SILVIO SZAFIR

SISTEMA BASEADO EM PROCESSADOR DIGITAL DE SINAIS PARA CONTROLE EM TEMPO REAL DE MOTOR ELÉTRICO

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia

São Paulo
2000

SILVIO SZAFIR

SISTEMA BASEADO EM PROCESSADOR DIGITAL DE SINAIS PARA CONTROLE EM TEMPO REAL DE MOTOR ELÉTRICO

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia

Área de Concentração:
Engenharia Mecatrônica e de Sistemas Mecânicos

Orientador:
Prof. Dr. Marcelo Godoy Simões

São Paulo
2000

AGRADECIMENTOS

Ao Prof. Dr. Marcelo Godoy Simões, pela orientação segura, pelo incentivo e pelo apoio no decorrer do trabalho, expressando o significado concreto da palavra Orientador.

À minha familia, RaquelClaudiaMilenaRubinRicardoAriehRafael, pela paciência, compreensão, o apoio e, principalmente, pelo bom ambiente proporcionado.

Aos politécnicos e futuros engenheiros, Eric Conrado de Souza e Newton Cesar Omune, pela parceria, dedicação, empenho e aprendizagem que este projeto proporcionou.

Ao amigo Nilson Noris Franceschetti, pelo companheirismo e suporte, em todos os momentos. Agradecimento estendido aos amigos Paulo E. M. de Almeida, ao Petronio Vieira Jr., ao F. Javier R. Pelaez, ao Paulo Peres, ao A. Fernando Maiorquim e ao Marcos Hunold.

Aos professores envolvidos, pela oportunidade de discutir idéias, receber sugestões e acima de tudo, aprender. À amiga Silvana A. Silva, presente em alguns destes momentos.

Ao meu amigo Amir Manasterski, pelas dicas, discussões, sugestões, apoio e acima de tudo, seu refinado senso de humor.

Ao engenheiro Alan Tavares de Souza, pelo suporte e companheirismo em momentos críticos durante a implementação deste projeto. Estendo o agradecimento, aos professores Newton Maruyama, Celso M. Furukawa, Jun Okamoto Jr., Etore Apolônio de Barros e Jaime Simão Sichman.

À FAPESP pelo apoio financeiro à realização deste trabalho, através do processo No. 98/11351-7, do Motor Embutido em Roda.

À empresa Motorola do Brasil, na pessoa do engenheiro Paulo Batista Lopes, pela sua presença e suporte neste trabalho.

O meu reconhecimento sincero àqueles que anônimamente colaboraram para a conclusão deste objetivo.

Errata

no texto da dissertação:

"Sistema Baseado em Processador Digital de Sinais para Controle em Tempo Real de Motor Elétrico"

Página 1, onde se lê "... as áreas em há necessidade ...", leia-se "... as áreas em que há necessidade ..."
Página 8, onde se lê "[19][19][21]", leia-se "[19][20][21]"
Página 13, onde se lê "a (3) em (2)", leia-se "a (3) em (1)"
Página 15, onde se lê "Fig. 3::", leia-se "Fig. 3:"
Página 26, na equação (12) onde se lê

$$v_c(n) = v_c(n-1) + \left(\frac{KI}{2}\cdot T - K_P\right)\cdot e(n-1) + \left(\frac{K_I}{2}\cdot T + K_P\right)\cdot e(n)$$, leia-se

$$v_c(n) = v_c(n-1) + \left(\frac{K_I}{2}\cdot T - K_P\right)\cdot e(n-1) + \left(\frac{K_I}{2}\cdot T + K_P\right)\cdot e(n)$$

Página 26, onde se lê "dissipar este energia", leia-se "dissipar esta energia"

Página 28, na equação (13) onde se lê $(V_a)_{médio} = \frac{1}{Ts}\int_0^{Ts} V_a(t)dt$, leia-se $(V_a)_{médio} = \frac{1}{T_s}\int_0^{Ts} V_a(t)dt$

Página 34, onde se lê "[9][38]", leia-se "[11][38]"
Página 36, onde se lê "retorno um ao", leia-se "retorno ao"
Página 44, onde se lê "tipos recursos", leia-se "tipos de recursos"
Página 49, onde se lê "possível conhecer", leia-se "possível de conhecer"
Página 53, onde se lê "unidade acesso outro", leia-se "unidade acesse outro"
Página 72, onde se lê "operanting", leia-se "operating"

Página 84, na equação (27) onde se lê $T2(k)\cdot(T_s + T_A) = J_n\left(\frac{\omega_m(k) - \omega_m(k-1)}{T_s}\right) + T_A \cdot T_2(k-1)$,

leia-se $T2(k)\cdot(T_s + T_A) = J_n \cdot (\omega_m(k) - \omega_m(k-1)) + T_A \cdot T_2(k-1)$

Página 84, na equação (28) onde se lê

$$T2(k) = \left(\frac{J_n}{T_s + T_A}\right)\cdot\left(\frac{\omega_m(k) - \omega_m(k-1)}{T_s}\right) + \left(\frac{T_A}{T_s + T_A}\right)\cdot T_2(k-1)$$, leia-se

$$T2(k) = \left(\frac{J_n}{T_s + T_A}\right)\cdot(\omega_m(k) - \omega_m(k-1)) + \left(\frac{T_A}{T_s + T_A}\right)\cdot T_2(k-1)$$

Página 84, na equação (29) onde se lê $E_1(k) = \omega_{ref}(k)\cdot\omega_m(k)$, leia-se $E_1(k) = \omega_{ref}(k) - \omega_m(k)$

Página 84, na equação (30) onde se lê $E_2(k) = i_{a1}(k)\cdot i_{dis}(k)$, leia-se $E_2(k) = i_{a1}(k) - i_{dis}(k)$

Página 84, na equação (31) onde se lê $E_3(k) = E_2(k)\cdot i_m(k)$, leia-se $E_3(k) = E_2(k) - i_m(k)$

Página 86, onde se lê "o o circuito", leia-se "o circuito"

Marcelo Godoy

3 OUT / 2003

SUMÁRIO

RESUMO

Este trabalho apresenta o projeto e a implementação de um sistema baseado em processador digital de sinais para o controle digital e a supressão de distúrbios de torque em motores elétricos. O controle digital é processado em tempo real, afim de executar o código de um observador de distúrbios de torque em uma planta analógica de simulação de um motor elétrico de corrente contínua e imãs permanentes. O modelo do motor foi desenvolvido e posteriormente simulado em ambiente Matlab/Simulink. Os resultados da resposta do motor de corrente contínua, de um regulador PI e do observador de distúrbios de torque, demonstrando a supressão de ruídos injetados no motor elétrico, permitiu confirmar experimentalmente o funcionamento de todo o sistema de controle implementado.

ABSTRACT

This work presents the design and implementation of a digital signal processing based system for digital control of electrical motor with functions of supressing torque disturbance. A real-time kernel runs the algorithm. The digital control made up of a PI controller plus a disturbance torque observer is initially designed with a Matlab/Simulink environment and later fine-tuned with a dc electrical machine analog plant. Very sucessful results show the supression of disturbances, confirming the correct execution of the implemented control approach.

1 INTRODUÇÃO

Sistemas baseados em processadores digitais de sinais (DSP), estão se tornando usuais em todas as áreas em há necessidade de controle digital, pois a utilização destes processadores na automação de sistemas permite a implementação de novos enfoques de controle de motores elétricos, observadores de distúrbios, estratégias de controle robusto, identificação de sistemas; permitindo além da aplicação, o estudo de viabilidade de tais técnicas no contexto do sistema.

1.1 Sistemas baseados em processadores digitais de sinais

O fator que incentiva a crescente utilização de processadores digitais de sinais (DSP) em controle digital é a sua capacidade de processamento [1]. No caso de sistemas baseados em DSP para controle em tempo real de motores elétricos, sejam para aplicações puramente em engenharia elétrica, ou para aplicações em engenharia mecatrônica, têm exigido a utilização de DSP em seus circuitos de controle digital. Com uma preocupação, cada vez maior, para que os motores elétricos ofereçam um desempenho superior, associado aos níveis cada vez mais exigentes, para atender aos padrões do mercado atual em fatores como o consumo baixo em relação às melhorias de características como o torque. Assim torna-se imperativa a maior demanda por implementações de sistemas de controle digital através desses dispositivos.

Na área de engenharia mecatrônica, grandes esforços têm sido feito nas últimas duas décadas afim de encontrar soluções para problemas como o posicionamento e a resposta de um motor elétrico, através de métodos que eliminem, ou diminuam, os distúrbios que são normalmente encontrados nestes dispositivos eletromecânicos. A

literatura mostra que nestas duas décadas o controle digital têm sido empregado de maneira crescente [1][2][3][4][5] e mais recentemente, na última década o emprego de DSP têm sido feito com maior frequência na implementação cada vez mais complexa de sistemas de controle em tempo real [6].

O Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos da EPUSP está atualmente desenvolvendo o projeto de um motor elétrico pentafásico embutido-em-roda que é destinado à tração de veículos elétricos. Este motor, além de seu módulo inversor de potência, utiliza um sistema digital de controle. Tal sistema foi baseado em DSP para que comporte a complexidade dos cálculos exigidos em certas estratégias de controle ao qual este motor será submetido. O emprego da plataforma de um sistema baseado em DSP para o controle em tempo real de motor elétrico é o escopo deste trabalho. Para que testes e verificação do sistema sejam feitos, um esquema para diminuir, ou eliminar, os distúrbios em motores elétricos foram implementados nesta plataforma, afim de testar o sistema baseado em DSP, no controle em tempo real de um motor de corrente contínua.

1.2 Motivação e Objetivos

A motivação inicial para este trabalho foi o interesse de criar um ambiente de desenvolvimento de controladores digitais, de fácil utilização e implementação, minimizando as dificuldades e o tempo de projeto, implementação e o teste de tais controladores. A atualidade e a relevância deste assunto no contexto da automação industrial e da engenharia mecatrônica, em nossos dias, despertaram a necessidade de tal ambiente, conferindo ao trabalho características de modernidade e aplicabilidade prática.

Este trabalho descreve o projeto e a implementação de uma plataforma que permita o controle digital de motores elétricos. Visando uma estrutura flexível que permita a experimentação de estratégias de controle, é proposto um sistema baseado em processador digital de sinais [7]. A implementação deste sistema, exige o projeto de um circuito específico [8][9] voltado para o controle em tempo real de motores elétricos. Para efetuar os testes do sistema desenvolvido ao longo deste trabalho, foi necessário projetar e implementar um simulador de motor de corrente contínua, atuando como a planta a ser controlada. Neste conjunto, afim de testar o sistema, foi proposto a implementação do algoritmo de um Observador de Distúrbios de Torque (DTO) baseado na implementação analógica inicialmente desenvolvida na dissertação de mestrado "Aplicação do Observador de Distúrbios de Torque no Robô Móvel Ariel", defendida por Márcio José Chiaramonte no Depto. de Mecânica da Escola Politécnica da Universidade de São Paulo [10]. No sistema obtido, através da implementação de uma placa controladora, foi necessário preparar rotinas específicas para a utilização do sistema de controle através da linguagem C. Dessa forma, o algoritmo do observador pode ser implementado em linguagem de alto nível, facilitando a portabilidade de código e a conveniência de alteração da estrutura programada, similar ao abordado pela literatura, sobre o uso do processador digital de sinais como vantagem no controle de motor, utilizando a linguagem C para a programação dos algoritmos de controle, através de placa hospedada em microcomputador [11].

1.3 Organização do Trabalho

Este trabalho aborda a discussão de um modelo de sistema de controle baseado em processador digital de sinais, a escolha dos circuitos de apoio que permitiram sua implementação, o desenvolvimento de uma estrutura para a implementação do código,

bem como a apresentação dos resultados de sua aplicação. O escopo deste trabalho foi dividido em capítulos, cujos conteúdos foram organizados de modo à apresentar a ordem cronológica de sua execução. Resumidamente, estes capítulos são descritos a seguir:

- **Capítulo 1** – Introdução

Este capítulo apresenta a motivação para o desenvolvimento do presente trabalho, utilizando uma pesquisa direcionada aos sistema baseados em processador digital de sinais usando controle em tempo real, aplicando em motores elétricos. É feita uma breve introdução ao universo atual dos processadores digitais de sinais, os desenvolvimentos de sua utilização em eletrônica industrial e em engenharia mecatrônica, particularmente quanto ao controle de motores, resultando nos objetivos deste trabalho.

- **Capítulo 2** – Controle digital de motores elétricos

No capítulo 2 são apresentados aspectos sobre o projeto e sua implementação, sendo citadas e analisadas suas características, com base na literatura pesquisada. Dessa maneira, são explorados os conceitos básicos necessários ao desenvolvimento do trabalho, através da análise do motor de corrente contínua, suas características de projeto, seu controle e acionamento, visando a implementação do sistema de controle digital e do observador de distúrbio de torque. A utilização deste tipo de sistema, através de pesquisa desenvolvida na literatura e a implementação de observadores de distúrbio são também discutidos.

- **Capítulo 3** – Sistema de controle em tempo real

Uma análise de sistemas de controle em tempo real, através da definição da implementação de um sistema operacional e seu núcleo, suas características e seus requisitos, são abordados neste capítulo.

- **Capítulo 4** – Sistema baseado em processador digital de sinais

Neste capítulo é desenvolvido e apresentado o projeto do sistema baseado em processador digital de sinais, no que diz respeito ao tipo de processador escolhido, aos circuitos de apoio, à capacidade e aos parâmetros do sistema que foi implementado. Os códigos e rotinas utilizados pelo sistema são apresentados e discutidos, baseados na placa controladora que foi implementada.

- **Capítulo 5** – Implementação do observador de distúrbios

O capítulo 5 trata o projeto e o modelo, e sua posterior implementação, de um observador de distúrbios de torque, para a eliminação dos efeitos dos distúrbios de carga aos quais o sistema de controle está sujeito. São apresentados os resultados dos ensaios realizados com o controle em tempo real desenvolvido no sistema baseado em processador digital de sinais, mostrando a redução ou eliminação do ruído quando o controle do motor de corrente contínua é executado.

- **Capítulo 6** – Considerações finais

Neste último capítulo são feitas, uma análise do trabalho desenvolvido e dos resultados obtidos. As discussões finais sobre o trabalho, com base nos resultados alcançados, sugerem propostas de continuação deste trabalho.

