' chamado antes da principal
std TOC3                     ; nunca sera' chamado antes da principal
std TOC4                     ; nunca sera' chamado antes da principal
ldaa #$f0                    ; apaga interrupcoes pendentes
staa TFLG1
ldaa #$f0                   ; habilita interrupcoes do timer
staa TMSK1
cli                          ; habilita interrupcoes (geral)
ldab FREQ
stab FREQ_MUL                ; armazena N p/ calculo posterior

WAIT:
ldaa MUDA_FASE
beq WAIT                     ; loop principal
clr MUDA_FASE                ; aguardando termino da int. principal
ldaa PORTC                   ; leitura das chaves
anda #$05                    ; bits 0 e 1 = chaves, bit 4 = botao (pressionado=0)
cmpa #$00
beq #DEC_FREQUENCIA
cmpa #$01
beq #INC_FREQUENCIA
ldd FREQ
std TEMP_FREQ
bra #FREQ_NAO_MUDA
DEC_FREQUENCIA:
ldd TEMP_FREQ                ; diminui a frequencia de saida
addd #$0001                  ; aumenta o Tc (Tc = FREQ no programa)
std TEMP_FREQ
bra #FREQ_AUMENTOU
INC_FREQUENCIA:
ldd TEMP_FREQ                ; aumenta a frequencia de saida
subd #$0001                  ; diminui o Tc (Tc = FREQ no programa)
std TEMP_FREQ
bra FREQ_DIMINUIU

FREQ_AUMENTOU:               ; (Frequencia da saida diminuiu)
ldd TEMP_FREQ                ; verifica se a frequencia diminuiu (Tc aumentou)
cmpd #1880 ;1325+as
bls FREQ_NAO_MUDA            ; se possivel, aumente mf (indice modulacao)
ldd #0002
cmpd DELTA_A
bne FREQ_AUMENT_CONT         ; se DELTA != 1, entao continue normalmente em
FREQ_AUMENT_CONT
ldd FREQ                     ; senao, desfaca as mudancas e pule p/ o final: FREQ_NAO_MUDA
std TEMP_FREQ
bra FREQ_NAO_MUDA

FREQ_AUMENT_CONT:
ldd #TAB_MF_FREQ_AUM         ; p/ mudar mf, mudamos a var. DELTA
addd TEMP_DELTA
xgdx
ldab 0,x                     ; consulta qual o proximo valor de DELTA
clra
std TEMP_DELTA               ; armazena DELTA
aslb                         ; b = 2 * DELTA (ajuste do indexador da tabela)
```

```
addd #TAB_FREQ_AUM          ; aponta p/ novo valor de Tc na tabela
xgdx
ldd 0,x                     ; le novo valor de Tc da tabela
std TEMP_FREQ
bra CONTR_VHZ

FREQ_DIMINUIU:              ; (Frequencia da saida aumentou)
ldd TEMP_FREQ               ; verifica se a frequencia aumentou (Tc diminuiu)
cmpd #1220 ;795+as
bhs FREQ_NAO_MUDA           ; se necessario, diminua mf
ldd #0024
cmpd DELTA_A
bne FREQ_DIMIN_CONT         ; se DELTA = 24, entao continue normalmente em
FREQ_AUMENT_CONT
ldd FREQ                    ; senao, desfaca as mudancas e pule p/ o final: FREQ_NAO_MUDA
std TEMP_FREQ
bra FREQ_NAO_MUDA

FREQ_DIMIN_CONT:
ldd #TAB_MF_FREQ_DIM        ; p/ mudar mf, muda-se a var. DELTA
addd TEMP_DELTA
xgdx
ldab 0,x                    ; consulta qual o proximo valor de DELTA
clra
std TEMP_DELTA              ; armazena DELTA
aslb                        ; b = 2 * DELTA (ajuste do indexador da tabela)
addd #TAB_FREQ_DIM          ; aponta p/ novo valor de Tc na tabela
xgdx
ldd 0,x                     ; le novo valor de Tc da tabela
std TEMP_FREQ

CONTR_VHZ:
ldaa 01                     ; carrega o valor de DELTA
ldab #K_VHZ                 ; carrega o valor da constante K
mul                         ; multiplica
ldx FREQ
fdiv                        ; divide por FREQ
xgdx                        ; Pega o resultado da divisao e coloca no Reg. D
cmpa #$7f                   ; se nao ultrapassou o valor maximo (freq atual > freq
nominal)
bls CONTR_VHZ_1             ; pule para CONTR_VHZ_OK
ldaa #$7f                   ; caso contrario, limite no valor maximo possivel
bra CONTR_VHZ_OK

CONTR_VHZ_1:
cmpa #VMIN_VHZ              ; compara com o valor minimo
bhs CONTR_VHZ_OK
ldaa #VMIN_VHZ

CONTR_VHZ_OK:
staa AMPLITUDE              ; armazena a amplitude calculada
ldd PONT_TAB3               ; atualiza apontador da tabela Fase C
addd DELTA_A                ; muda conforme a freq. da portadora
cmpd #TAB_FIM+1
blo #INT_FIM_C              ; se passou do final da tabela
subd #360                   ; subtrai 360 posicoes
std PONT_TAB3               ; salva apontador Fase C
ldd TEMP_FREQ
std FREQ                    ; atualiza o valor de FREQ
staa FREQ_MUL               ; ajusta o valor de FREQ p/ mult.
subd #TW_MAX_EQU
std TW_MAX                  ; calcula o novo valor de TW_MAX
bra INT_FIM2_C

INT_FIM_C:
std PONT_TAB3               ; salva apontador Fase C

INT_FIM2_C:
ldd PONT_TAB2               ; atualiza apontador da tabela Fase B
addd DELTA_A                ; muda conforme a freq. da portadora
cmpd #TAB_FIM+1
blo #INT_FIM_B              ; se passou do final da tabela
subd #360                   ; subtrai 360 posicoes
```

```
std PONT_TAB2               ; salva apontador Fase B
ldd TEMP_FREQ
std FREQ                    ; atualiza o valor de FREQ
staa FREQ_MUL               ; ajusta o valor de FREQ p/ mult.
subd #TW_MAX_EQU
std TW_MAX                  ; calcula o novo valor de TW_MAX
bra INT_FIM2_B

INT_FIM_B:
std PONT_TAB2               ; salva apontador Fase B

INT_FIM2_B:
ldd PONT_TAB1               ; atualiza apontador da tabela Fase A
addd DELTA_A                ; muda conforme a freq. da portadora
cmpd #TAB_FIM+1
blo #INT_FIM_A              ; se passou do final da tabela
subd #360                   ; subtrai 360 posicoes
std PONT_TAB1               ; salva apontador Fase A
ldd TEMP_DELTA
std DELTA_A                 ; atualiza o valor de DELTA somente no final do ciclo da FASE
A
ldd TEMP_FREQ
std FREQ                    ; atualiza o valor de FREQ
staa FREQ_MUL               ; ajusta o valor de FREQ p/ mult.
subd #TW_MAX_EQU
std TW_MAX                  ; calcula o novo valor de TW_MAX
bra INT_FIM2_A

INT_FIM_A:
std PONT_TAB1               ; salva apontador Fase A

INT_FIM2_A:
xgdx
ldab 0,x                    ; carrega em B o valor atual do seno
ldaa AMPLITUDE              ; carrega em A a amplitude
bmi #NUM_NEG_A              ; se no. e' negativo, muda multiplicacao
mul
lsla                        ; pega mais um bit de precisao na multipl.
eora #$80                   ; soma $80 ao no. (retira o sinal)
bra FIM_MUL_A

NUM_NEG_A:
negb                        ; pega o modulo do numero e
mul                         ; multiplica
lsla                        ; pega mais um bit de precisao na multipl.
nega                        ; inverte o sinal do no.
eora #$80                   ; soma $80 ao no. (retira o sinal)

FIM_MUL_A:
ldab FREQ_MUL               ; carrega N (FREQ em 8 bits)
mul                         ; multiplica => N * ()
cmpd TW_MAX                 ; verifica se o valor de TWA calculado e' menor que o maximo
blo SEM_AJUSTE_A            ; e maior que o minimo, mudando se necessario.
ldd TW_MAX
bra SEM_AJUSTE2_A