Tendo o objetivo de tornar o texto o mais claro e auto-suficiente possível, inclusive para aqueles que gostariam de obter mais informações, 6 apêndices foram incorporados ao texto.

Apêndices – Os 6 apêndices no final do presente trabalho, apresentam:

1. rotinas desenvolvidas para o sistema monitor;
2. circuitos elétricos do sistema baseado em DSP;
3. programa para descarregar código na memória de acesso dual;
4. circuito da planta analógica do motor de corrente contínua;

5. modelos desenvolvidos em MatLab/Simulink, e
6. código do controlador digital.

2 CONTROLE DIGITAL DE MOTORES ELÉTRICOS

2.1 Introdução

Técnicas de controle digital baseadas em processadores são requisitos para uma supervisão mais abrangente visando a necessidade da melhoria do desempenho nos sistemas de controle de motores. Para a aplicação prática das técnicas de controle avançado, como as que são baseadas na teoria de controle moderno, devido à complexidade encontrada nos algoritmos sempre existiam obstáculos quando da implementação em microprocessadores mais antigos. A quase maioria dos estudos deste tipo de controle limitavam-se ao estudo de sua simulação em computadores [12]. O desenvolvimento de novos processadores nos últimos 15 anos, nos permite agora a aplicação direta de sistemas sofisticados no controle de motores. É notório que um tipo específico de processador, neste caso o processador digital de sinais, tem uma grande contribuição neste avanço na área de controle de motores.

2.2 Sistemas baseados em DSP

O controle baseado em DSP tem sido aplicado de forma exponencial na última década, principalmente devido ao aumento do desempenho nestes dispositivos e a sua maior variedade de opções, no mercado. Com tal popularização do DSP, a área de controle de motores é uma das mais beneficiadas com a sua utilização.

São inúmeros, os autores na literatura que têm utilizado DSP no controle de motores, sejam através de sistemas comerciais ou de projetos específicos desenvolvidos para o controle de um determinado motor ou de uma categoria de motores

[11][13][14][15][16][17][18][19][19][21]. Em fins da década de 80 e ínicio da década de 90, já se explorava a utilização destes processadores, no controle de motores ou em sistemas de controle em tempo real [22][23], época em que os processadores de uso geral, de 8 bits, estavam fixando-se como padrão, sendo largamente utilizados no controle microprocessado de motores [24], devido ao seu baixo custo. Apesar do avanço do controle microprocessado de maneira geral, muitos controladores tanto na área acadêmica, como na área industrial, eram implementados parcialmente de maneira digital e parcialmente de maneira analógica [25][26]. Mesmo no que tange à utilização do DSP, há 5 anos atrás, a maioria restringia-se aos processadores de 32 bits, de ponto flutuante, graças à sua maior capacidade computacional [27]. Os microprocessadores de uso geral, particularmente os processadores utilizados nos computadores IBM-PC e seus compatíveis foram e continuam sendo dispositivos de grande aceitação, pela sua facilidade de programação e quantidade de material de apoio disponível, como ferramentas de depuração e linguagem de programação. Com o aparecimento dos processadores de 32 bits, particularmente do tipo Intel i486, algoritmos mais complexos, e portanto que exigem maior poder computacional, foram implementados neste processador [28] além de processadores da familia Motorola 68000 [29]. Uma exceção a hegemonia do processador de 32 bits, foi o aparecimento de projetos utilizando processadores digitais de sinais de 24 bits, da familia DSP56000 da Motorola [22][23].

Na maioria dos casos, quando os projetistas faziam uso de processadores de menor capacidade, como os processadores de 16 bits, normalmente valiam-se de um segundo processador, menor, de 8 bits, de apoio para funções específicas, como p.ex. a geração do sinal de controle modulado por largura de pulso, deixando assim o processador central livre para os cálculos mais pesados. Utilizando processadores

menores e com menor desempenho, o controle de motores em tempo real era possível de realizar, porém muitas vezes suas malhas de controle eram da ordem de várias dezenas e às vezes até centenas de milissegundos [12][30], mais lentos do que os atuais controles, implementados com malhas da ordem de algumas centenas de microssegundos, às vezes não chegando na faixa dos milissegundos. Recentemente, estão chegando ao mercado processadores digitais de sinais com funções de geração de modulação por largura de pulso, conversão analógica-digital, já disponíveis entre seus periféricos internos, na mesma pastilha. Até então, certas funções como a geração da modulação por largura de pulso, eram implementadas através dos seus contadores/temporizadores internos ou de unidades específicas, externas ao DSP conhecidos como *Timer Processing Unit* (TPU) e muito utilizados como dispositivos de apoio aos processadores e microcontroladores. Estes novos processadores digitais de sinais, de 16 bits em sua maioria, voltados a aplicações envolvendo o controle de motores e produtos de consumo, possuem um alto desempenho computacional, principalmente quando comparados com os microcontroladores de 16 bits atualmente disponíveis no mercado.

Em aplicações de controle digital para aplicações de eletrônica de potência, p.ex., uma pequena melhoria ou alternativa durante o projeto do acionamento pode representar um desempenho melhor do sistema. Devido a este fato, a utilização do controle digital permite uma facilidade intrínseca, que muitas vezes não é possível, ou até mesmo não realizável, em circuitos de controle contínuo, ou analógico. No trabalho de comparação realizado no artigo [17], o autor mostra exatamente onde um detalhe de uma aplicação permite modificar o desempenho de todo um sistema. Nesse artigo, os autores exploram o fato de que a característica da utilização de conversores de modulação por largura de pulso, para aplicações convencionais em acionamento de motores resultam em perdas

no chaveamento, o que pode resultar em um problema a ser considerado, quando trabalha-se com dispositivos de alta potência. Neste caso, segundo os autores, controladores digitais podem ser projetados de maneira a implementarem diferentes padrões de chaveamento de modulação por largura de pulsos, sendo estes padrões projetados de forma otimizada. Segundo o artigo, resultados obtidos através de algoritmos preditivos, implementados em processadores digitais de sinais, para o mesmo sistema de acionamento, externo ao controlador digital nos diversos casos estudados, permitiu uma melhora na resposta do sistema resultando na redução dos componentes dos filtros e portanto, obtendo um desempenho mais alto no sistema como um todo. Aplicações de controle preditivo e adaptativo entre outros, são grandes vantagens dos sistemas de controle digital, principalmente os baseados em processador digital de sinais, devido a uma boa relação Custo x Benefício, pois a facilidade de alterar a estratégia de controle, com a simples mudança do algoritmo ou até mesmo de seus parâmetros, permite muitas vezes a utilização do mesmo equipamento em sistemas diferentes.

O controle de motores de corrente contínua, normalmente está baseado na realimentação da informação de velocidade, proveniente de um sensor que pode ser do tipo tacogerador ou do tipo encoder. Alternativas para aplicações de controle sem a utilização de sensores de velocidade, têm sido bastante exploradas, através da utilização de um modelo de referência ou da identificação dinâmica do motor que está em uso, implementado em algoritmo no próprio processador digital de sinais que está gerando o controle, através da técnica de Controle Adaptativo por Modelo de Referência (MRAC, Model Reference Adaptative Control). O controle digital adaptativo é uma área extremamente fascinante que devido à sua abrangência e desmembramento em sub-

áreas, (todavia transcendendo os objetivos deste trabalho) pode ser objeto de um estudo mais detalhado na utilização deste sistema.

Uma versão simplificada do modelo da máquina foi necessária, para o desenvolvimento do algoritmo de controle adaptativo da velocidade; este modelo, é construído e baseado de tal maneira, que assumindo a carga como um torque de distúrbio simples, a dinâmica mecânica do sistema pode ser representada como uma equação diferencial de segunda ordem. Tal representação é similar à representação convencional de um motor de corrente contínua, com o qual iremos trabalhar a partir de agora.

2.3 Motores de Corrente Contínua

Motores de corrente contínua continuam sendo largamente aplicados em robôs, na indústria, em produtos de consumo e na área automobilística, apesar dos notáveis avanços nos outros tipos de motores elétricos e seus respectivos controles digitais. Sua popularidade, reside na facilidade do controle de velocidade do eixo do rotor do motor, através apenas da variação da tensão aplicada nos terminais da bobina de armadura deste motor. Um motor de corrente contínua possui basicamente dois circuitos elétricos: o circuito do campo e o circuito da armadura.

O circuito de campo está localizado na parte estacionária, ou estator, da máquina, consistindo de enrolamentos ao redor dos pólos magnéticos do estator. Estes enrolamentos criam, ao circular corrente num sentido, pólos norte e sul na máquina. O circuito de campo tem o propósito de criar o fluxo magnético no entreferro do motor entre o estator e o rotor. A corrente fornecida no circuito do campo serve para criar e manter o fluxo magnético. Com a utilização de imãs permanentes nos pólos magnéticos, o circuito de campo não é necessário.

O circuito de armadura é o circuito de potência do motor de corrente contínua. Quando o enrolamento da armadura é percorrido por uma corrente, uma força é gerada a partir da interação entre a corrente e o fluxo magnético do campo. Como a corrente atravessa toda o enrolamento, uma força igual e contrária é gerada em cada lado do enrolamento e juntos, produzem o torque do motor.

2.3.1 Circuito elétrico equivalente do motor de corrente contínua

As equações elétricas de um motor de corrente contínua, equacionadas a partir do seu circuito elétrico equivalente, são descritas a partir do modelo da Fig. 1.

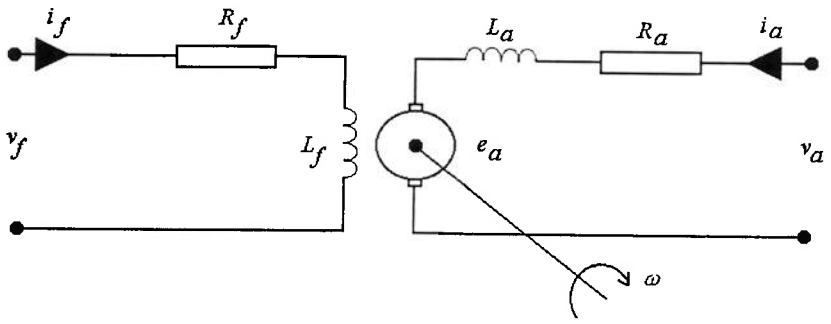


Fig. 1: Circuito elétrico equivalente do motor de corrente contínua

A equação da tensão de armadura é dada por:

$$v_a = r_a \cdot i_a + l_a \frac{di_a}{dt} + e_a \tag{1}$$

onde

v_a tensão nos terminais da armadura

r_a resistência de armadura

i_a corrente que circula na armadura

l_a indutância da armadura

e_a f.c.e.m induzida na armadura

A equação da tensão nos terminais, no circuito, da bobina de campo, pode ser escrita como:

$$v_f = r_f \cdot i_f + l_f \cdot \frac{di_f}{dt} \tag{2}$$

onde

v_f tensão nos terminais do enrolamento de campo

r_f resistência do enrolamento de campo

i_f corrente que circula no enrolamento de campo

l_f indutância do enrolamento do campo

Lembrando que sendo i_f = constante = fluxo constante, o que resulta na constante da força contra-eletromotriz (constante da velocidade), k_E. Como:

$$e_a = k_E \cdot \omega \tag{3}$$

a (3) em (2), resulta em:

$$v_a = r_a \cdot i_a + l_a \frac{di_a}{dt} + k_E \cdot \omega \tag{4}$$

Se a constante k_E é fornecida com a unidade V/rad.s^{-1}, o valor numérico de k_E é igual à constante de torque k_T, apesar de serem suas unidades diferentes, sendo a unidade de k_T N.m/A. Daí, a equação do torque elétrico pode ser escrita como:

$$T_e = k_T \cdot i_a \tag{5}$$

sabe-se que:

$$(T_e - T_L) = J \frac{d\omega}{dt} \tag{6}$$

logo,

$$\frac{di_a}{dt} = \frac{1}{l_a}\left[v_a - r_a \cdot i_a - k_E \cdot \omega\right] \tag{7}$$

e

$$\frac{d\omega}{dt} = \frac{1}{J}\left[k_T \cdot i_a - T_L\right] \tag{8}$$

2.3.2 Modelo do motor de corrente contínua

Baseado nas equações (7) e (8) apresentadas no ítem anterior, foi construído um modelo (Fig. 2) do motor de corrente contínua com bobina de campo. Neste modelo, variando-se a entrada com um nível de tensão de armadura, v_a, obtém-se como resultado o sinal de velocidade ω. Considerando a tensão aplicada nos enrolamentos da bobina de campo como sendo fixa ($v_f = c^{te}$), podemos substituir o modelo com bobina de campo apresentado pelo modelo modificado do motor de corrente contínua para um motor do tipo de imãs permanentes, onde e_a é proporcional à velocidade do motor ω, através da constate elétrica do motor k_e. A somatória dos distúrbios existentes num motor elétrico mais uma perturbação aplicada no motor, é inserida como T_L em ambos os modelos do motor. Utilizando imãs permanentes nos pólos magnéticos do motor, o diagrama de blocos do modelo do motor é modificado, para um motor de imãs permanentes, conforme mostrado na Fig. 3.