SEM_AJUSTE_A:
cmpd #TW_MIN
bhi SEM_AJUSTE2_A
ldd #TW_MIN

SEM_AJUSTE2_A:
std  TWA                    ; armazena o valor de TWA calculado
ldd FREQ                    ; calcula o tempo complementar (TDA)
subd TWA
clc
rora                        ; divide por dois o reg. D
rorb
std TDA

;COMECO_MUL_B:
ldaa AMPLITUDE              ; carrega em A a amplitude
ldx PONT_TAB2
```

```
ldab 0,x                    ; carrega em B o valor atual do seno
bmi #NUM_NEG_B              ; se no. e' negativo, muda multiplicacao
mul
lsla                        ; pega mais um bit de precisao na multipl.
eora #$80                   ; soma $80 ao no. (retira o sinal)
bra FIM_MUL_B

NUM_NEG_B:
negb                        ; pega o modulo do numero e
mul                         ; multiplica
lsla                        ; pega mais um bit de precisao na multipl.
nega                        ; inverte o sinal do no.
eora #$80                   ; soma $80 ao no. (retira o sinal)

FIM_MUL_B:
staa PORTB                  ; mostra senoide no DAC
ldab FREQ_MUL               ; carrega N (FREQ em 8 bits)
mul                         ; multiplica => N * ()
cmpd TW_MAX                 ; verifica se o valor de TWB calculado e' menor que o maximo
blo SEM_AJUSTE_B            ; e maior que o minimo, mudando se necessario.
ldd TW_MAX
bra SEM_AJUSTE2_B

SEM_AJUSTE_B:
cmpd #TW_MIN
bhi SEM_AJUSTE2_B
ldd #TW_MIN

SEM_AJUSTE2_B:
std  TWB                    ; armazena o valor de TWB calculado
ldd FREQ                    ; calcula o tempo complementar (TDB)
subd TWB
clc
rora                        ; divide por dois o reg. D
rorb
std TDB

;COMECO_MUL:
ldaa AMPLITUDE              ; carrega em A a amplitude
ldx PONT_TAB3
ldab 0,x                    ; carrega em B o valor atual do seno
bmi #NUM_NEG_C              ; se no. e' negativo, muda multiplicacao
mul
lsla                        ; pega mais um bit de precisao na multipl.
eora #$80                   ; soma $80 ao no. (retira o sinal)
bra FIM_MUL_C

NUM_NEG_C:
negb                        ; pega o modulo do numero e
mul                         ; multiplica
lsla                        ; pega mais um bit de precisao na multipl.
nega                        ; inverte o sinal do no.
eora #$80                   ; soma $80 ao no. (retira o sinal)

FIM_MUL_C:
ldab FREQ_MUL               ; carrega N (FREQ em 8 bits)
mul                         ; multiplica => N * ()
cmpd TW_MAX                 ; verifica se o valor de TWC calculado e' menor que o maximo
blo SEM_AJUSTE_C            ; e maior que o minimo, mudando se necessario.
ldd TW_MAX
bra SEM_AJUSTE2_C

SEM_AJUSTE_C:
cmpd #TW_MIN
bhi SEM_AJUSTE2_C
ldd #TW_MIN

SEM_AJUSTE2_C:
std  TWC                    ; armazena o valor de TWC calculado
ldd FREQ                    ; calcula o tempo complementar (TDC)
subd TWC
clc
rora                        ; divide por dois o reg. D
```

```
rorb
std TDC
ldaa MUDA_FASE               ; verifica se as contas foram acabadas
beq WAIT2                    ; antes do final da int. principal
sei                          ; se nao deu tempo, trave o programa

WAIT2:
jmp  WAIT
; Tabela de mf em função de fs
; Tabela que indica qual o proximo valor de DELTA, caso a frequencia aumente (Tc diminui)

TAB_MF_FREQ_DIM:
  db #00, #02, #03
  db #04
  db #06
  db #00
  db #09
  db #00, #00
  db #12
  db #00, #00
  db #18
  db #00, #00, #00, #00, #00
  db #24
  db #00, #00, #00, #00, #00
  db #24

; Tabela que indica qual o proximo valor de DELTA, caso a frequencia diminua (Tc aumente)

TAB_MF_FREQ_AUM:

  db #$00, #01, #01
  db #02
  db #03
  db #00
  db #04
  db #00, #00
  db #06
  db #00, #00
  db #09
  db #00, #00, #00, #00, #00
  db #12
  db #00, #00, #00, #00, #00
  db #18

; Tabela que indica qual o proximo valor de FREQ (Tc), caso o mf seja alterado

TAB_FREQ_DIM:

 dw #0000, #1831, #1831
 dw #1627
 dw #1830
 dw #0000
 dw #1830
 dw #0000, #0000
 dw #1627
 dw #0000, #0000
 dw #1830
 dw #0000, #0000, #0000, #0000, #0000
 dw #1627
 dw #0000, #0000, #0000, #0000, #0000
 dw #1627

; Tabela que indica qual o proximo valor de FREQ (Tc), caso o mf seja alterado

TAB_FREQ_AUM:
 dw #0000, #1253, #1253
 dw #1253
 dw #1410
 dw #0000
 dw #1253
 dw #0000, #0000
 dw #1253
 dw #0000, #0000
```

```
 dw #1410
 dw #0000, #0000, #0000, #0000, #0000
 dw #1253
 dw #0000, #0000, #0000, #0000, #0000
 dw #1410

; Tabela do seno

TABELA:
  db #$0
  db #$2
  db #$4
  db #$6
  db #$8
  db #$b
  db #$d
  db #$f
  db #$11
  db #$13
  db #$16
  db #$18
  db #$1a
  db #$1c
  db #$1e
  db #$20
  db #$23
  db #$25
  db #$27
  db #$29
  db #$2b
  db #$2d
  db #$2f
  db #$31
  db #$33
  db #$35
  db #$37
  db #$39
  db #$3b
  db #$3d
  db #$3f
  db #$41
  db #$43
  db #$45
  db #$47
  db #$48
  db #$4a
  db #$4c
  db #$4e
  db #$4f
  db #$51
  db #$53
  db #$54
  db #$56
  db #$58
  db #$59
  db #$5b
  db #$5c
  db #$5e
  db #$5f
  db #$61
  db #$62
  db #$64
  db #$65
  db #$66
  db #$68
  db #$69
  db #$6a
  db #$6b
  db #$6c
  db #$6d
  db #$6f
  db #$70
  db #$71
```

```
db #$72
db #$73
db #$74
db #$74
db #$75
db #$76
db #$77
db #$78
db #$78
db #$79
db #$7a
db #$7a
db #$7b
db #$7b
db #$7c
db #$7c
db #$7d
db #$7d
db #$7d
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7e
db #$7d
db #$7d
db #$7d
db #$7c
db #$7c
db #$7b
db #$7b
db #$7a
db #$7a
db #$79
db #$78
db #$78
db #$77
db #$76
db #$75
db #$74
db #$74
db #$73
db #$72
db #$71
db #$70
db #$6f
db #$6d
db #$6c
db #$6b
db #$6a
db #$69
db #$68
db #$66
db #$65
db #$64
db #$62
db #$61
db #$5f
db #$5e
db #$5c
db #$5b
db #$59
db #$58
```

```
db #$56
db #$54
db #$53
db #$51
db #$4f
db #$4e
db #$4c
db #$4a
db #$48
db #$47
db #$45
db #$43
db #$41
db #$3f
db #$3d
db #$3b
db #$39
db #$37
db #$35
db #$33
db #$31
db #$2f
db #$2d
db #$2b
db #$29
db #$27
db #$25
db #$23
db #$20
db #$1e
db #$1c
db #$1a
db #$18
db #$16
db #$13
db #$11
db #$f
db #$d
db #$b
db #$8
db #$6
db #$4
db #$2
db #$0
db #$fe
db #$fc
db #$fa
db #$f8
db #$f5
db #$f3
db #$f1
db #$ef
db #$ed
db #$ea
db #$e8
db #$e6
db #$e4
db #$e2
db #$e0
db #$dd
db #$db
db #$d9
db #$d7
db #$d5
db #$d3
db #$d1
db #$cf
db #$cd
db #$cb
db #$c9
db #$c7
db #$c5
db #$c3
```