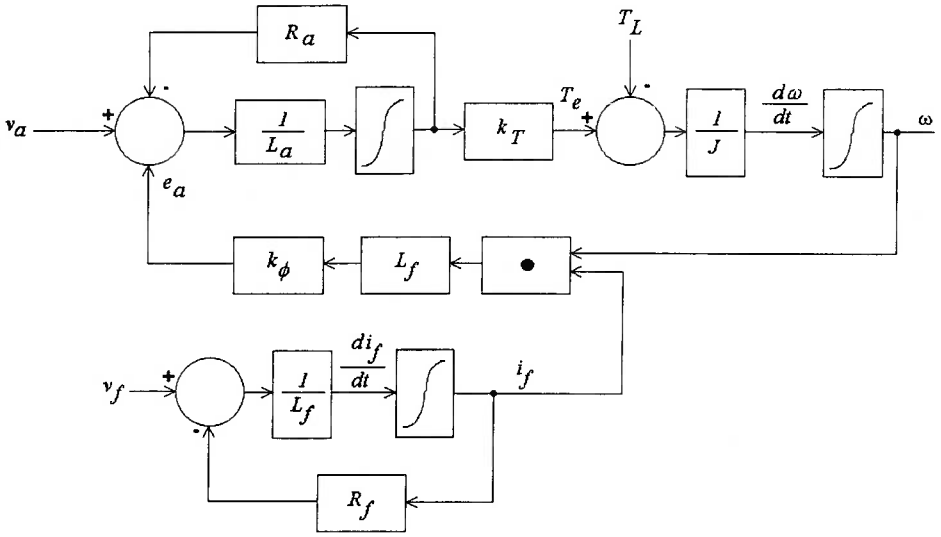


Fig. 2: Diagrama de blocos do modelo do motor de corrente contínua com bobina de campo

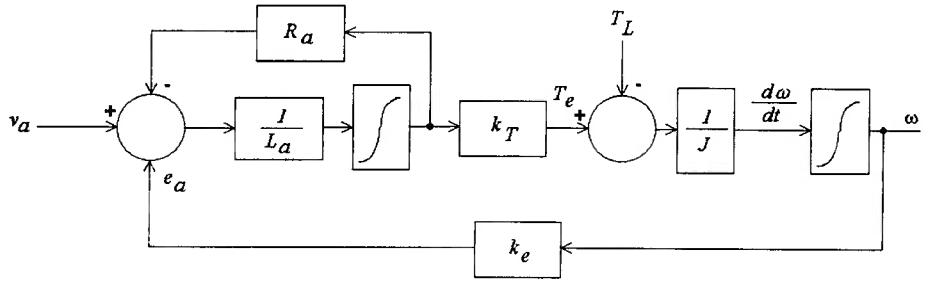


Fig. 3: : Diagrama de blocos do modelo do motor de corrente contínua de imãs permanentes

2.4 Acionamento de Motores

O motor de corrente contínua é por sua natureza reversível, podendo passar do funcionamento como um motor ao funcionamento como um gerador. Utilizando nosso circuito elétrico do motor e supondo que conectamos um motor ideal, sem perdas, à uma tensão contínua v_a o motor gira, com um sentido e uma velocidade de tal modo a gerar um f.e.m. e_a igual e de sinal oposto a v_a. Em regime permanente, sendo $v_a = e_a$, não há absorção de corrente. O torque do motor nesse caso portanto é nulo (motor ideal). Agora, se for aplicado um torque resistente T_L ao motor, o motor tenderá a tornar-se mais lento. a f.e.m. e_a diminuirá, e a diferença v_a-e_a dará lugar a uma corrente

de armadura i_a, que reagindo com o fluxo produzirá um torque motor T_m. Nesse caso, a máquina desenvolve um torque motor, com a energia passando da fonte primária de tensão à carga. Se agora, ao invés de aplicarmos um torque resistente, aplicarmos um torque positivo, que tende a fazer o motor girar no mesmo sentido, o motor tenderá a acelerar, de tal maneira que $e_a > v_a$. Nessas condições i_a inverte de sinal, da mesma maneira que o torque (considerando o fluxo permanecido constante, num motor de imãs permanentes). Este torque tenderá a frear o motor, opondo-se ao torque externo, que tende a acelerá-lo. Neste caso, a máquina funciona como freio e como gerador. Dessa maneira, a energia passa da carga à fonte primária de tensão. De maneira resumida, foi mostrado a atuação da máquina, como motor e gerador, para quatro quadrantes. É necessário lembrar que, para o motor operar nestas quatro condições, a fonte primária de tensão onde o motor está conectado deve permitir tanto fornecer como receber corrente. Com o objetivo de testar o funcionamento da máquina como motor de corrente contínua, um sistema de acionamento de torque reversível (regenerativo) não será aplicado, sendo analisado apenas o acionamento de torque unidirecional (não-regenerativo).

Em um acionamento analógico tradicional, a velocidade desejada do motor é representada pela entrada da tensão de referência V, onde $V_{\min} \leq V \leq V_{máx}$. Dessa maneira, a velocidade máxima em um sentido de rotação é representada por V_{max} e a velocidade máxima no sentido inverso de rotação é representada por V_{min}. Para V = 0 o motor está parado, e tensões intermediárias representam velocidades proporcionais à tensão aplicada.

Em um acionamento de torque unidirecional, a energia pode fluir em um único sentido, da rede, ou barramento, ao motor. Nesse caso, o motor pode funcionar nos 1º e 3º quadrantes, observando que a passagem entre um quadrante para o outro deve,

portanto, ser executada com o motor parado, para evitar uma passagem temporária no 2º ou 4º quadrantes em que o motor não pode funcionar e por consequência, inutilizar o circuito de acionamento. Para evitar tais acontecimentos, é muito comum os acionamentos possuirem circuitos de atraso no chaveamento dos braços de comutação e faz-se a utilização de diodos de *free-wheeling* para descarga da corrente na indutância de armadura. Um acionamento de topologia do tipo meia-ponte (ou tipo T) para um motor de corrente contínua é mostrado na Fig. 4. Nesta topologia de acionamento, nos pontos C1 e C2, são injetados sinais de controle modulados por largura de pulso, que operam os dispositivos de potência (transistores do tipo bipolar) como chaves liga/desliga. Utilizando-se este tipo de acionamento por modulação da largura do pulso do sinal de controle, evitam-se as perdas normalmente associadas aos acionamentos do tipo linear, devido ao funcionamento contínuo das chaves em sua região linear. O amplificador linear pode utilizar esta mesma topologia de meia-ponte, como acionamento. Atualmente é muito comum, para acionamentos de até 100A, utilizar módulos integrados, que já incluem este tipo de topologia em sua pastilha e ainda permitem, nas entradas C1 e C2 de sinais de controle, a utilização de níveis lógicos compativeis ao padrão TTL, permitindo assim o acoplamento direto com microprocessadores, atuando como seu sistema de controle. É comum estes dispositivos também oferecerem acopladores óticos, que permitem isolar o circuito de controle microprocessado, do circuito de acionamento de potência. Nesta topologia, as chaves são alimentadas por fontes bipolares, ou simetricamente por +Vcc e –Vcc, permitindo desse modo realizar a inversão do sentido da rotação do motor. Uma outra topologia, permite trabalhar com apenas uma tensão no barramento de alimentação do circuito de acionamento do motor, de tal maneira que permite a inversão do sentido de rotação. Isso é possivel utilizando-se duas meia-pontes em conjunto, formando a topologia conhecida

por ponte-H, conforme mostrado na figura Fig. 5, verifica-se a necessidade do dobro de chaves. Neste exemplo as chaves são transistores do tipo bipolar, outra chave muito comum é a do tipo IGBT (Insulated Gate Bipolar Transistor), que reune as vantagens dos dispositivos MOSFET às vantagens de alta capacidade de potência do transistor bipolar. No exemplo da ponte-H apresentada na figura, foi incluído um circuito lógico que oferece proteção contra o acionamento simultâneo de chaves complementares, evitando o curto-circuito. Também pode-se observar que os diodos de *free-wheeling* que permitem a extinção da energia residual no enrolamento do motor, foram incluídos neste circuito da ponte-H. Os pontos C1 e C2 recebem o sinal com a modulação por largura de pulso, referente ao valor de acionamento do motor, e o sinal E que permite habilitar ou desabilitar o acionamento do motor de corrente contínua, a partir do sistema de controle.

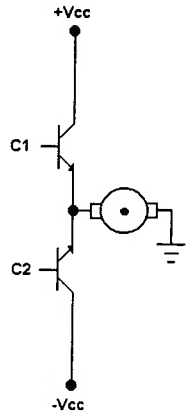


Fig. 4: Acionamento em topologia meia-ponte (ponte T)

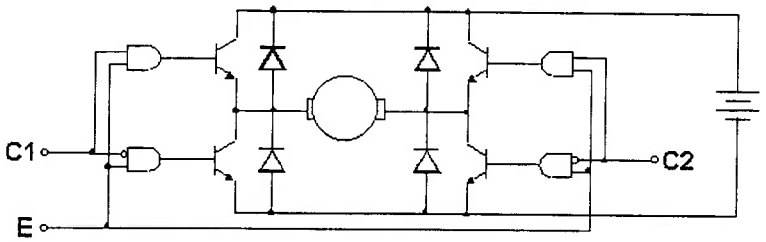


Fig. 5: Acionamento em topologia ponte completa (ponte H)

2.5 Controle de Motores

Um motor de corrente contínua com enrolamento de campo, pode operar nos modos de controle de campo e controle de armadura, um motor de corrente contínua de imãs permanentes, pode operar apenas no modo de controle de armadura. Geralmente, o controle de campo oferece menos torque e uma resposta mais lenta, porém ele utiliza menor potência da fonte de alimentação. Em contraste, o controle de armadura necessita um maior consumo de potência, porém em retorno ganha-se um maior torque de saída e uma resposta mais rápida. Além disso, o próprio projeto da armadura pode ter um efeito significativo na velocidade da resposta e na habilidade do motor operar suavemente em baixas velocidades, sem ocorrência de sobressaltos. No motor de corrente contínua, a constante de tempo mecânica é diretamente proporcional à inércia da armadura. Como exemplo, o momento de inércia para um cilindro sólido é $\frac{1}{2} \cdot M \cdot R^2$ sendo possível, portanto, reduzir a inércia do cilindro da armadura reduzindo seu diâmetro ou reduzindo seu peso, ou ambos. Diminuindo-se a inércia, a constante de tempo mecânica deve diminuir permitindo ao motor uma resposta mais rápida às mudanças bruscas na entrada. O motor de corrente contínua de imãs permanentes disponível comercialmente, inclui no projetos recursos como o discutido acima de maneira a possuir uma alta taxa de torque-inércia, permitindo que inicialize e pare rapidamente, com um alto torque inicial e operando sem sobressaltos, nas baixas velocidades. Estes recursos do torque, fazem do motor de corrente contínua e imãs permanentes ser ideal para aplicações de controle [31].

A demanda de torque do amplificador de velocidade vem da necessidade de mais corrente no motor. O controle da corrente é conseguido através de uma malha de realimentação que compara o torque com a corrente no motor. Esta corrente é

sensorizada por um resistor R, que produz uma tensão proporcional à corrente do motor, ou por um sensor de efeito Hall. Esta malha interna de realimentação é frequentemente conhecida como amplificador de torque uma vez que seu objetivo é criar torque em resposta à demanda a partir do amplificador de velocidade. O amplificador de torque pode ser usado como a base de um servo acionamento. Alguns tipos de controle de posição geram um sinal de torque na saída no lugar da velocidade demandada e há também aplicações na qual o torque, e não a velocidade, é o interesse primário. Na prática, o sinal de entrada é sempre obtido no mesmo ponto, mas o amplificador de velocidade é desconsiderado.

Em um circuito de regulagem de um motor, é comum considerarmos:

- Que a velocidade do motor corresponda o máximo possível ao valor de referência;
- A corrente absorvida pelo motor não supere um valor máximo, pré-definido, durante a aplicação de dado um degrau na referência;

Essa dupla consideração pode ser alcançada através de:

- Uma regulagem de malha simples, de velocidade, tendo porém um circuito que limite a corrente que intervem, mudando a referência quando a corrente tende a superar o valor pré-determinado.
- Uma regulagem com duas malhas, uma mais interna, de corrente, e outra mais externa, de velocidade (Fig. 6).

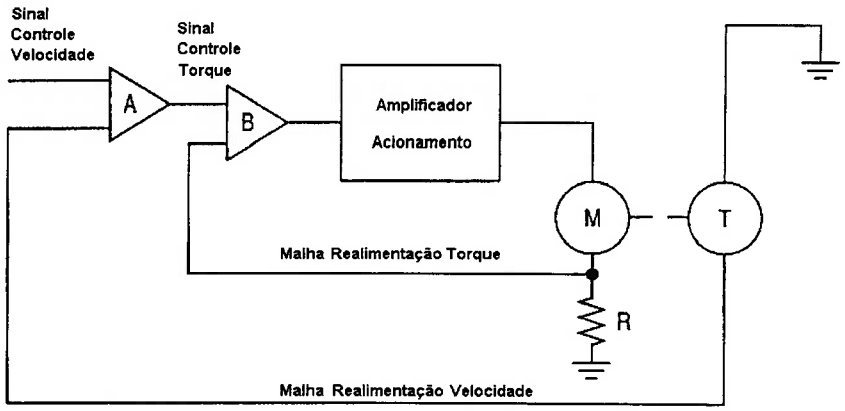


Fig. 6: Controle analógico da velocidade de um motor de corrente contínua

Numa malha de controle, a situação clássica mais comum para o motor de corrente contínua é o controle através da realimentação do seu sinal de velocidade na saída, onde a referência é o próprio valor desejado no eixo do motor. A realimentação do valor instantâneo, obtido no eixo do motor fornece o sinal de erro para o controlador, conforme ilustra a Fig. 7. O controlador mais utilizado em controle de motores é o regulador PI (Fig. 8).

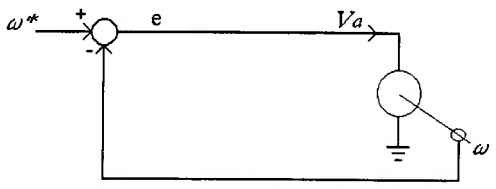


Fig. 7: Malha de Velocidade

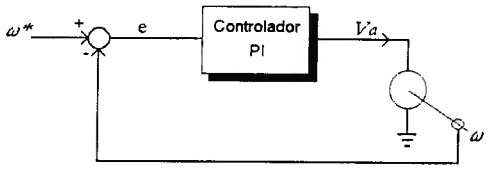


Fig. 8: Malha de Controle PI

Durante seu funcionamento, o controle do motor de corrente contínua pode ser comprometido devido aos distúrbios de torque, que são consequências de forças externas como a força da gravidade, atritos, força de Coriolis e força centrífuga. Os distúrbios provocam alterações na dinâmica de sistemas de acionamento o que resulta em erros de resposta aos comandos de controle. Uma compensação do distúrbio de torque torna um acionamento robusto em relação as alterações na carga e nos torques não modelados. Para realizar a compensação, o sistema de controle do acionamento, precisa conhecer o torque de distúrbio. Infelizmente, este valor não pode ser facilmente medido mas pode-se estimar o torque de distúrbio, conhecendo a velocidade do motor [32], sendo esta variável já utilizada pela malha do controlador PI, utilizando o modelo desenvolvido no ítem 2.3.2. A Fig. 9 apresenta um diagrama simplificado de um estimador do distúrbio de torque utilizando a velocidade e a corrente de armadura do motor de corrente contínua apresentado. Porém, as técnicas para aplicar controle robusto exigem do sistema de controle digital complexas rotinas computacionais [10], devido à grande quantidade de cálculos envolvidos como multiplicação de matrizes, função exponencial, dentre outras, culminando na escolha de um DSP para sua implementação [23].

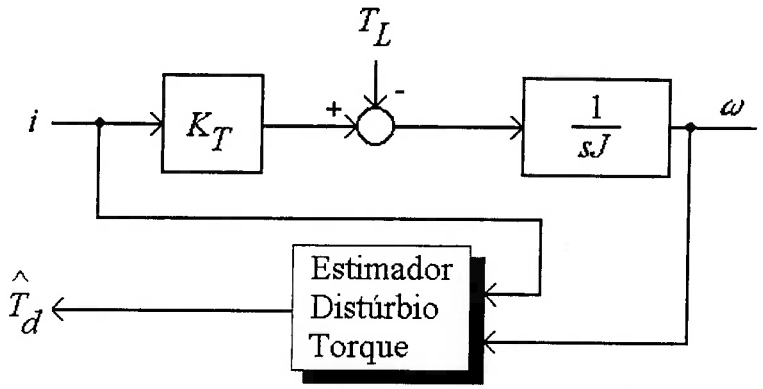


Fig. 9: Estimação do distúrbio de torque

2.5.1 Controlador PI

Os recursos principais do controlador PI são sua habilidade em manter o erro estático em zero para um degrau na referência, num sistema de ordem zero e sua simples e direta implementação em microprocessadores [30]. A equação da resposta de um controlador PI é dado por:

$$y(t) = K_P x(t) + K_I \int x(t)dt \tag{9}$$

Onde x(t) é a entrada e y(t) a saída do controlador.

Na Fig. 10 é apresentada a implementação de um controlador PI, tendo então uma parte multiplicativa proporcional P e uma parte integrativa que é multiplicada pelo ganho I, atuando no sinal de entrada, que somadas ao final resultam no sinal de saída. No nosso caso o sinal de entrada *x(t)* é o sinal de erro, *e(t)*, resultante da subtração do sinal de referência para a velocidade do sinal da velocidade medida, realimentado, apresentado na Fig. 8, e o sinal de saída *y(t)* é a tensão de controle, do enrolamento da armadura do motor.

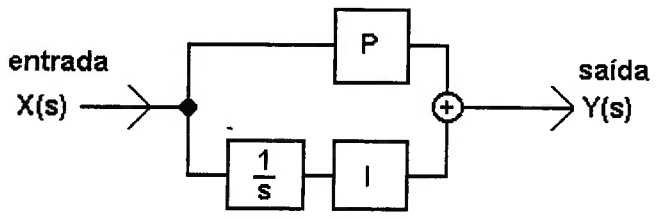


Fig. 10: Diagrama de blocos do controlador PI

O termo proporcional é de aplicação direta e não gera dúvidas, sendo portanto que o ganho P amplifica o sinal de erro pela quantidade constante em P. Já o termo integral, fornece ao sistema realimentado o ajuste da quantidade da integral do sinal de erro e

qualquer valor de I diferente de zero, implica que nenhum erro em regime permanente pode ser tolerado pela malha de realimentação do filtro PI.

2.5.2 Controle Digital

A Fig. 11 mostra os componentes de um acionamento digital de um servo motor. Todas as funções de controle principais são tratadas pelo processador, que comanda um conversor digital-analógico para produzir um sinal analógico do torque demandado. A partir deste ponto, o acionamento é muito parecido com o de um amplificador analógico. A informação de realimentação em um controle digital, é na maioria das vezes, derivada de um encoder conectado no eixo do motor. O encoder gera um trem de pulsos a partir de onde o processador pode determinar o ângulo percorrido e através do cálculo da frequência do pulso é possível medir a velocidade. O acionamento digital executa as mesmas operações que seu similar analógico, porém faz isso através da operação de uma série de equações. O processador é programado a partir de um modelo matemático e um algoritmo é gerado, ou através de seu equivalente sistema analógico. Este modelo prediz o funcionamento do sistema. O tempo de uma malha de controle é definido pelo tempo que o processador necessita para executar todas as operações, situando-se tipicamente entre 100 μs e 2 ms. Durante este tempo, o valor de controle deve permanecer constante no valor previamente calculado e não acontecerá uma resposta para uma mudança na entrada ou saída até que uma nova amostragem seja feita e novos cálculos executados. Este tempo entre a amostragem atual e a anterior é o período de amostragem T_S, e portanto, é um fator crítico no desempenho de um controle digital devendo ser definido como parâmetro importante do sistema e utilizado para o cálculo das equações.

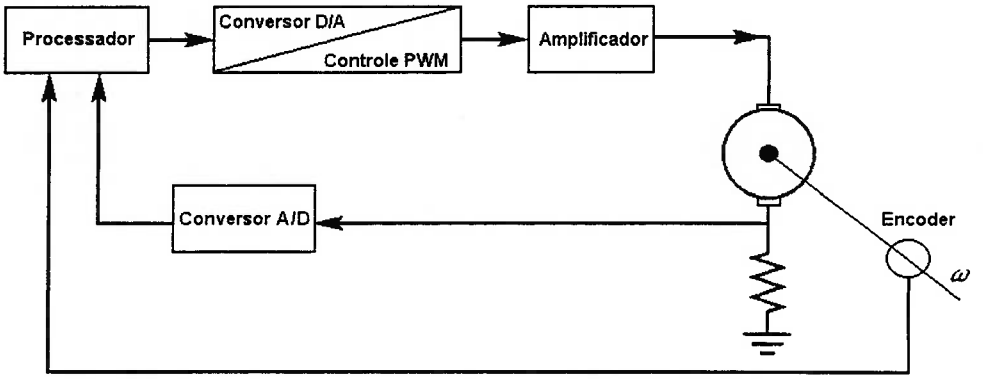


Fig. 11: Controle digital de motor de corrente continua

Para implementar o algoritmo de controle PI em um processador digital, a função de transferência do controlador é desenvolvida da mesma maneira que na versão para o tempo contínuo (analógico). Da tabela de transformadas Z, obtem-se a transformação para o integrador:

$$\frac{1}{s} = \frac{z}{z-1} \tag{10}$$

Desta forma, utilizando a aproximação trapezoidal para o integrador (Aproximação de Tustin) o controlador implementado possui a seguinte equação:

$$V_C(z) = \left(K_I \cdot \frac{T}{2} \cdot \frac{\left(1+z^{-1}\right)}{\left(1-z^{-1}\right)} + K_P \right) \cdot E(z) \tag{11}$$

Onde:

T é o período da amostragem;

K_I é o ganho da parte integrativa;

K_P é o ganho da parte proporcional

V_C é a saída com a tensão de controle

A partir da (11), podemos gerar a equação de diferença do controlador PI, para implementação computacional, da seguinte maneira:

$$v_c(n) = v_c(n-1) + \left(\frac{KI}{2} \cdot T - K_P\right) \cdot e(n-1) + \left(\frac{K_I}{2} \cdot T + K_P\right) \cdot e(n) \quad (12)$$

Onde n é o estado presente, que está sendo amostrado e n-1 é o estado da amostragem anterior. Isso faz com que as variáveis do estado anterior sejam armazenadas pelo sistema de controle digital, para efetuar o cálculo do novo valor da tensão de controle v_c, que deve ser imposto ao motor de corrente contínua (planta).

2.5.3 Amplificadores

2.5.3.1 Amplificador Linear

O amplificador linear opera de tal maneira que, dependendo da direção de rotação do motor, tanto TR1 ou TR2 estão em série com o motor com uma queda de tensão sobre eles (Fig. 12). Esta característica é a primeira limitacão ao uso de amplificadores lineares, uma vez que sempre há energia dissipada nos estágios de saída do amplificador. Para dissipar este energia, grandes transistores e dissipadores serão necessários, fazendo tal tipo de amplificador inviável para uso em sistemas de alta potência. Entretanto o amplificador linear oferece o beneficio de gerar pequeno ruído elétrico.

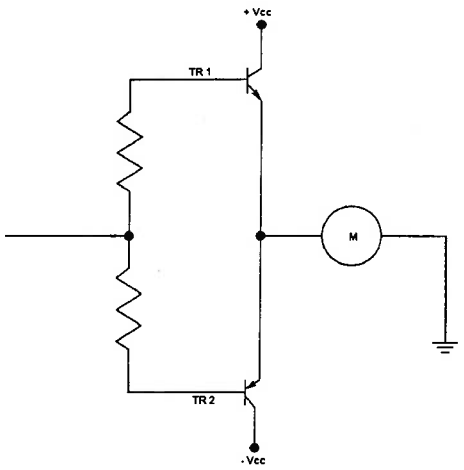


Fig. 12: Amplificador Linear

2.5.3.2 Amplificador PWM

O amplificador chaveado é o tipo mais comumentemente usado, devido aos seus baixos requisitos de potência, e o método que é usado para o controle da saída é o de modulação por largura do pulso (PWM), conforme mostra a Fig. 13. A dissipação de energia é drasticamente reduzida uma vez que os transistores estão num estado "ligado" (T_{ON}) ou "desligado" (T_{OFF}). No estado "desligado", nenhuma corrente é conduzida e portanto nenhuma energia é dissipada. No estado "ligado" a tensão através dos transistores é muito baixa (1~2 V), fazendo com que a quantidade de energia dissipada seja pequena.

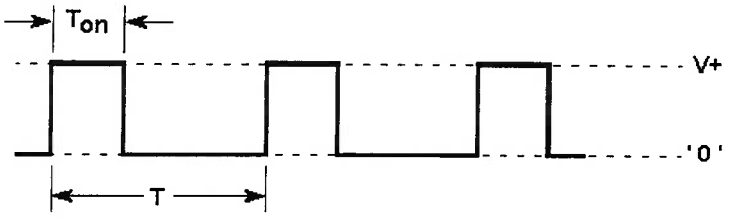


Fig. 13: Modulação por largura do pulso

2.5.4 Amplificador de Tensão

Um motor de corrente contínua pode ter, ou um estágio de amplificação de tensão, ou de corrente. O amplificador de tensão pode alimentar o motor com tensão contínua linear, operando da maneira explicada no item 2.4, sobre o acionamento analógico tradicional. O amplificador de tensão pode ser utilizado para alimentar o motor de maneira pulsada, com a tensão fixa, através da modulação da largura de pulso, definida pela razão cíclica de acionamento, através de um circuito PWM ou acionado através de um processador. Neste caso, a tensão de controle aplicada ao motor pode ser determinada pela equação (13) [10]. A Fig. 14 apresenta o diagrama simplificado de um amplificador de tensão, com ganho A_v e a Fig. 15 apresenta seu diagrama de blocos quando implementado com o modelo do motor.

$$(V_a)_{médio} = \frac{1}{Ts}\int_0^{T_S} V_a(t)dt \tag{13}$$

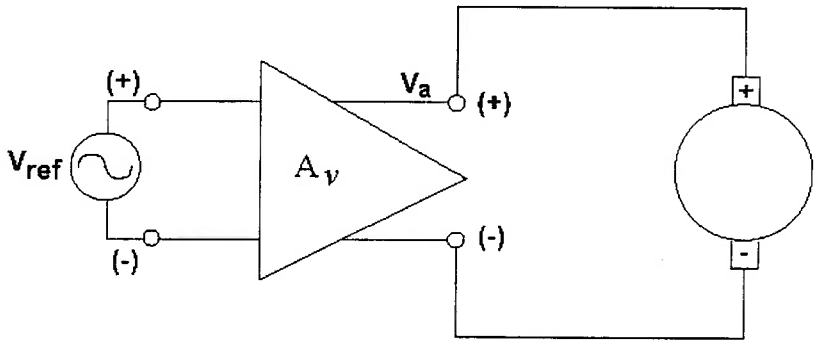


Fig. 14: Diagrama simplificado de um amplificador de tensão

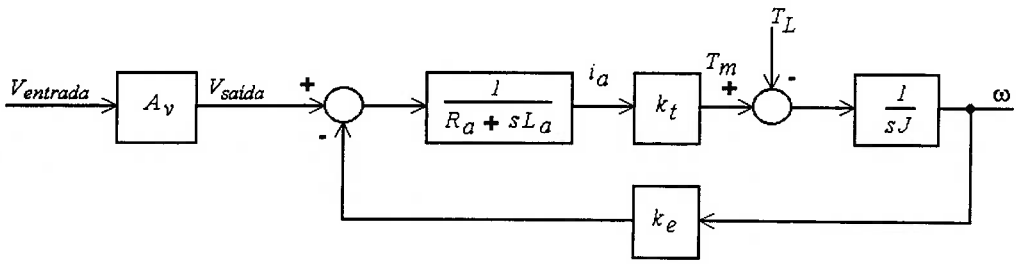


Fig. 15: Diagrama de blocos do modelo do motor com o amplificador de tensão

2.5.5 Amplificador de Corrente

Outra alternativa ao acionamento de motores é o amplificador de corrente, como mostra o diagrama simplificado da Fig. 16.

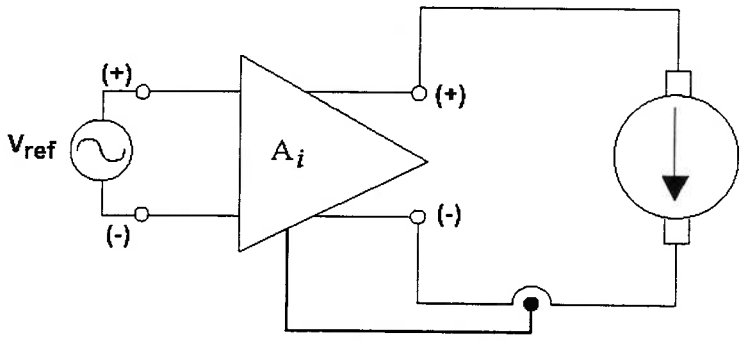


Fig. 16: Diagrama simplificado de um amplificador de corrente

Na Fig. 17, é apresentado o diagrama de blocos de um amplificador de corrente para o modelo do motor de corrente contínua do ítem 2.3.2.