```
db #$c1
db #$bf
db #$bd
db #$bb
db #$b9
db #$b8
db #$b6
db #$b4
db #$b2
db #$b1
db #$af
db #$ad
db #$ac
db #$aa
db #$a8
db #$a7
db #$a5
db #$a4
db #$a2
db #$a1
db #$9f
db #$9e
db #$9c
db #$9b
db #$9a
db #$98
db #$97
db #$96
db #$95
db #$94
db #$93
db #$91
db #$90
db #$8f
db #$8e
db #$8d
db #$8c
db #$8c
db #$8b
db #$8a
db #$89
db #$88
db #$88
db #$87
db #$86
db #$86
db #$85
db #$85
db #$84
db #$84
db #$83
db #$83
db #$83
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$82
db #$83
db #$83
db #$83
db #$84
db #$84
```

```
db #$85
db #$85
db #$86
db #$86
db #$87
db #$88
db #$88
db #$89
db #$8a
db #$8b
db #$8c
db #$8c
db #$8d
db #$8e
db #$8f
db #$90
db #$91
db #$93
db #$94
db #$95
db #$96
db #$97
db #$98
db #$9a
db #$9b
db #$9c
db #$9e
db #$9f
db #$a1
db #$a2
db #$a4
db #$a5
db #$a7
db #$a8
db #$aa
db #$ac
db #$ad
db #$af
db #$b1
db #$b2
db #$b4
db #$b6
db #$b8
db #$b9
db #$bb
db #$bd
db #$bf
db #$c1
db #$c3
db #$c5
db #$c7
db #$c9
db #$cb
db #$cd
db #$cf
db #$d1
db #$d3
db #$d5
db #$d7
db #$d9
db #$db
db #$dd
db #$e0
db #$e2
db #$e4
db #$e6
db #$e8
db #$ea
db #$ed
db #$ef
db #$f1
db #$f3
db #$f5
```

```
  db #$f8
  db #$fa
  db #$fc

TAB_FIM:
  db #$fe
end
```

Apêndice B

DISPOSITIVOS ELETRÔNICOS

B.1 Chaves Semicondutoras

Os conversores eletrônicos para acionamento de motores de corrente alternada, com velocidade variável, utilizam chaves semicondutoras para implementar a ponte inversora. Nos conversores de alta potência e baixa freqüência utilizam-se tiristores (SCR), os quais estão restritos a aplicações industriais, de tal forma que a maior parte das aplicações são para potências médias e baixas. Para estes casos o elemento de chaveamento é indiscutivelmente o transistor. O transistor de junção bipolar (TJB) foi bastante utilizado em conversores de freqüência devido às baixas perdas de condução, normalmente chaveando abaixo de 10 kHz, em aplicações abaixo de 1200 V e 400 A e operando normalmente na configuração emissor-comum. Atualmente são pouco utilizados neste tipo de aplicação em função da sua baixa freqüência de comutação e pelo fato de serem dispositivos controlados por corrente, o que encarece o circuito de acionamento da base.

Os transistores MOSFET de potência são usados em conversores de alta freqüência de chaveamento os quais estão disponíveis comercialmente em potências não muito altas, normalmente são caracterizados para 1000 V, 50 A e freqüências na faixa de dezenas a centenas de kHz. Para aplicações de baixa tensão, os MOSFETs de potência oferecem resistências extremamente baixas no estado ligado, $R_{DS(on)}$, e aproximam-se de uma chave ideal. Em aplicações de alta tensão, os MOSFETs exibem um aumento de $R_{DS(on)}$ resultando em baixa eficiência devido ao aumento das perdas de condução.

Da combinação das baixas perdas de condução do TJB com a velocidade de chaveamento do MOSFET surgiu um novo dispositivo, o *Insulated-Gate Bipolar Transistor* (IGBT), cuja tecnologia combina estes atributos. Os IGBTs possuem alta impedância de entrada e alta velocidade de comutação do estado desligado

para o ligado. Estes dispositivos substituem os MOSFETs em aplicações de altas tensões onde as perdas de condução devem ser mantidas baixas. Os MOSFETs são muito utilizados devido à simplicidade do circuito de acionamento (*drive*) do seu terminal de porta (*gate*), porém, como a estrutura do IGBT é muito similar, na maior parte dos casos é possível substituir um MOSFET por um IGBT sem que se tenha necessidade de re-projetar este circuito. Da mesma forma que os MOSFETs, os IGBTs são dispositivos baseados em transcondutância, os quais podem permanecer ligados mantendo-se a tensão de porta acima de um certo limiar. Os IGBTs, são inerentemente mais rápidos que os TJB, mas ainda não tão rápidos quanto os MOSFETs. Entretanto, são aplicáveis em acionamentos que requerem altas tensões, altas correntes e podem operar em freqüências de até 20 kHz. Estão disponíveis comercialmente até 1200 V e 400 A.

As tabelas B.1 e B.2 comparam alguns dos principais dispositivos de potência utilizados em acionamento de máquinas elétricas e a referência [67] compara MOSFETs e IGBTs em aplicações de controle de motor.

Tabela B.1 – Comparação entre dispositivos semicondutores de potência

Dispositivo		**Tensão Reversa (kV)**	**Corrente Direta (A)**	**Freq. Op. (kHz)**	**Tempo de Chaveamento (μs)**	**Resistência Ligado (mΩ)**
Diodos	Propósito geral	5	5000	1	100	0.16
	Alta velocidade	3	1000	10	2 a 5	1
	Schottky	0,04	60	20	0.23	10
Tiristores	Bloqueio reverso	5	5000	1	200	0.25
Desligam.	Alta velocidade	1.2	1500	10	20	0.47
Forçado	Condução reversa	2.5	400	5	40	2.16
	GATT	1.2	400	20	8	2.24
	LASCR	6	1500	400	200 a 400	0.53
Tiristores	GTO	4.5	3000	10	15	2.5
Auto-deslig.	SIT	4	2200	20	6.5	5.75
TRIAC		1.2	300	400	200 a 400	3.57
TJB	Simples	0.4	250	20	9	4
		0.4	40	20	6	31
		0.63	50	25	1.7	15
	Darlington	1.2	400	10	30	10
MOSFET	Simples	0.5	8.6	100	0.7	0.6
		1	4.7	100	0.9	2
		0.5	50	100	0.6	0.4
IGBT	Simples	1.2	400	20	2.3	60
SIT		1.2	300	100	0.55	1.2
MCT	Simples	0.6	60	20	2.2	18

Tabela B.2 – Comparação entre transistores de potência

Dispositivo	Tipo de controle	Potência de Controle	Circuito de controle	Requer Circuito Snubber	Tensão estado ligado
TJB	Corrente	Alta	Complexo	Não	baixa
MOSFET	Tensão	Baixa	Simples	Não	alta
IGBT	Tensão	Baixa	Simples	Não	baixa

Os conversores de potência dependem cada vez mais das características de chaveamento dos dispositivos de estado-sólido, de tal forma que os fabricantes de semicondutores sentem-se impelidos a criar produtos que se aproximem de chaves ideais. Uma chave ideal deveria possuir as seguintes características: a) resistência zero ou queda de tensão nula em estado ligado, b) resistência infinita em estado desligado, c) freqüência de chaveamento infinita e, d) não requer potência de entrada para fazê-la comutar [11].

Quando utiliza uma chave real, o projetista do *hardware* deve desviar-se das características da chave ideal e escolher um dispositivo que melhor se adapte à sua aplicação com um mínimo de perda de eficiência. A escolha envolve considerações tais como: tensão, corrente, freqüência de chaveamento, circuito de acionamento da chave, carga e os efeitos da temperatura. Como foi visto antes, existem diversas tecnologias de chaves de estado-sólido disponíveis comercialmente que possuem pontos fortes e fracos. As características primárias mais desejáveis em uma chave de estado-sólido são: alta velocidade de chaveamento, circuitos de acionamento de base ou *gate* simples e baixa perda de condução.

Estão disponíveis no mercado inúmeras configurações e arranjos de transistores de potência, tais como: transistores discretos com e sem diodos de retorno, semi-pontes, pontes monofásicas, trifásicas, com transistor extra para freio, com *gate drive*, com sensor de corrente e módulos compactos, entre outros. Entretanto, muitas destas configurações – exceto os transistores discretos – ainda são caras para serem utilizadas em eletrônica de consumo.

A tabela B.3 compara os preços de alguns dispositivos disponíveis comercialmente e passíveis de serem utilizados neste tipo de aplicação.

Em um produto destinado ao consumo, os transistores discretos apresentam as seguintes desvantagens: ocupam uma maior área na placa de

circuito impresso, geralmente requerem um dissipador maior, suas características elétricas variam mais do em uma ponte integrada.

Como vantagem, pode-se destacar: se um transistor da ponte falhar, só será substituída a peça com defeito, ao contrário da ponte integrada que é inutilizada totalmente. O custo é menor e maiores são as opções na escolha dos dispositivos.

Tabela B.3 – Comparativo de custo de transistores de potência

Dispositivo	Fabricante	Tipo	Preço (*)	Características
IRF730	ST	MOSFET	1,00	400V/5,5A/1,0Ω/Canal N/TO220
MGP4N60ED	ON Semicon.	IGBT	1,00	600V/4A/TO220
MJF18008	ON Semicon.	TJB	1,08	450V/8A/NPN/TO220
IRF730	IR	MOSFET	1,37	400V/5,5A/1,0Ω/Canal N/TO220
IRG4BC20F	IR	IGBT	1,58	600V/9A/3-10KHz/TO220
IRG4BC10K	IR	IGBT	1,62	600V/5A/3-10KHz/TO220
IRF830S-ND	IR	MOSFET	1,70	500V/4,5A/1,5Ω/Canal N/SMD
IRF740	IR	MOSFET	2,16	500V/8A/0,85Ω/Canal N/TO220
IRF840S-ND	IR	MOSFET	2,50	500V/8A/0,85Ω/Canal N/SMD
IRG4BC30K-S	IR	IGBT	2,65	600V/16A/10-100KHz/SMD
IRG4BC20FD-S	IR	IGBT	2,83	600V/9A/3-10KHz/SMD

(*) preços de referência em dólares americanos, fornecidos pelos fabricantes ou distribuidores.

B.2 Processadores

O capítulo 4 mostrou a importância da escolha correta do processador digital para a aplicação desejada, principalmente quando se tratam de projetos sensíveis a custo, onde uma escolha mal feita pode inviabilizar a aplicação.

Da mesma forma que os transistores, os processadores, sejam eles, microprocessadores, microcontroladores os processadores digitais de sinais, estão disponíveis comercialmente em várias opções, fazendo com que a escolha por parte do projetista seja cada vez mais difícil. Este sabe que existe um processador adequado - técnica e comercialmente - à sua aplicação. Cabe a ele encontrá-lo. A tabela B.4 compara alguns dispositivos comerciais passíveis de utilização nas aplicações de interesse deste trabalho. Deve ser observado que os fabricantes estão disponibilizando cada vez mais processadores dedicados para vários tipos de aplicações, entre elas, o controle de máquinas elétricas, facilitando