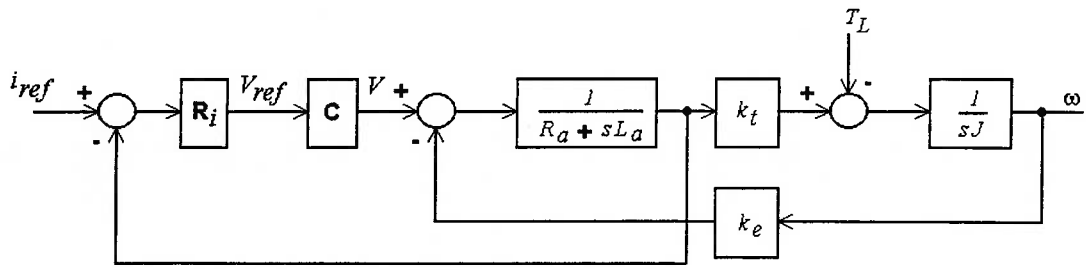


Fig. 17: Diagrama de blocos do modelo do motor com o amplificador de Corrente

Segundo CHIARAMONTE (1993) "Em aplicações onde é necessário controlar a velocidade do sistema de acionamento, a opção mais adequada é a utilização de malhas de controle de tensão"

Com os dados apresentados até o momento, constata-se que para implementar um controle digital de motor de corrente contínua, deve ser considerada a dinâmica do sistema do motor em questão, no momento da parametrização de um sistema de controle baseado em processador digital de sinais, de modo a garantir o sucesso deste sistema de controle; ficando evidente que a realização de uma simulação do modelo do motor facilitará na escolha dos parâmetros e a discussão da melhor estratégia para o algoritmo de controle.

A avaliação inicial – baseada na modelagem do motor e na literatura pesquisada - é de que um controlador do tipo PI atende as necessidades de controle, de um motor de corrente contínua. Outro fator que deve ser salientado é que independente da escolha da utilização de um amplificador de tensão, ou de corrente, no acionamento deste motor, o surgimento de distúrbios de torque não modelados, interferem no sistema, influenciando o controle, de maneira geral, o que sugere a aplicação de uma estratégia de controle que inclua uma "robustez" ao sistema e permita a diminuição ou eliminação destes distúrbios.

3 SISTEMAS DE CONTROLE EM TEMPO REAL

3.1 Introdução

Em um sistema de controle analógico, pode-se observar que o mesmo já opera como um controle em tempo real. Isso significa que este sistema controlador têm condições de lidar com as variáveis do sistema e executar as operações necessárias ao controle definido. Esta é uma característica dos sistemas de controle analógicos, que operam em tempo contínuo, diferente dos sistemas de controle digital. Porém, para a discussão de um sistema de controle em tempo real, baseado em um sistema de controle digital com processador, faz-se necessária uma definição formal do que significa efetivamente um sistema operando em tempo real. Antes disso, porém, deve-se ter em mente o que é uma aplicação computacional em tempo real e quais são as suas necessidades.

Uma maneira simplificada para se definir uma aplicação computacional de tempo real é considerar que, sempre que exista um processador e algum tipo de interface com o mundo externo que possua necessidades temporais de execução de rotinas de controle e/ou de aquisição de dados, estar-se-á se tratando de uma aplicação de tempo real [33].

Quando uma aplicação computacional em tempo real típica possui restrições críticas nos tempos de execução de suas tarefas, de maneira a exigir que um sistema gerencie os seus recursos de forma adequada, ao sistemas de controle deste tipo de aplicações dá-se o nome de sistemas operacionais de tempo real (RTOS) [33]. Este tipo de sistema, atualmente, se encontra em muitas áreas do cotidiano, estando presente em dispositivos mecatrônicos da indústria automobilística, como injeção eletrônica, o freio

anti-blocante (ABS), na indústria de consumo: nos aparelhos de vídeo-cassete, nos fornos de micro-ondas, vídeo-games e telefones celulares, dentre outros.

Por definição, aplicações de tempo real de uma maneira geral necessitam de um período de tempo máximo, bastante definido, para completar as operações que se fazem necessárias [33]. O controle de motores elétricos, pelas suas necessidades, por exigir que as rotinas de controle e manipulação de variáveis ocorram em tempo pré-definidos, é uma aplicação em tempo real. As conseqüências para uma falha neste limite de tempo podem variar desde intermitir, causando um mero incômodo, até casos extremos, de risco de vida ao usuário, como no caso de um sistema de controle e acionamento dos atuadores de um avião.

3.2 Características de um sistema em tempo real

Para garantir o atendimento de todas as necessidades das aplicações em tempo real, de uma maneira confiável, um sistema operacional em tempo real deve reunir certas características, ou atributos. Dentre estes atributos, é importante citar os seguintes:

- Determinismo: É a capacidade de um sistema executar uma ação que seja completamente determinado e reproduzível. Esta é uma tarefa que pode teoricamente ser idealizada totalmente [34].
- Tempo de Resposta: O tempo gasto entre o final de um pedido feito e o ínicio da resposta do sistema, ao pedido [35].
- Tolerância a Falhas: é o método de garantir a operação contínua do sistema, na presença de falhas [34], ou seja é a habilidade do sistema para continuar funcionando, quando da presença de falhas no equipamento (*hardware*) ou no programa (*software*) [36].

- Contexto: é a mínima informação necessária, com o propósito de salvar a tarefa em execução, de modo que ela possa ser completamente reativada [36].
- Chaveamento do Contexto: é o salvamento dos estados de um processo que está sendo executado e a restauração do estado de outro processo, permitindo ao processador chavear entre tarefas, sem perda das informações de execução [37].
- Latência: é o atraso entre um pedido e o tempo que o pedido é satisfeito [37].

Há quatro perfis padronizados para núcleos em tempo real [33], definidos pelo grupo de trabalho POSIX.13: o núcleo mínimo, o tipo controlador, o tipo dedicado e o multi-propósitos. A Fig. 18 apresenta o diagrama de relacionamento entre estes perfis que foram padronizados, onde cada um destes perfis, engloba seu antecessor acrescentando mais serviços, maior capacidade de processamento e características de maior porte:

- perfil mínimo inclui aplicações de sistemas dedicados. Segundo a definição [33], nestes sistemas não existe memória de massa (discos ou fitas magnéticas), e a aplicação se resume a um único processo com serviços de interrupções geradas pelo relógio de tempo real ou por dispositivos externos. Exemplos deste tipo de aplicação são os *firmwares* que controlam aparelhos de vídeo-cassete, vídeo-games e fornos de micro-ondas.
- O perfil controlador acrescenta dispositivos de E/S estruturados (padronizados), sistemas de arquivos e armazenamento de dados e código em discos virtuais em memória RAM.

- perfil dedicado adiciona características de multi-processamento (multi-tarefas e multi-processadores) e de gerenciamento de acesso a memória compartilhada,
- O perfil multi-propósitos consiste da totalidade das expansões POSIX.1 e POSIX.4, incluindo capacidades de interligação em rede, sistemas de gerenciamento de janelas e sistemas de armazenamento de massa de alta velocidade.

Sistemas com o perfil multi-propósitos, para sistemas de testes em tempo real, são oferecidos no mercado. O ambiente Matlab, somando ao seu pacote Simulink, oferece este tipo de recurso através da construção de interfaces utilizando a linguagem C e o próprio MatLab, como o sistema dSpace [9][38].

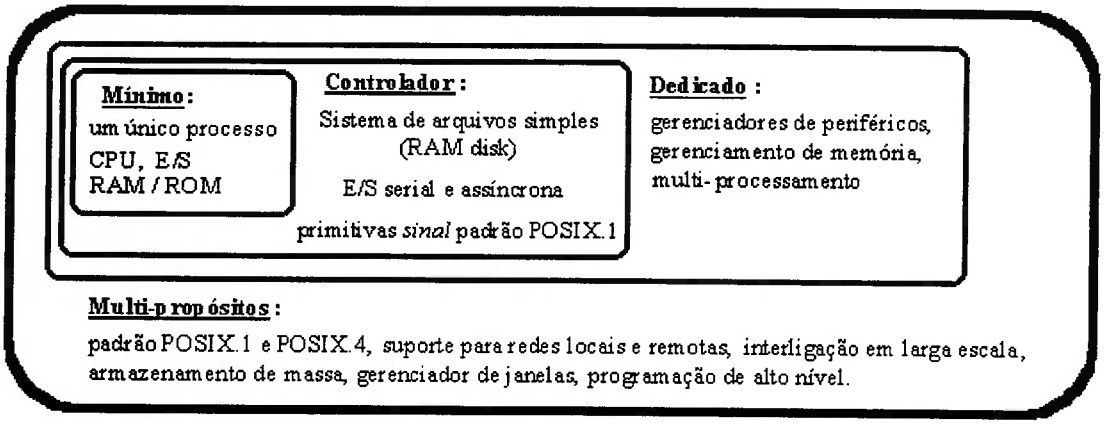


Fig. 18: Perfis dos núcleo em tempo real do padrão POSIX.

3.3 O núcleo

O núcleo de um sistema operacional, para um sistema baseado em DSP, considerando os requisitos mínimos, está apresentado na Fig. 19. Este núcleo é conhecido como *kernel* e é um sistema operacional utilizado em uma grande variedade de aplicações, tais como: controle digital, filtragem digital, monitoração de sinais, dentre outras. No tópico da E/S (entrada e saída) do perfil mínimo, são utilizados

dispositivos de aquisição de sinais, que nesses sistemas requerem amostragem uniforme, normalmente de múltiplos canais.

É muito usual utilizar um sistema operacional que divide o tempo das tarefas que necessita executar, em intervalos de tempo, ou executá-las, conforme a definição apresentada, em uma aplicação que se resume em um único processo. A utilização de intervalos de tempo permite a execução de processos não prioritários, que sejam completamente não intrusivos às tarefas em tempo real. Utilizando-se particulariedades da arquitetura do DSP e da aplicação na qual o processador é aplicado, o sistema operacional poderá estar altamente otimizado para obter o máximo do DSP, minimizando possíveis latências de interrupção e sobrecarga. Um outro tipo de núcleo, com múltiplas prioridades, mais flexíveis, requer maiores cuidados, porquê uma sequência de execução ativa num dado momento, pode ser difícil de determinar, especialmente se forem permitida às tarefas alterarem dinâmicamente suas prioridades.

Durante os últimos anos, muita atenção têm sido dedicada para reduzir o tempo de desenvolvimento de sistemas baseados em DSP. Conceitos de programação, oriundos da computação em geral, tais como: o BIOS, o sistema operacional e as linguagens de alto nível (C, C++) têm sido aplicadas, com vários níveis de sucesso, em projetos de sistemas DSP. Os sistemas onde tal aplicação é adequada, tipicamente gerenciam alguns dentre os seguintes tópicos:

- Interface direta com o equipamento;
- Processamento em tempo real de alta velocidade
- Processamento em blocos ou contínuo
- E/S periódicamente determinística

Para que o núcleo possua um bom desempenho e habilidade para lidar com os requisitos listados acima, deve-se utilizar programação de baixo nível, em linguagem de máquina (linguagem montadora).

3.3.1 Modos de Operação

No processamento em tempo real manuseiam-se tarefas de tempo crítico que permitem a amostragem uniforme de um dispositivo de E/S pelo núcleo. Neste caso, o processamento de tempo-crítico é executado atuando em procedimentos definidos pelo usuário, tais como: pré-amostragem, amostragem e saída e processo, além de cálculos complexos que forem necessários.

A pré-amostragem configura o dispositivo para fazer a amostragem; a amostragem e saída adquire a amostra, gera uma saída no dispositivo, ou retorno um ao sistema operacional. O processo faz os cálculos da amostra, normalmente consistindo de operações simples, tais como: soma, multiplicação, comparação, armazenamento e recuperação de variáveis, deslocamento de bits. Já os cálculos mais complexos, são normalmente: divisão, raiz quadrada, FFT, e comunicação entre processadores. Estes, podem envolver múltiplos ciclos de instrução, podendo ser manuseados pelo procedimento de retaguarda do sistema operacional, sendo definido pelo usuário. Uma sequência destes procedimentos, na forma de um diagrama, pode ser observada na Fig. 19, onde o sistema operacional executa o modo de configuração. Na Fig. 20, o modo de amostragem é executado e eventualmente, um modo de processamento em retaguarda pode ser executado.

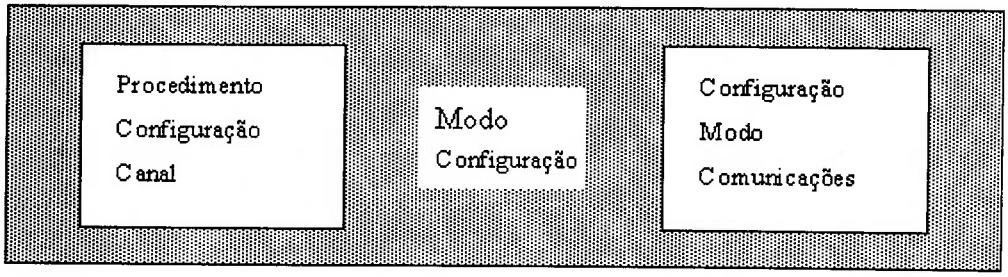


Fig. 19: Modo de operação

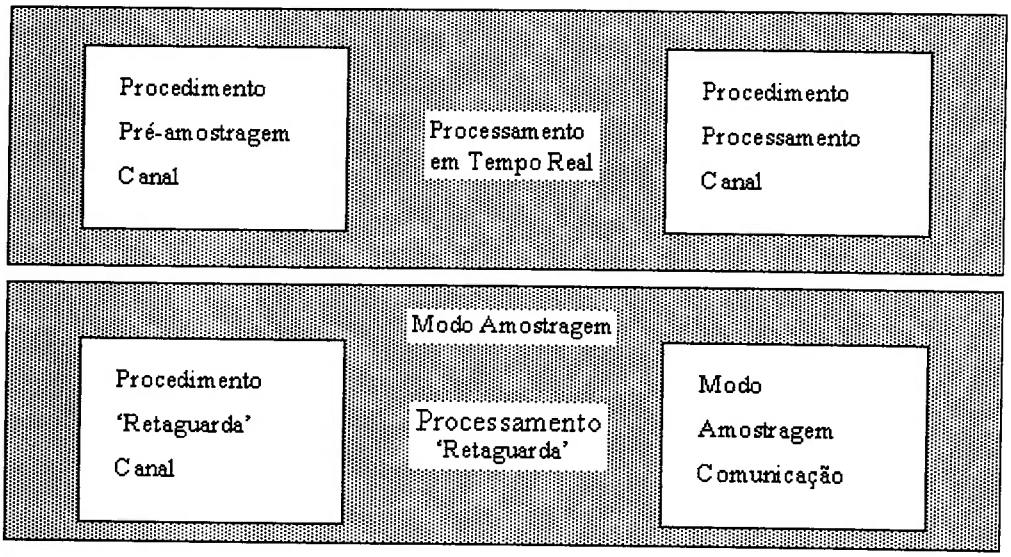


Fig. 20: Processamento em tempo real

Os procedimentos apresentados criam uma tarefa definindo o fluxo completo do processamento de uma amostra individual ou grupo de amostras relacionadas. O código nestes procedimentos é o que o projetista do sistema de controle necessita providenciar para completar uma aplicação.