a implementação. Porém, em eletrônica de baixo custo, nem sempre o mais fácil é o melhor.

Tabela B.4 – Comparativo de custo de processadores

Dispositivo	Fabricante	Tipo	Preço (*)	Características
MC68HC11P1	Motorola	MCU	1,00	Uso geral
MC68HC908JK1	Motorota	MCU	1,30	Uso geral
MC68HC908JK3	Motorola	MCU	1,70	Uso geral
MC68HC908JL3	Motorota	MCU	2,10	Uso geral
MC68HC11A1	Motorola	MCU	2,20	Uso geral
MC68HC08MR4	Motorola	MCU	2,50	Dedicado a controle de motor
MC68HC908MR4	Motorola	MCU	2,50	Dedicado a controle de motor
COPSGE728	NSC	MCU	2,55	Uso geral
COPCG884	NSC	MCU	2,70	Uso geral
ADMC328	Analog Devices	DSP	2,95	Dedicado a controle de motor em aplicações de baixo custo
Z8PE013	Zilog	MCU	3,00	Uso geral
TMS320LC2402	Texas	DSP	3,00	Dedicado a controle de motor em aplicações de baixo custo
MC68HC908MR24	Motorola	MCU	8,00	Dedicado ao controle de motor em aplicações industriais
ADMC401	Analog Devices	DSP	15,00	Alta performance, dedicado ao controle de motor em aplicações industriais

(*) preços de referência em dólares americanos, fornecidos pelos fabricantes ou distribuidores.

Apêndice C

REFERÊNCIAS BIBLIOGRÁFICAS

[1] C. Chen, D. Divan, D. Novotny, "A Hybrid Inverter/Cycloconverter-Based Variable-Speed Three-Phase Induction Motor Drive for Single-Phase Inputs", *IEEE Transactions on Ind. Application*, vol. 31, No. 3, May/Jun. 1995

[2] P. Enjeti, A. Rahman, R. Jakkli, "Economic Single-Phase to Three-Phase Converter Topologies for Fixed and Variable Frequency Output", *IEEE Transactions on Power Electronics*, vol. 8, no. 3, Jul. 1993

[3] B. Welchko, T. Lipo, "A Novel Variable Frequency Three-Phase Induction Motor Drive System Using Only Three Controlled Switches", *IEEE Industry Application Society Annual Meeting*, Rome, Italy, Oct. 2000

[4] K. Jungreis, "Adjustable Speed Drive for Residential Applications," *IEEE Transactions on Ind. Application*, vol. 31, no. 6, Dec. 1995

[5] L. Clawson, J. Petty, J. Adamski, "PSC Motor Improvement for Direct Drive Clothes Washer Applications", *IEEE Transactions on Ind. Application*, vol. 30, no. 2, Mar./Apr. 1994

[6] Y. Shimma, K. Iida, "Inverter Applications to Air Conditioning Field," *IPEC*, pp 1747, Tokyo, 2000

[7] A. Murray, "DSP Motor Control Boosts Efficiency in Home Appliances," *Electronic Design*, vol. 46, no. 12, pp. 102, may 1998

[8] --------, "Portrait of the Latin American Appliance Industry," *Appliance Magazine*, vol. 57, no. 12, Dec 2000

[9] --------, "Japanese Makers Competing for Higher CPOs in PACs," *Appliance Magazine*, vol. 59, no. 1, Jan 2002

[10] J. Donlon, J. Achhammer, H. Iwamoto, M. Iwasaki, "Power Modules for Appliance Motor Control," *IEEE Industry Applications Magazine*, pp. 26-34, Jul/Aug 2002

[11] N. Mohan, T. Underland, W. Robbins, *Power Electronics - Converters, Applications and Design*. Segunda Edição - John Wiley & Sons, Inc., 1995

[12] M. Rashid, *Power Electronics - Circuits, Devices and Applications*. Segunda Edição, Prentice Hall, 1993

[13] C. Ong, *Dynamic Simulation of Electric Machinery*. Primeira Edição, Prentice Hall PTR, 1998

[14] R. Krishnan, *Electric Motor Drives - Modeling, Analysis and Control*. Primeira Edição, Prentice Hall Inc., 2001

[15] F. Labrique, J. Santana, *Eletrônica de Potência*. Primeira Edição, Fundação Calouste Gulbenkian, 1991

[16] R. Houldsworth, "Introduction to PWM Speed Control System for 3-Phase AC Motors," *Electronic Components and Applications*, Vol. 2, No. 2, Feb. 1980.

[17] G. Dubey, *Power Semiconductor Controlled Drives*. Prentice Hall, 1989

[18] B. Bose, *Power Electronics and AC Drives*, Primeira Edição, Prentice Hall, 1986

[19] S. Bowes, "Discussion of an Algebraic Algorithm for Microcomputer-Based (direct) Inverter Pulsewidth Modulation," *IEEE Transactions on Ind. Application*, vol. 24, No. 6, Nov./Dec. 1988

[20] --------, "Constant V/Hz Operation for Variable Speed Control of Induction Motor," Analog Devices Application Note AN331-24, Jan. 2000

[21] D. Garcia, "Precision Digital Sine-Wave Generation with the TMS32010," Texas Instruments, Application Report SPRA007, 1989

[22] A. Chrysafis, "Digital Sine-Wave Synthesis Using the DSP56001/2," Motorola Digital Signal Processing Division, APR1, 1988

[23] R. Valentine, *Motor Control Electronics Handbook*, Primeira Edição, McGraw Hill, 1998

[24] A. Oppenhein, R. Schafer, *Discrete Time Signal Processing,* Primeira Edição, Prentice Hall, 1989

[25] R. Alves, E. da Silva, A. Lima, "Moduladores MLP para Inversores com Entrada CC Constante e Pulsada" COBEP, pp. 650-655, 1997

[26] B. Bose, H. Sutherland, "A High Performance Pulsewidth Modulator for an Inverter-Fed Drive System Using a Microcomputer," *IEEE Transactions on Ind. Applications*, vol. IA-19, no. 2, Mar./Apr. 1983

[27] F. Moynihan, “High-Performance Motion Control”, *Power Electronic Systems – PCIM Europe*, no. 1/2, 1999

[28] S. Bowes, “Advanced Regular-Sampled PWM Control Technique for Drives and Static Power Converters,” *IEEE Transactions on Industry Applications*, vol. 42, no. 4, pp.367, Aug. 1995

[29] S. Bowes, P. Clark, “Simple Microprocessor Implementation of New Regular-Sampled Harmonic Elimination PWM Techniques,” *IEEE Transactions on Ind. Applications*, vol. 28, no. 1, Jan./Feb. 1992

[30] B. Mwinyiwiwa, Z. Wolanski, B. Ooi, “Microprocessor-Implemented SPWM for Multiconverters with Phase-Shifted Triangle Carriers,” *IEEE Transactions on Ind. Applications*, vol. 34, no. 3, May/Jun. 1998

[31] A. Muñoz, T. Lipo, “On-Line Dead-Time Compensation Technique for Open-Loop PWM-VSI Drives,” *IEEE Transactions on Power Electronics*, vol. 14, no. 4, Jul. 1999

[32] S. Choi, “Inverter Output Voltage Synthesis Using Novel Dead-Time Compensation,” *IEEE Transactions on Power Electronics, vol. 11, no. 2, Mar. 1996*

[33] D. Wilson, “Making Low-Distortion Motor Waveforms with the MC68HC708MP16,” Motorola Field Applications, AN1728, 1997

[34] V. Agelidis, P. Ziogas, G. Joos, “Dead-Band PWM Switching Patterns,” *IEEE Transactions on Power Electronics*, vol. 11, no. 4, Jul. 1996

[35] T. Sukegawa, K. Kamiyama, K. Mizuno, T. Matsui, T. Okuyama, “Fully Digital, Vector-Controlled PWM VSI-Fed AC Drives with an Inverter Dead-Time Compensation Strategy,” *IEEE Transactions on Ind. Applications, vol. 27, no. 3, May/Jun. 1991*

[36] N. Kannan, “Simple Dead-Band Generator Targets Push-Pull Drives,” *Electronic Design*, vol. 47, no. 14, pp. 90, Jul. 1999

[37] M. Di Guardo, G. Grasso, M. Lo Presti, “Three-Phase Motor Control by Using ST52x301,” *ST Microelectronics*, Application Note AN1112, Jan. 1999

[38] D. Leggate, R Kerkman, “Pulse-Based Dead-Time Compensator for PWM Voltage Inverters,” *IEEE Transactions on Ind. Applications*, vol. 44, no. 2, Apr. 1997

[39] Y. Murai, T. Watanabe, H. Iwasaki, “Waveform Distortion and Correction Circuit for PWM Inverters with Switching Lag-Times,” *IEEE Transactions on Ind. Applications*, vol. IA-23, no. 5, Set/Oct. 1987

[40] *Motorola Semiconductor Data Manual,* "MC68HC11A8," Revision 3, 1991

[41] F. Blaadjerg, J. Pedersen, U. Jaeger, P. Thoegersen, "Single Current Sensor Technique in the DC Link of Three-Phase PWM-VSI Inverters: A Review and a Novel Solution," *IEEE Transactions on Ind. Application*, vol. 33, No. 5, Set./Oct. 1997

[42] *International Rectifier Data Manual*, "High and Low Side Driver IR2112" Data Sheet no. PD60026-K

[43] *Mitsubishi Electric*, "3rd Generation IGBT and Intelligent Power Modules Application Manual"

[44] *Intersil Data Manual*, "HGTP7N60C3D 14A, 600V, UFS Series N-Channel IGBT with Anti-Parallel Hyperfast Diodes"

[45] W. Wolf, "Hardware-software Co-design of Embedded Systems," *Proceedings of the IEEE*, vol. 82, no. 7, July 1994

[46] N. Woo, A. Dunlop, W. Wolf, "Codesign from Cospecification," *Computer*, Jan. 1994

[47] R. Gupta, G. de Micheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design & Test of Computers*, Sept. 1993

[48] J. Adams, D. Thomas, "The Design of Mixed Hardware/Software Systems," *33rd Design Automation Conference*, Jun. 1996

[49] D. Gajski et al, *Specification and Design of Embedded Systems,* Prentice Hall, Inc., 1994

[50] M. Magalhães, "Algoritmos de Escalonamento para Sistemas de Tempo Real Crítico," Notas de aula, Unicamp, 1995