Uma abordagem mais geral, do funcionamento deste núcleo, envolve a utilização de um mecanismo de sincronização. A tarefa pode ser suspensa enquanto ela espera por um sinal de uma interrupção, que aciona uma rotina, conhecida como rotina de serviço de interrupção (*ISR*). Até que este sinal apareça, a tarefa a ser executada não deve consumir ciclos do processador, desta maneira liberando o processamento do DSP para outras tarefas potenciais. Quando, além da tarefa principal, outras tarefas são

adicionadas ao sistema, usa-se um objeto convencional de sinalização que pode ser obtido via um semáforo. O semáforo é uma representação de que um recurso, num dado momento está ativo. Por exemplo, enquanto ocorre a leitura de um conversor analógico-digital, a rotina de serviço de interrupção pode possuir o controle do semáforo, desabilitando o serviço de interrupções naquele momento, e liberá-lo, habilitando-o quando tiver concluído a conversão; desta maneira, permitindo que uma outra tarefa de processamento obtenha a propriedade do semáforo.

Este esquema implica que se tenha um núcleo, do sistema operacional, que possa suspender a busca de uma tarefa em um recurso utilizado e possa gerenciar multíplas tarefas em um processador apenas. Em um núcleo desse tipo, permite-se expandir o sistema, somente com a adição de tarefas à lista do sistema. A forma como esta expansão é feita, dependerá da maneira que o tipo de chaveamento de tarefas é utilizado pelo núcleo.

Em adição às complexidades da programação, a maior preocupação em aplicações DSP em tempo real, é a sobrecarga – em termos do tempo de execução do processador, tamanho do código e espaço de dados utilizado – que pode ser dramaticamente maior do que em outros esquemas. Gerenciando múltiplas prioridades há a necessidade de um agendador (*scheduler*) de tarefas, que mantenha uma lista das multíplas tarefas para cada nível de prioridade, procurando pela tarefa de mais alta prioridade, para que seja executada sempre que houver a sincronização de eventos. Em sistemas de pequena escala (em ponto-fixo) que utilizam todo o ciclo disponível do processador este agendador pode consumir uma porção significante dos recursos disponíveis. Portanto, sua pouca aplicação em sistemas de DSP de pequena escala reside no fato de que em sua maioria, tais sistemas são sensíveis ao seu custo final, uma vez que a maioria das aplicações são em produtos de consumo.

3.3.2 Intervalo de tempo

O intervalo de tempo (*time-slice*) é uma temporização do processador que pode ser utilizada para gerar uma interrupção periódica, com um período de tempo específico. O uso do temporizador interno, ao processador DSP, para gerar uma interrupção específica para amostragem, simplifica a temporização em sistemas DSP. Uma interrupção de um intervalo de tempo pode ser gerada em uma taxa de:

$$s = \frac{1}{nT} \tag{14}$$

onde:

s = taxa do intervalo de tempo

n = número de canais

T = período da amostragem no dispositivo de aquisição

A interrupção de intervalo de tempo cria uma taxa de amostragem periódica e determinística para uma amostragem uniforme de múltiplos canais.

A grande maioria dos algoritmos aplicados nos sistemas de amostragem de sinais podem ser derivados da teoria de sistemas lineares. Uma admissão fundamental no processamento em tempo discreto baseado na teoria de sistemas lineares é de que o sinal é amostrado numa taxa de amostragem uniforme e periódica

$$u(k) = x(kT) \tag{15}$$

sendo:

u = sinal amostrado

k = número de amostras

x = sinal analógico

T = período de amostragem

Desta forma, o período de amostragem (T) deve permanecer constante e preciso. Erros na taxa de amostragem resultam em erros na precisão do sistema no qual o algoritmo é aplicado.

Basear a conversão da aquisição de dados no sinal de saída do temporizador pode reduzir o tempo de dependência da instrução. Entretanto, devido ao número limitado de saídas de temporizadores do sistema com múltiplos conversores e múltiplos canais, a utilização de E/S como memória mapeada no interfaceamento do equipamento pode melhorar as características do sistema. Neste caso, cada conversor ou canal pode ser individualmente selecionado através de endereços reservados de E/S. Nestes sistemas, uma seleção cuidadosa da temporização economiza um subconjunto de instruções, o que pode ser um ganho, para uso no processamento em "retaguarda".

A principal fonte de sobrecarga de processamento do núcleo, ou do sistema operacional comumente reside nas rotinas que armazenam e recuperam os registros, no código para atualizar o estado dos dados dos ponteiros de memória e executar as instruções de chamada dos procedimentos definidos pelo usuário. O ciclo de tempo necessário para executar as interrupções de intervalo de tempo, contribui para sobrecarregar o sistema. Uma maneira de possuir algum controle sobre o tempo crítico de sobrecarga de processamento é através da colocação de certos registradores dedicados para o processamento em retaguarda e ao processamento do intervalo de tempo. Isto é alcançado através da eliminação da necessidade de se ter de carregar e armazenar os registradores dedicados, em cada processo.

3.3.3 Avaliação de desempenho

Avaliações de desempenho do núcleo do processador DSP são importantes para definir parâmetros e tempos críticos do sistema operacional. Tais índices de desempenho podem ser expressos em ciclos de instruções (i/s) ou pelo tempo consumido (s) para uma dada tarefa ou processo. Os seguintes parâmetros são importantes:

1. Tempo chaveamento da tarefa ou intervalo de tempo;
2. Sobrecarga do processamento no intervalo;
3. Amostragem e saída, pré-amostragem e procedimento do processo para assumir execuções;

Além destes, também consideram-se:

4. Máxima taxa de amostragem (normalmente expressa em kHz), e
5. Tamanho do código.

Estes índices podem ser caracterizados para um ambiente que salve o contexto (registradores e variáveis) ou para um sistema operacional que possua recursos de dividir a utilização dos registradores. É de interesse do presente trabalho o de se caracterizar um núcleo de perfil mínimo.

Além destes fatores, um sistema operacional de propósito geral pode ter seu código escrito de maneira altamente otimizada, em linguagem montadora ou então permitir a utilização de funções, em linguagem de alto nível (como a linguagem C). Isto libera o projetista para focar-se na escrita do código específico de sua aplicação. Normalmente, utiliza-se sistemas com esquema do tipo *round robin*, onde há um chaveador de tarefas baseadas no tempo e cada tarefa recebe uma oportunidade igual de ser executada. Muitas vezes, determinadas tarefas necessitam de mais tempo do que

outras. Neste caso, o intervalo de tempo em alguns núcleos pode ser projetado de maneira que cada tarefa tenha seu próprio valor de intervalo de tempo e o código seja otimizado pelo projetista da aplicação de controle. Este pode ser um jeito simples de fornecer uma proporção específica do processador para cada tarefa, de acordo com a complexidade do sistema e da aplicação que esta sendo executada.

3.3.4 Encapsulando o processador em um sistema operacional

Uma aplicação como a do algoritmo de controle PI de um motor de corrente contínua deve fazer acesso às portas de E/S do processador ou de algum elemento periférico externo ao processador, como os dispositivos de aquisição de sinais. No instante que este acesso é processado, o aplicativo deve possuir meios de acessar o dispositivo. Como trata-se do acionamento de um pino ou de um periférico, do dispositivo físico, as linguagens de alto nível, como a linguagem C, permitem o recurso de acesso ao dispositivo utilizando uma intrução de escrita e outra de leitura, de acordo com a implementação das bibliotecas de função para aquele processador. No caso de processadores que utilizam a própria região de memória de dados do processador para mapear seus periféricos ou pinos de entrada e saída, a linguagem também deve oferecer funções que manipulem da memória para os respectivos dispositivos através da utilização de ponteiros.

Tratando do caso específico de processadores dedicados (microcontroladores e DSPs), onde é comum a existência de pinos de E/S e periféricos na própria pastilha, o funcionamento destes periféricos exige a configuração de registradores específicos em suas respectivas unidades, da mesma maneira que ocorre com os dispositivos periféricos programáveis, externos ao processador. Nesse caso, deve-se informar ao programa aplicativo (ou à própria linguagem de programação utilizada, no caso de tratarem-se de

linguagem de alto nível) a forma de acessar e programar tais registradores e os recursos dos periféricos que pretende-se utilizar, conforme a necessidade do modo configuração e o de pré-amostragem.

Uma maneira direta de programar o código do aplicativo e a manipulação de registradores e recursos dos dispositivos periféricos do processador, é utilizar a linguagem montadora (*assembly*) deste processador, através de seu compilador respectivo. Apesar desta forma ser a mais direta possível nos processadores, muitas vezes a familiaridade do projetista do sistema de controle (aplicação) com o código nativo (mneumônicos), com o compilador e os recursos internos do processador, não são de total conhecimento do projetista do aplicativo. Muitas vezes é secundária a requisição de execução do código, uma vez que foi escolhido utilizar um ambiente de desenvolvimento que apresente recursos de programação através de uma função mais intuitiva e de fácil manipulação e aprendizagem. Nestes casos, o acesso aos recursos de E/S deve estar "encapsulado" (ou escondido) em uma função de alto nível onde o projetista apenas necessite passar ou receber, nos casos de escrita ou leitura respectivamente, as variáveis que o seu aplicativo utiliza. Para realizar esta tarefa, normalmente um sistema operacional é implementado utilizando-se de camadas ou níveis conforme o diagrama da Fig. 21, que determinam como o recurso "encapsulado" – ou seja, a função na linguagem de alto nível – se relaciona com o nivel físico da programação do processador ou do código nativo.

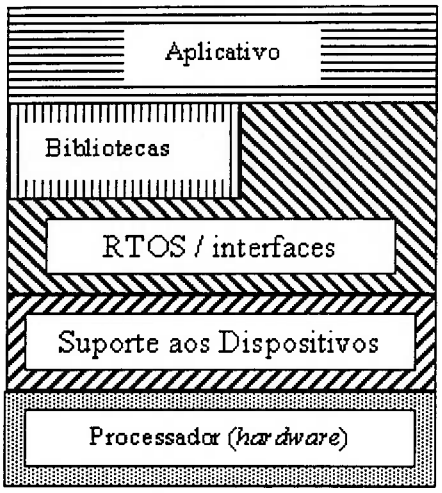


Fig. 21: Diagrama de referência do acesso aos dispositivos

O programa aplicativo descrito na Fig. 21 encontra-se localizado na quinta camada de baixo para cima (a mais externa das camadas) em relação à camada que permite o acesso direto ao dispositivo, a camada de baixo ou a mais interna. A primeira camada é o próprio processador onde o dispositivo está instalado. A segunda camada pode ser considerada como uma camada intermediária entre o sistema operacional (SO) e o processador, quando um SO permite ser utilizado entre diversos tipos recursos diferentes, sejam placas ou dispositivos, disponíveis no processador (que novamente podem ser internos ou externos à pastilha). Esta camada serve então de interface entre as rotinas suportadas no sistema operacional e o acesso físico aos dispositivos propriamente dito. A terceira camada, que acessa a camada que oferece suporte aos dispositivos é o próprio SO que contém interfaces de acesso entre o aplicativo e os dispositivos do processador e vice-e-versa. O próprio SO pode suportar o aplicativo diretamente, que encontra-se localizado na quinta camada do diagrama, ou pode ser acessado através de bibliotecas de funções que o aplicativo ou a linguagem de programação utilizada possui. Estas funções externas ao aplicativo estão localizadas na quarta camada no diagrama de referência.

Além do modo direto comentado, há muitos sistemas em que o SO está diretamente relacionado ao processador que se está utilizando e onde tal camada já contém as funções necessárias para a implementação de aplicativos, assim as camadas intermediárias 2 e 4 podem ser abolidas, ficando o aplicativo conectado aos dispositivos através do SO. A vantagem deste método é a economia de tempo que é possível em um acesso entre o aplicativo e o dispositivo. Em contrapartida, uma solução deste tipo inviabiliza a portabilidade do SO, uma vez que o mesmo deve ser modificado toda vez que os recursos do processador ou do equipamento utilizado são diferentes.

Núcleos operacionais, quando bem projetados possibilitam:

- Isolar o equipamento do programa (hardware do software, respectivamente), permitindo a migração para uma nova plataforma;
- Permitir que o código do dispositivo seja reutilizado;
- Permitir que o projetista do aplicativo acesse o equipamento facilmente, reduzindo ambiguidades que poderiam existir no acesso através de uma padronização possível com o "encapsulamento" criado.
- Permitir que sejam selecionadas configurações do tipo usualmente mais utilizadas, nos dispositivos periféricos do processador e/ou do equipamento, dessa maneira eliminando trabalhos tediosos que uma configuração manual por ventura exija do projetista.

Transformar, ou "encapsular", os dispositivos em objetos os quais são vistos pelo código do aplicativo permite a facilidade de operação do sistema, bem como a portabilidade do código fonte do aplicativo, para outras implementações. A inclusão de um recurso deste tipo, em um sistema de controle, permite flexibilizar o controlador digital empregado em uma planta. É desta maneira que os programas industriais e

comerciais são desenvolvidos. Assim a portabilidade, a flexibilidade e a facilidade de operação, são requisitos atualmente exigidos em um projeto de controle e automação, os quais originaram metodologias de implementação de programas de controle, tais como redes de Petri, Grafcet IEC[1]-848, programação UML e recentes padronizações como a *SoftLogic*, norma IEC 1131-3, originada a partir do projeto *Next Generation Controller*, NGC [39].

O núcleo do sistema foi implementado como sendo de perfil mínimo, e foi permitido definir o intervalo de tempo, através de funções de configuração dos temporizadores internos do processador DSP. Dessa maneira, o projetista pode otimizar o sistema para cada controle específico, de acordo com suas necessidades e o tempo requerido para a execução do processo. Isto flexibiliza o sistema de controle e serve como alternativa para a ausência de métodos de chaveamento do contexto, tornando-se útil quando atuando em conjunto com o controle de interrupções.

[1] International Electrotechnical Comission (IEC)

4 SISTEMA DE CONTROLE BASEADO EM DSP

4.1 Introdução

No passado fatores como a utilização de uma linguagem específica e de um desempenho não muito robusto perante aos processadores de uso geral, deixavam o DSP isolado em aplicações processadoras de sinais, onde a possibilidade de alteração dos parâmetros contava mais do que a necessidade de processamento rápido do sinal. Dispositivos lógico-programáveis muitas vezes eram utilizados na construção dos circuitos de filtragem digital e seu uso era restrito em áreas específicas, em projetos militares e de telecomunicações [1].

Segundo DOTE (1990), há uma exigência em mecatrônica, para diminuir o quanto for possível o tempo envolvendo o desenvolvimento de um projeto até a sua produção. Essas exigências e novos métodos, têm requerido esforços computacional altíssimos, os quais podem ser desempenhados por um DSP de alta velocidade combinado com um computador pessoal. Acredita-se que cada vez mais o DSP será utilizado, não apenas como um controlador digital ou processador de sinal, mas como um equipamento de projeto ou um analisador. Tais afirmações, feitas há 10 anos, hoje em dia estão tornando-se realidade, com DSPs assumindo funções em equipamentos de áudio e vídeo, utilizados como analisadores de espectro e em equipamentos de monitoração.

A possibilidade de criar um sistema baseado em DSP, voltado para as condições de controladores de motores foi o caminho escolhido como tema deste trabalho.

4.2 Escolha do DSP do sistema

Tradicionalmente os sistemas de controle utilizam circuitos e filtros analógicos com projetos específicos e de díficil implementação devido à necessidade de ajustes localizados nos circuitos de controle. Além desse fato, os circuitos analógicos sofrem variação de parâmetros ao longo do tempo e devido a fatores externos como a temperatura e humidade. Como atualmente os circuitos de controle são executados por processamento digital, a necessidade de utilizar DSP tornou-se uma constante nos novos projetos de circuitos eletrônicos e de controladores [45][46][47]. O projetista de tal controlador precisa identificar nos parâmetros de seus projetos, quais os valores e os limites que determinam o desempenho e a possibilidade de implementação do controlador pelo circuito escolhido. Neste projeto, foi enfatizado a construção de uma placa digital de controle, utilizando um processador digital de sinais, para servir de alicerce ao sistema baseado em DSP. O projeto deste circuito de controle visa permitir a utilização de um processador com uma arquitetura recente, aliado às características de baixo custo e ótimo desempenho.

Antigamente um DSP era visto como um coprocessador. Isso significa que desde o surgimento do processador digital de sinais, quando então foi produzido em um único dispositivo (pastilha) semicondutor em 1980, sua arquitetura interna possuia instruções complexas. Tal conceito de arquitetura chamada por convencional começou a ser alterado a partir de 1996, através da introdução de uma arquitetura chamada de convencional-melhorada. Nesta nova arquitetura, suas intruções ainda continuavam complexas, porém recursos para executar uma única instrução com múltiplos dados (SIMD, Single Instruction Multiple Data) foram incorporados. Isso significa que uma instrução executa a mesma operação em múltiplos (independentes) conjuntos de dados.

em termos de desempenho, para cada instrução SIMD, pode-se alcançar duas, quatro, ou até oito vezes o trabalho de uma instrução. Duas maneiras de realizar SIMD: (1) dividir o DSP com unidades de execução de instruções e (2) múltiplas unidades de execução (ou barramento de dados) operando a instrução. Tais recursos, associados a outras inovações, também surgidos na segunda metade da década de 90, como a utilização de arquiteturas do tipo VLIW (Very Long Instruction Word) e arquitetura superescalar, sendo que esta última permite o recurso de planejamento da operação das instruções durante a sua própria execução. As gerações dos processadores DSP estão associadas à inclusão destes recursos.

De forma a incluir tais novos recursos em DSPs, a necessidade da utilização de ambiente de programação baseado em linguagem de alto nível é muito importante. Hoje, segundo os fabricantes e projetistas estes processadores devem em sua maioria serem programados em linguagens como a linguagem C e C++ sendo este o argumento para a escolha e utilização desta para o processador DSP do presente projeto. Alguns fabricantes chegam a ocultar e não fornecer a tradicional tabela de *OpCodes* de determinado DSP, sendo o único modo possível conhecer suas instruções através da desmontagem do programa que foi executado. Isso significa que um mesmo recurso escrito numa linguagem de alto nível, como a linguagem C, pode assumir instruções e operações diferentes dependendo do conjunto destes recursos utilizados na linguagem por dois programas diferentes de acordo com o nível e a escolha da otimização executada durante o processo de compilação do código fonte para o código binário.

Atualmente, é muito comum os processadores de uso geral (GPP, General Purpose Processors) de alto desempenho, como um processador do tipo Intel Pentium MMX executarem as tarefas DSP mais rapidamente do que os próprios processadores DSP. Dentre vários motivos para os processadores DSP continuarem sendo escolhidos

pelos projetistas é o de possuir vantagens como a integração de funções em um dispositivo orientado para DSP, o preço, o consumo de energia, a disponibilidade de programas comerciais e ferramentas de desenvolvimento orientadas para DSP, como por exemplo a integração de ferramentas de simulação matemática do tipo Mathworks MatLab e o ambiente integrado de desenvolvimento Metrowerks Codewarrior, na geração de código em linguagem C a partir de modelos simbólicos além de permitir que o tempo de execução seja previsível, que é algo especialmente problemático com GPPs de alto desempenho.

Uma categoria de dispositivo de baixo custo que vêm popularizando-se no mercado é o de arquitetura híbrida. Este é um processador extremamente atraente em aplicações dedicadas e embarcadas, ou *embedded applications* como são conhecidas. Muitos fabricantes que têm direcionado GPPs neste mercado, agora direcionam seus DSPs. Assim como seus irmãos mais potentes, os *embedded* GPPs, possuem recursos avançados que afetam a previsibilidade do tempo da execução de instruções, facilitando a escolha de utilizar um DSP. A arquitetura híbrida permite uma variedade de facetas do que podemos considerar como um processador híbrido. Dentre as várias concepções destas arquiteturas, podem-se citar:

1. a utilização de múltiplos processadores numa única pastilha, compartilhando barramentos, para comunicação em comum;
2. processadores combinados à coprocessadores DSP no mesmo dispositivo;
3. microcontroladores do tipo RISC de 32 bits, de alto desempenho e popularidade no mercado, que são enxertados com instruções e uma unidade DSP de 16 bits em seu núcleo interno.
4. inserir recursos de microcontroladores aos já existentes núcleos de DSP.

Esta última, é a arquitetura utilizada no processador DSP56824, da familia Motorola DSP56800, escolhido para o sistema baseado em DSP.

Dentro do objetivo do controle de um motor de corrente contínua, foi considerado como parâmetro de projeto uma referência ao tempo necessário para realização da malha de controle. O valor de referência considerado e adotado em várias aplicações na literatura, para a dinâmica de um motor elétrico é de 1ms. Para obter uma malha de controle dentro desta especificação, a literatura têm mostrado que muitas vezes os processadores de 32 bits têm sido utilizado para executar as estratégias de controle de motores eléticos mais sofisticadas, em tempo real [14][16][17][18][27][28][40]. Estes processadores, operam com frequências da ordem de 33 a 50 MHz. Em [11] para a implementação da estratégia de controle apresentado, dois DSP foram necessários na mesma placa, devido a um tempo de referência menor ainda por trabalhar com altas frequências. Em muitos casos [18][40][42][43], utilizam-se dois processadores, sendo um deles o próprio DSP e um outro processador, muitas vezes genérico, de apoio ao DSP. O microprocessador ou microcontrolador de apoio fica encarregado de tarefas periféricas, deixando o DSP livre para executar as instruções e realizar os cálculos necessários, no limite de seu processamento.

Já no sistema baseado em DSP deste trabalho, com a escolha de utilizar apenas um processador na placa de controle, tal dispositivo deve-se ocupar do processamento do sinal em si e também das tarefas periféricas, tais como: a comunicação com outras placas, a aquisição do sinal e também atividades de controle de portas de entradas e saídas (I/O). A escolha de um DSP híbrido que reune as características de um processador de sinais, através da existência em seu núcleo de uma arquitetura de DSP completa e a flexibilidade de instruções e módulos periféricos internos, no estilo do que

é encontrado apenas em microcontroladores, permitiu utilizar apenas um processador na implementação do circuito do sistema.

Em um processador híbrido, como o modelo escolhido, há dois modos independentes de operação de seu núcleo interno. No primeiro modo, o processador atua como um DSP apenas. A Fig. 22 apresenta este caso, em que o módulo do controlador do programa decodifica uma instrução DSP do tipo MAC (número 1); esta instrução é despachada para a Unidade Lógica Aritimética de Dados (ALU, número 2); a ALU toma os dados na RAM (número 3) e o resultado é escrito na memória (número 4). Já na Fig. 23, o processador está atuando como um microcontrolador. Neste modo, o controlador do programa decodifica uma instrução de microcontrolador (número 1), esta instrução é despachada para unidade de manipulação de barramento e bit (número 2) e a unidade de geração de endereços (AGU) seleciona o módulo GPIO do dispositivo (número 3) e a unidade de manipulação de barramento e bit envia o bit manipulado para GPIO (número 4).

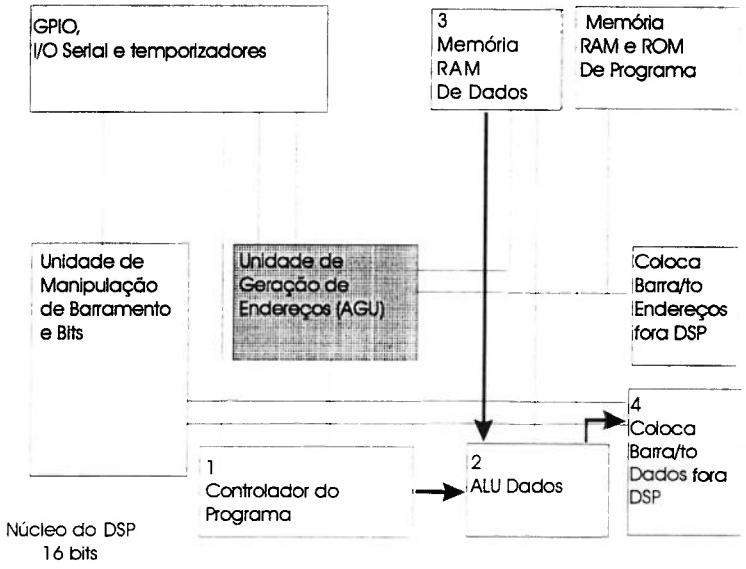


Fig. 22: Modo de operação como DSP

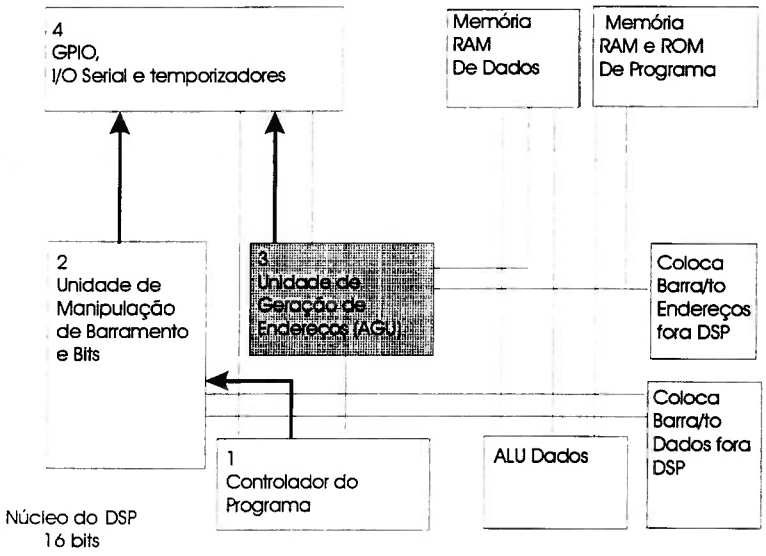


Fig. 23: Modo de operação do tipo microcontrolador

Uma visão completa da arquitetura do processador DSP híbrido DSP56824 pode ser observada na Fig. 24. Nesta figura destacam-se os sete barramentos, além do barramento de geração de *clock* (relógio), internos do processador. Destes, três barramentos são bidirecionais, sendo XAB1, XAB2 de 16 bits para acesso aos endereços da memória de dados e PAB de 19 bits, para transferência do endereço do segmento do programa. Um barramento exclusivo de transferência de dados (PDB). Outro barramento, geral, do núcleo do processador (CGDB), além do barramento geral dos periféricos (PGDB). Um segundo barramento, do segmento de dados, para acesso externos, o XDB2, unidirecional e de 16 bits. Isso permite observar que, para executar instruções SIMD é necessário escolher corretamente os registradores e o acesso à memória (interna e externa), pois do contrário pode-se piorar o desempenho do processador, devido ao intertravamento de barramentos pela necessidade de um dado ter de esperar que uma unidade acesso outro endereço, até que possa executar ou completar uma operação em andamento. Este é um dos principios e principal motivo, da necessidade e do uso de compiladores em linguagem C para extrair o melhor

desempenho deste tipo de processador por parte do programador do aplicativo, preterindo a linguagem assembly, tradicional em sistemas em tempo real de processadores dedicados.