[51] M. Klein, J. Lehoczky, R. Rajkumar, "Rate-Monotonic Analysis for Real-time Industrial Computing," *Computer Magazine*, pp 24, Jan 1994

[52] J. Gong, D. Gajski, S. Narayan, "Software Estimation from Executable Specifications," *Journal of Computer and Software Engineering*, 1994

[53] W. Wray, J. Greenfield, *Using Microprocessor and Microcomputers, the Motorola Family*, Prentice Hall Inc. Third Edition, 1994

[54] N. Audsley, "Deadline Monotonic Scheduling," Department of Computer Science, University of York, Heslington, England, Sept. 1990, disponível em: http://citeseer.nj.nec.com/cs, [consulta: 23/08/03]

[55] A. Burns, “Scheduling Hard Real-Time Systems: A Review,” *Software Engineering Journal*, 6, pp 116, May 1991

[56] N. Audsley, A. Burns, M. Richardson, A. Wellings, “Hard-Real Time Scheduling: The Deadline-Monotonic Approach,” Department of Computer Science, University of York, Heslington, England, disponível em: http://citeseer.nj.nec.com/cs, [consulta: 23/08/03]

[57] L. Sha, B. Sprunt, J. Lehoczky, “Aperiodic Task Scheduling for Hard Real-Time Systems,” *The Journal of Real-Time Systems*, 1, pp 27-69, 1989

[58] C. Liu, J. Layland, “Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment,” *Journal of the ACM*, January, 1973

[59] G. Manimaran, “Fundamentals of Real-Time Systems,” Department of Electrical and Computer Engineering, Iowa State University, disponível em: http://citeseer.nj.nec.com/cs, [consulta: 23/08/03]

[60] L. Sha, M. Klein, J. Goodenough, “Rate Monotonic Analysis for Real-Time Systems,” Technical Report CMU/SEI-91-TR-6, March 1991, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA

[61] J. Lehoczky, L. Sha, Y. Ding, “The rate monotonic scheduling algorithm: Exact characterization and average case behaviour,” *Proceedings IEEE Real-Time Systems Symposium*, Santa Monica, California, pp 166, Dec. 1989

[62] N. Forman, “Rate Monotonic Theory,” disponível em: http://citeseer.nj.nec.com/cs, [consulta: 23/08/03]

[63] L. Sha, J. Goodenough, “Real-time scheduling theory and ADA,” *IEEE Computer*, no. 23, pp 53, Apr. 1990

[64] Z. Yu, D. Figoli, “Using Constant V/Hz Principle and Space Vector PWM Technique for AC Induction Motor Control with TMS320C24,” Texas Instruments Application Note, SPRA284, Dec. 1997

[65] H. Rajamani, R. McMahon, “Induction Motor Drives for domestic Appliances”, *IEEE Industry Application Magazine*, May/Jun. 1997

[66] B. Cunha, J. Camacho, C. Bissochi, “Single-Phase Induction Motor speed Control Through a PIC Controlled Sinusoidal PWM Inverter – The Mathematical Model and Various Load Conditions”, Escola de Engenharia Elétrica, Universidade Federal de Uberlândia, não publidado

[67] B. Maurice, T. Castagnet, “Comparison of MOSFET and IGBT Transistors in Motor Drive Applications” *Discrete Power Semiconductor Handbook*, 1st edition, AN663, ST Microelectronics, Apr. 1995

[68] Simone Buso, “DSP and Microcontroller Applications in Power Electronics”, apostila de curso oferecido pelo Prof. Buso (Universidade de Padova – Itália) na FEEC/Unicamp, Agosto de 2004.

Apêndice D

ARTIGO PUBLICADO

O artigo *Digital Processor Performance Estimation Applied to Low Cost AC Motor Drives* foi aceito e apresentado na conferência *IEEE International Electric Machine and Drives Conference* – IEMDC 2003, realizado em Madison, Wi, USA em Junho de 2003. Este artigo está reproduzido a seguir.

Digital Processor Performance Estimation Applied to Low Cost AC Motor Drives

Mário L. Botêga Jr., Ernesto Ruppert Filho
DSCE/FEEC/UNICAMP
CP 6101- CEP 13083970 - Campinas - SP – Brazil

***Abstract* – Adjustable speed drives are migrating from industrial applications to home appliances, where the cost is one of the main product requirements. The correct choice of digital microprocessor is essential to achieve the cost restrictions. This paper presents some performance metrics which can be used to select the right processor for each motor control application. The simulation results were validated by an experimental laboratory prototype.**

I. INTRODUCTION

Adjustable speed motor drives based on voltage source inverters (VSI) have been used in a large variety of industrial processes since the 70s. Their popularity is due to their fast and accurate speed control, load torque, harmonic elimination, and some others abilities. After some decades of intensive industrial applications, residential appliances manufacturers are focusing on drives using fractional three-phase induction motors fed from PWM-VSI, properly designed for home application, due to the increasingly search for cost reductions, performance improvements and even electrical energy efficient use. Contrasting with industrial applications where the performance requirements are very important, in the AC low cost drives designed for residential applications, speed control accuracy and fast responses are almost neglected.

Although the adjustable-speed drives are very efficient they are more complex, and even more expensive, so that the design of fractional three-phase induction motor drives, for low cost applications, is still a technical and economic big challenge. However, the large market for this type of application is searching for technical and economic improvements.

In this type of drive the power section is imposed by the load and it doesn't permit any optimization while the control part is imposed by the application and gives to the designer the opportunity to perform a suitable design and implementation. In industrial applications this type of drive has been implemented using digital signal processors (DSP) or high performance microcontrollers (MCU) and closed-loop space vector control techniques demanding very high computational capacity of the digital processors. In residential appliances applications only a few of those techniques – even nowadays – are allowed to be used and the big challenge is the performance implementation control, using low cost and low performance computational devices. So, the matter is how to select the right processor for each application. In the last years we can find several technical papers about embedded systems hardware and software co-design to get rid of the design empiricism. These two parts, hardware and software, must be designed together to ensure that the application, not only works properly, but it reaches its performance, cost and reliability requirements.

This paper presents a method to predict digital processor performance when used in motor drives applications, providing to the drive designers a simulation tool to help them during the processor choice.

II. DESIGN ESTIMATION

In the first steps of embedded system design the designer needs to estimate program size, data size memory and execution time requirements to decide how to manage the software tasks time constraints (like tasks deadlines), to decide which processor will take place in one's design. This decision will have straight influence on cost and design performance. Designers have used their own experience to decide the first firmware design steps and this procedure is not optimized and has slow convergence ratio to get to a good solution. Reference [1] proposed some estimation quality metrics, useful to characterize embedded systems, where the most important things are the hardware, the software costs and performance metrics. These will enable the system designer to explore design alternatives by providing quick feedback for any design decision. This way a designer can explore a great number of alternatives instead of synthesizing a complete implementation and measuring the particular design quality metric for each design alternative.

A. Software Estimation

For a software implementation, the behavior has to be compiled into the instruction set of the

target processor. In the case of a single processor implementation, the execution of the concurrent tasks may have to be intercalated in order to satisfy data dependencies or timing constraints. According to [1], two models can be adopted for software estimation: the processor specific model and the generic model.

A.1 Processor-specific Estimation Model

Under the processor-specific estimation model, the exact value of a metric is computed by compiling each behavior into the instruction set of the desired processor using a specific compiler for that processor. From the *timing* information (such as the number of clock cycles to execute each instructions) and *size* information (such as the number of bytes required for each instruction) associated with each processor, it is possible to determine the size and execution time for the compiled behavior. The main advantage of this model is that the estimation obtained are very accurate, since the behavior is actually compiled for execution on the selected processor. However, such an estimation approach requires a dedicated compiler for each processor. Consequently, it is difficult to adapt an existing estimator for a new processor. In addition, compiling an algorithm just for the estimation purposes is computationally expensive.

A.2 Generic Estimation Model

Instead of using different compilers and estimator for different target processors in the processor-specific model, an alternative generic estimation model was proposed by [2], shown in the Fig. 1. Under the generic estimation model, the tasks are first compiled into a set of generic instructions. Processor-specific technology files are made available containing information about the number of clock cycles and bytes that each type of generic instruction requires. The estimator computes the software metrics for the tasks based on the generic instructions and technology files for target processors. Technology files can be derived from processor's instruction set, for each target processor. The proposed generic instruction set consist of the following five classes:

1. Arithmetic/logic/relational instructions;
2. Move/load/store instructions;
3. Conditional jump instructions;
4. Unconditional jump instructions;
5. Procedure calling instructions.

Compared with the processor-specific model, the generic estimation model has several advantages. First, the generic model does not require different compilers and estimators for different target processors. Second, this model makes retargeting the estimator to a new processor much easier. Retargeting consists of providing a technology file for the new processor. A disadvantage of the generic model is the lower accuracy of its estimation due to the fact of the generic instruction set represents only a small portion of the processor's entire instruction set. But as we are going to see in the simulation presented below when compared with the experimental prototype, the error introduced by generic model can be accepted due its estimation easiness

B. Program Execution Time Estimation

The software or program execution time of an algorithm can be determined in one of the two following ways: dynamic simulation and static simulation. Given a set of input data, dynamic simulation executes the program and records the number of clock cycles required for each simulation. Given different sets of input data, dynamic simulation may obtain a different number of clock cycles for that program due to the data- dependent conditional branches and loops.

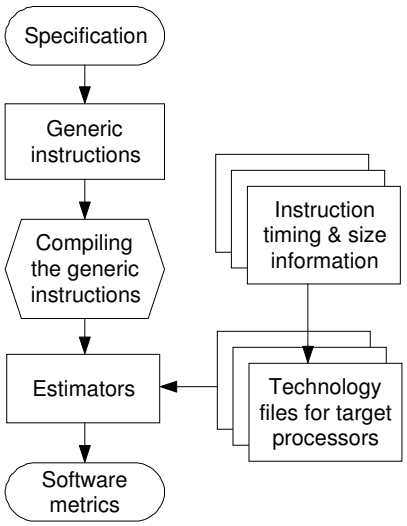


Fig. 1 – Generic model estimation

Static simulation, on the other hand, is insensitive to input data and can yield fairly accurate results if the number of loop iterations is known and the conditional branching probability can be predicted. In addition, static simulation is faster and requires less memory space than dynamic simulation. The static simulation uses the generic model estimation with advantages and disadvantages already mentioned. This paper

considers static simulation only, due its sufficient accuracy for the first design approach.

B.1 Static Execution Time Estimation

If an algorithm is described by *n* straight-lines code, the start-to-finish execution time for it depends on the number of control steps into which the behavior is scheduled. Let *cstep* be the number of control steps estimated for the schedule for the algorithm *B*, and let T_{clk} be clock cycle selected for the design implementation. Then the execution time, *exectime*, in seconds, is computed as follow:

$$exectime(B) = \sum_{j=1}^{n} cstep(j) \cdot T_{clk} \quad (1)$$

In general case, a computer program may consist of sequential statements that have branching and iteration structures (such as loops, if and case statements). Since branching may be dependent upon input data, (1) cannot be directly applied to determine the execution time for such behaviors. A technique based on the branch execution frequency can be used. The branch probability is a measure of how often a branch is executed. To determine the execution time for the entire task, the execution time needs to be weighed by the execution frequency, *freq*, of each control step, that defines the number of times that each control step will be executed on the average, during a single execution of the behavior, then the (1) can be re-write as follow:

$$exectime(B) = \sum_{j=1}^{n} cstep(j) \cdot freq(j) \cdot T_{clk} \quad (2)$$

C. Program Memory Size Estimation

The first step in program size estimation also involves compiling the given algorithm into a set of generic instructions, where the size of each type of generic instruction is specified in the technology file for the target processor. Based on the size of each generic instruction, the program-memory size of an algorithm is then computed as the sum of those generic instructions in this algorithm. So, if a task *B* is compiled into a set of generic instructions and *instrsize* represents the size of the generic instruction, then the size of the program required for task *B* is computed as follows:

$$progsize(B) = \sum_{j=1}^{n} instrsize(j) \quad (3)$$

D. Scheduling Estimation

On digital motor control software applications there are several time concurrent tasks, such as: transistor, TRIAC or SCR firing, AC line synchronism, voltage and current measurements, fail detections, feedback monitoring, etc. The choice of which task will get a specific computational resource and in which task sequence, is done by a software task called scheduler. These kinds of software applications are characterized by using time constraint tasks (hard deadlines) and the several software tasks share the same computational resources, such as the processor, data memory, timers, etc. These tasks must finish their computation before the deadline comes, otherwise they can induce to a bad functioning or even an accident. So, this is the reason why the scheduler is so important.

According to [4] hard real-time scheduling algorithms can be in one of two ways: static or dynamic. Static scheduling algorithms settle the scheduling tasks sequence, conferring them a fixed priority before the execution starts (off line). To do this, a previous task's behavior knowledge is necessary to determine the scheduling sequence. In the other hand, the dynamic scheduling algorithms settle the scheduling tasks sequence in real-time (on line) way, during the software execution, conferring to each task a priority that can change along the time. Comparing the two algorithms, it is possible to conclude that the static imposes low processor overhead, however they are inflexible and cannot adapt themselves to environmental changes. Otherwise, the dynamic algorithms impose larger processor overhead due to the continuous priority changing, but they provide more flexibility to the system.

In the examples showed below it was considered static scheduling algorithms only, because it is possible to predict the priority and the execution sequence to the periodic tasks. It was not considered non-periodic tasks, such as fail detection tasks, that are usually scheduled by hardware interruption.

D.1 Monotonic Ratio Algorithm

This algorithm is static algorithm (fixed priorities) applied to periodic tasks (in this algorithm the non-periodic tasks scheduling can be emulated by a polling server) that can be interrupted by a higher priority tasks and tasks with deadlines.

Once a set of tasks given, the Monotonic Ratio algorithm settles to each one a fixed priority according to their periods, just like that; a small period implies a bigger priority and vice versa. Reference [3] established a necessary and sufficient condition to assure that the periodic tasks deadlines are not exceeded, the theorem is as follows:

A set of m periodic and independent tasks, scheduled by Monotonic Ratio algorithm, always accomplish their deadline if:

$$U(m) = \sum_{i=1}^{m} \frac{C_i}{P_i} \leq m \cdot \left(2^{1/m} - 1\right) = LU(m) \qquad (4)$$

Where $U(m)$ is the CPU usage ratio for the m tasks, C_i is the computation time of the task i, P_i is period of the task i and $LU(m)$ is the CPU usage limit of the m tasks over that, the processor will be unable to schedule the task set considered. This theorem can be used to evaluate if a task set, with its respective deadlines and priorities, will be scheduled.

This means that it is possible to predict if there will be enough CPU time to compute all tasks within their periods and deadlines. This is a fundamental characteristic during processor specification. The procedure described above is generic, however it was adapted to estimate motor control applications.

III. PERFORMANCE ESTIMATION

To illustrate the technique described above, two static simulations were performed for low cost, three-phase, AC motor drive software, using pulse-width modulated inverter fed from voltage source (PWM-VSI). Two modulation techniques were simulated: scalar and space vector modulation. Under scalar modulation, an experimental prototype was built using a slow MC68HC11A1 MCU, to validate the simulation results.

Under generic estimation model was found: the computation time to each tasks, the memory size and the scheduling prediction. The applications related above were simulated with specific processors data, shown in the Table I and further generically appointed as A, B, C, D, F and G processors.

In Table I we can find timing information and in Table II, generic instructions (GI) size, for each instruction class, organized by bytes/clock cycles per instructions, to each specific processors related.

For each application it was developed a standard algorithm constituted by generic instructions, which was processed through the schematic shown in the Fig. 1. The Fig. 2 give us the code size of scalar modulation and computation time and Fig. 3 shows the CPU's usage ratio and the CPU's usage limit for each processor.

A standard software was developed for PWM-VSI, controlled under scalar modulation and was performed by five tasks, as indicated in Tables III, IV.a and IV.b that takes a small piece of application software to illustrate the code conversion, or generic compiling from assembly language to generic instructions, as stated on item A.2.

TABLE I
Technology file, *timing*

Processor	Int. Clock (MHz)	Clock Cyc. (ns)
A	40,0	50
B	40,0	30
C	8,2	120
D	33,0	120
E	25,0	160
F	2,0	500
G	1,0	1000

TABLE II
Technology file, *size*

Processor	Generic Instructions				
	Class 1	Class 2	Class 3	Class 4	Class 5
A	1/1	1/1	1/1	1/4	1/4
B	1/2	1/2	1/2	1/6	1/10
C	2/2	2/3	2/3	2/3	1/7
D	1/1	1/1	1/1	1/2	1/2
E	2/4	2/4	2/4	2/6	1/12
F	3/4	2/3	2/3	2/3	1/12
G	3/4	2/3	3/4	3/4	1/5

As can be noticed in these tables, there are small errors between "real" code and "generic" code. According (4) the usage limit is $LU(5)$=74,3%. In the Fig. 3 we can notice that just some processors are in accordance with Monotonic Ratio. Two situations were simulated: modulator output frequency of 3.5 Hz and 120 Hz. The processors A, B, C and D can be used from 3.5 up to 120 Hz, because their usage ratio is below LU. Some processors (E and F) can schedule all tasks for 3.5 Hz, but cannot do it until 120 Hz, their usage ratio is above LU. Finally, the processor G cannot be used in any situation, nether for low nor for high output frequencies, so this processor is rejected for this application. Fig. 4 and 5 compare the simulation with experimental results. Fig. 4 shows the relationship between output frequency and CPU usage. The crossing points of $LU(5)$ with simulated and experimental CPU usage ratios occurred in 55 Hz and 62 Hz respectively, so as it was predicted during the simulation, the MCU used on experimental prototype was unable to compute the application software for all output frequency range desired (3 ~ 120Hz). The results above $LU(5)$ were obtained "disconnecting" the software tasks for two output phases. As indicated in (4), small period brings large CPU usage ratio, that has direct influence over processor scheduling capability, as showed in Fig. 4.

TABLE III
Scalar modulator tasks data

	Memory Size (*progsize*)			CPU Computation Time (us) (*exectime*)			CPU Usage Ratio (%)					
							Output Frequency 3,5 Hz			Output Frequency 120 Hz		
Tasks	Simulated	Experim.	Error (%)	Simulated	Experim.	Error (%)	Simulated	Experim.	Error (%)	Simulated	Experim.	Error (%)
PWM Modulator	166	180	8,39	131	143	9,2	14,6	13,1	10,9	32,0	28,6	11,7
Dead Time	30	32	7,14	19,5	21	7,7	2,2	1,9	12,5	4,8	4,2	13,2
V/Hz Controller	54	60	11,76	60	68	13,3	6,7	6,2	6,9	14,6	13,6	7,6
Duty Cycle	255	272	6,67	224	240	7,1	24,9	22,0	13,0	54,6	48,0	13,8
Scheduler	143	149	4,37	98,5	103	4,6	10,9	9,4	15,8	24,0	20,6	16,6
Total	648	695	7,30	533	575	7,9	59,2	52,8	12,2	130,0	115,0	13,0

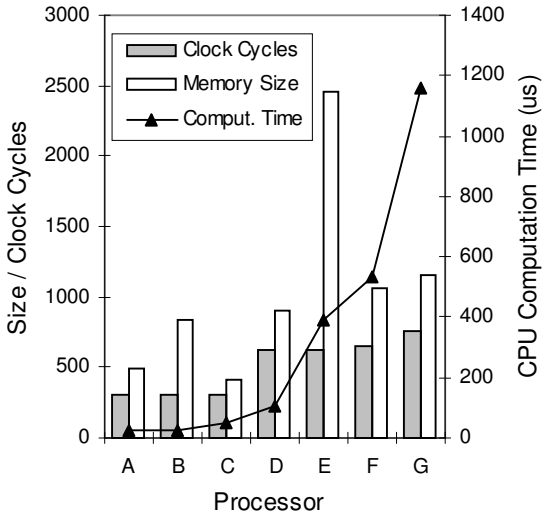


Fig. 2 - Size and computation time

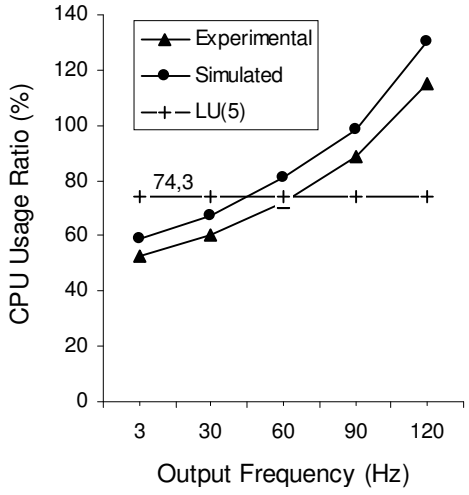


Fig. 4 - CPU usage ratio: experimental x simulated

Fig. 5 compares CPU computation time for all software tasks used in this application example.

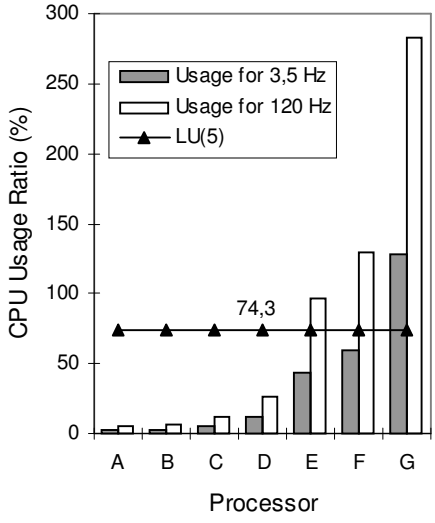


Fig. 3 - CPU usage ratio and limit

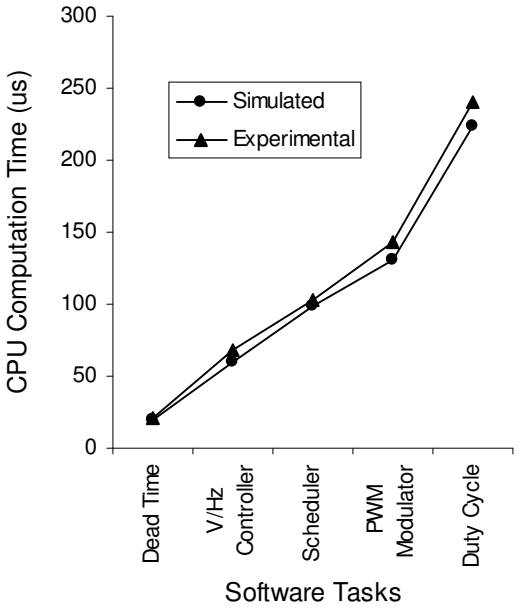


Fig. 5 - Tasks computation time

It can be noticed that the error between the simulation and experimental work is small, showing that the generic model used on the simulation is good. Table III gives the same errors.

TABLE IV.a
Application code piece

MC68HC11A1 Instructions	Bytes	Machine Cycles	CPU Usage (us)
ldaa PORTC	2	3	1.5
bita #$40	2	3	1.5
bne TURN_OFF_AH	2	3	1.5
anda #$bf	2	3	1.5
staa PORTC	2	3	1.5
ldab #$40	2	3	1.5
stab TFLG1	2	3	1.5
ldd TWA2	3	3	1.5
std TOC2	2	4	2.0
ldaa PORTC	2	3	1.5
ora #$80	2	2	1.0
staa PORTC	2	3	1.5
rti	1	12	6.0
Total	**26**	**48**	**24**

Generic Compiling

TABLE IV.b
Translation from assembly code into generic instructions

Generic Instructions	Class	Bytes	Machine Cycles	CPU Usage (us)
mls	2	2	3	1.5
dec	3	2	3	1.5
dec	3	2	3	1.5
alr	1	3	4	2.0
mls	2	2	3	1.5
mls	2	2	3	1.5
mls	2	2	3	1.5
mls	2	2	3	1.5
mls	2	2	3	2.0
mls	2	2	3	1.5
alr	1	3	4	2.0
mls	2	2	3	1.5
sub	5	1	12	6.0
Total	**-**	**27**	**50**	**25.5**

Fig. 6 and 7 illustrate a space vector modulation simulation. This application example didn't have their simulation results validated by an experimental prototype, like the scalar modulation, however as we could expect, a small amount of Table I processors can be used to schedule the standard software tasks developed to this simulation. Basically the DSP's.

The microcontrollers and DSP´s considered in this work were: MC68HC11A1, MC68HC08MR4, DSP56800 from Motorola, TMS320LC2402 from Texas Instruments, ST92141 from ST Microelectronics, PIC17C752 from Microchip and COP8SGE7 from National Semiconductors.

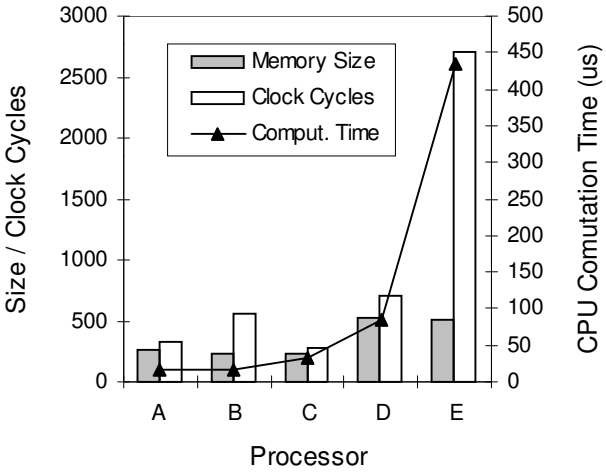


Fig. 6 - Size and computation time for space vector

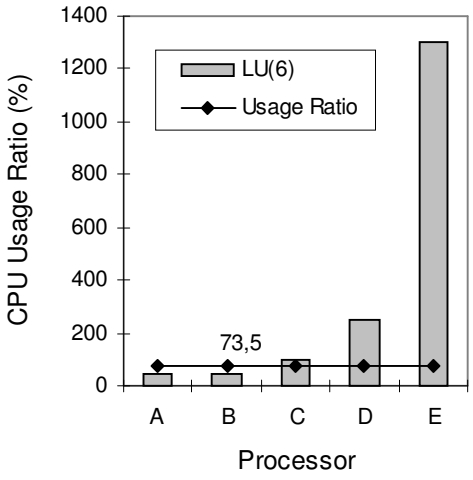


Fig. 7 - CPU usage ratio and limit for space vector

IV. CONCLUSION

This paper proposes an easy and quick simulation method to estimate if a digital processor can fit a desired motor control application, with an acceptable level of error and a minimum overhead of computational resources. In low cost drives, excess resources imply in more expensive drives, once in consumer electronics area cost does make difference. Design cost reduction is a constant searching for large scales drive's production. This method contributes to reach this metric.

V. REFERENCES

[1] D.D. Gajski, F. Vahid, S. Narayan and J. Gong, “Specification and design of embedded systems”, Prentice Hall, Inc, 1994.

[2] J. Gong, D. Gajski and S. Narayan, “Software estimation from executable specifications”, Journal of Computer and Software Engineering, 1994.

[3] C. Liu and J. Layland,, “Scheduling algorithms for multi-programming in a hard real-time environment”, Journal of the ACM, 1973, pp 40-61.

[4] M. Magalhães, “Scheduling algorithms for hard real-time systems”, Unicamp, 1995, Brazil, in Portuguese.