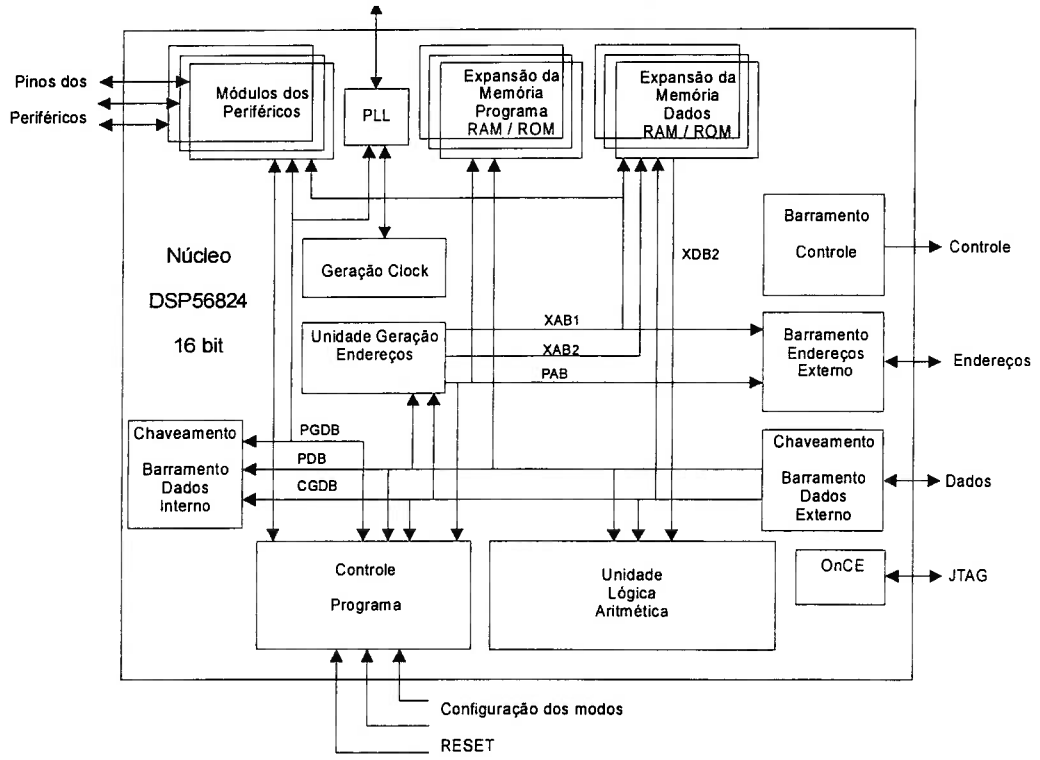


Fig. 24: Arquitetura interna do DSP56824

Com um poder de processamento de 35 MIPS (milhões de instruções por segundo), quando operando na frequência de 70 MHz, o processador Motorola DSP56824 escolhido possui desempenho igual ou superior aos processadores em uso para controle de motores e supressão de distúrbios [5][6][48][49][50], além das vantagens de tratar-se de um DSP de arquitetura híbrida, conforme já discutido, em um núcleo DSP de 16 bits de ponto fixo, contra os processadores mais sofisticados (tanto DSPs, como GPPs e microcontroladores) de 32 bits que usualmente têm sido utilizados em controle digital de motores elétricos em tempo real. E por último seu fator preço que permite colocá-lo como um DSP que concorre na faixa dos microcontroladores de 16

bits, importantes para aplicação de estratégias de controle em dispositivos industriais e de consumo.

4.3 Implementação do Circuito

Uma vez definido o processador que integra o núcleo do sistema, a implementação de seu circuito pode ser feita, através do projeto dos circuitos eletrônicos que estão apresentados no Apêndice 2. Na Fig. 25, apresenta-se um diagrama de blocos para o sistema baseado em DSP implementado. São os seguintes recursos que o sistema possui:

- Processador Digital de Sinais (DSP)
- Memória ROM
- Memória RAM
- Memória Dual Port (DP)
- Barramento ISA
- Conversor Analógico-Digital (ADC)
- Conversor Digital-Analógico (DAC)

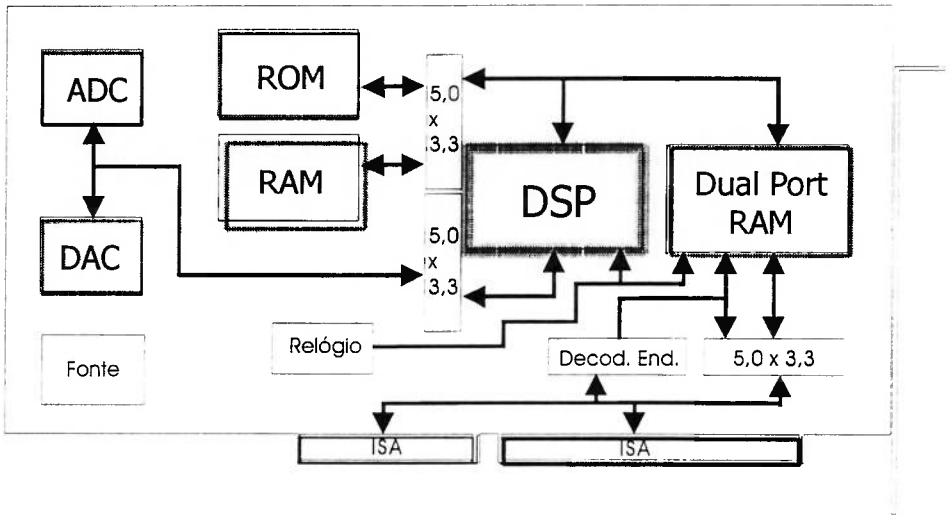


Fig. 25: Diagrama de blocos do sistema baseado em DSP

Para permitir o acesso do DSP aos outros módulos do sistema, uma vez que seu núcleo é formado de apenas um processador, foram utilizados as portas que estão disponíveis neste dispositivo e que permitem a comunicação de seus barramentos internos e seus periféricos, com o meio externo. Estas portas, que estão disponíveis ao usuário do sistema através de intruções do processador, são as portas A, B e C, conforme indicado na Fig. 26.

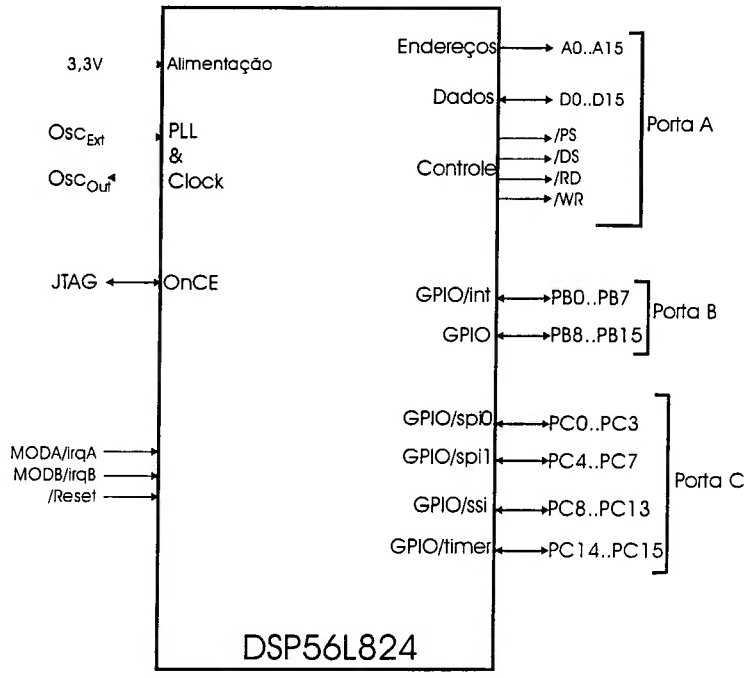


Fig. 26: Diagrama de blocos do DSP

O DSP utilizado é um processador com arquitetura do tipo Harvard, possuindo dois segmentos de memória (interna/externa), sendo um segmento para a memória de programa que contém 64 Kwords (embora seu núcleo permita endereçamento até 2^{19} = 512 Kword) e outro segmento para a memória de dados do sistema, com capacidade de até 64 Kwords. A conexão dos dispositivos de memória ao DSP é feita através da porta A do DSP. Esta porta permite o acesso aos dados e programas que estão nas memórias externas ao dispositivo. A escolha entre o segmento de programa e o segmento de dados, nos dispositivos externos de memória, é feita através dos sinais de controle PS e DS, respectivamente, enviados pelo processador de acordo com o tipo e sequência da instrução executada. O acesso externo aos 64k, de memória, é feito através dos pinos externos do barramento de endereços [A0..A15]. A Fig. 27 apresenta como os dois segmentos da memória do sistema foram implementados; nesta figura é possível verificar endereços no segmento de memória de dados que são exclusivos aos periféricos internos do processador DSP, sendo alguns deles utilizados no circuito, para acesso aos dispositivos conversores analógico-digital e digital-analógico. Nestes endereços, o processador desvia o acesso à memória externa, operando os bits dos registradores internos das portas B e C, manipulando respectivamente os pinos destas duas portas, através da unidade de manipulação de barramento e bits. As portas B e C do DSP são configuráveis, compostas de 16 pinos cada, permitindo que eles sejam utilizados como pinos de entrada e de saída, para propósito geral (GPIO) ou como periféricos específicos; ou uma combinação destes dois modos de funcionamento.

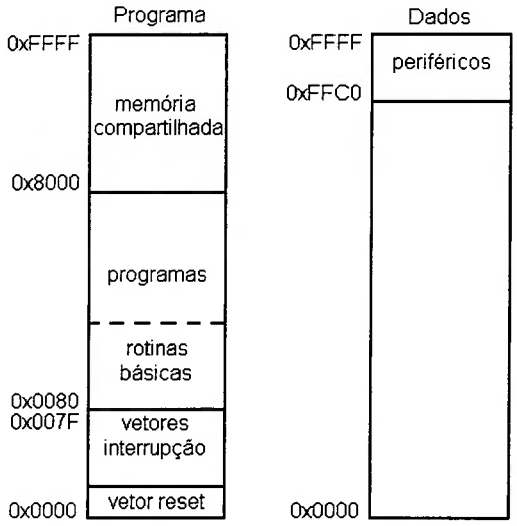


Fig. 27: Distribuição da memória no sistema

A memória de programa do sistema está fisicamente dividida em duas partes, sendo na parte mais baixa - onde localizam-se os vetores de inicialização e de interrupções do processador –utilizados dispositivos de memória não volátil, que armazenam tanto rotinas básicas do sistema, como podem armazenar, dependendo da quantidade de espaço ainda disponível, programas do usuário ou expansão do sistema monitor. A outra parte, mais alta, do segmento de programa é ocupada pela memória de acesso duplo, que está conectada em um barramento de um microcomputador hospedeiro. Ambas as partes de memória utilizadas, ocupam 32k cada uma.

A escolha de colocar na parte alta do segmento de programa – que compreende a área que endereça a memória de acesso duplo – permite usufruir de um recurso não invasivo ao sistema. Desta maneira é permitido ao DSP processar código sem interrupção para atender chamadas externas, diferentemente do que ocorre nos sistemas em que foi preferido utilizar o processador DSP junto de um segundo processador, dedicado para as tarefas periféricas. Nestes casos, o processador precisa atender aos

pedidos de transferência de dados entre os dispositivos periféricos, algumas vezes diminuindo o desempenho intrínseco do sistema.

A opção de utilizar a memória de acesso duplo no sistema, que é um recurso que permite ao processador DSP acessar código neste dispositivo, realizando tanto leitura como escrita de dados, e da mesma forma, permite que um microcomputador hospedeiro, do tipo IBM-PC compatível, leia e escreva nos endereços da memória de acesso duplo através de um barramento do tipo ISA de 16 bits. A utilização de memórias de acesso duplo, permite multi-processamento e diminuição na latência do sistema [14][15]. No sistema a memória de acesso duplo utilizada é um dispositivo síncrono, que permite operar com frequência máxima de 83MHz, estando acima, portanto, da frequência de operação do dispositivo DSP (frequência máxima de 70 MHz). Isso significa que não é necessário a implementação de ciclos de espera no dispositivo DSP ao se executar código através deste segmento, permitindo usufruir do máximo desempenho possível.

Como a memória de acesso duplo possui dois barramentos de acesso à sua matriz interna de posições de memória, pode-se ler e escrever nas mesmas posições da matriz através de uma lógica de controle a partir de dois barramentos mutuamente exclusivos. Outra função que estas memórias incorporam é o recurso de passar os dados provenientes do barramento de um lado para o outro lado. Por tratar-se de um dispositivo de lógica síncrona e permitir que dois barramentos alterem o conteúdo dos dados internos, regras de controle ao acesso são aplicadas ao modo como o acesso aos dados é executado. Dessa maneira, apesar de ambos os barramentos poderem ler simultâneamente o conteúdo interno de uma dada posição na memória, no momento de realizar uma escrita, uma porta tem prioridade sobre a outra. No projeto e implementação do circuito do sistema, a porta que tem a prioridade da escrita dos dados

na memória de acesso duplo é a porta que está conectada ao barramento do DSP, sendo a outra conectada ao barramento ISA do microcomputador hospedeiro.

A conexão ao barramento ISA, do computador hospedeiro, a partir do segundo barramento da memória, permite que, tanto o funcionamento do sistema, bem como a execução de seu código seja monitorado. Da mesma forma, pode-se alterar variáveis e também implementar estratégias de acesso posterior aos dados manipulados pelo DSP. Um programa que descarrega o código binário, no padrão Motorola SREC utilizado pelo processador, foi incluído no sistema e sua listagem é apresentada no Apêndice 3. Este programa atua do lado do PC hospedeiro, com a opção de utilizar o barramento ISA, sendo a memória do sistema baseado em DSP, vista pelo microcomputador, dentro da área que é reservada à extensão de ROM no projeto do PC. Esta área está compreendida entre os endereços 0xC0000h até 0xE0000h, num total de 128 kB de memória acessível total do segmento do PC permitindo endereçar a memória de acesso duplo. Para flexibilizar a utilização do protótipo implementado, na maioria dos PCs com barramento ISA, dentre os possíveis endereços disponíveis dentro da faixa indicada acima, foi disponibilizado um conjunto de chaves que de acordo com a sua configuração, alteram o endereço da memória de acesso duplo do sistema, que é visto pelo PC. A manipulação destas chaves não altera o endereço interno da memória de acesso duplo, que é visto pelo DSP e os programas em execução, no sistema baseado em DSP.

Além dos recursos descritos o projeto inclui dois dispositivos conversores: um conversor analógico-digital e um conversor digital-analógico. O dispositivo escolhido para o conversor analógico-digital, foi o AD7828, da Analog Devices, com resolução de 8 bits e taxa de conversão de 2,5 μs por canal. Este dispositivo possui 8 canais de entradas analógicas, multiplexadas de acordo com a seleção de endereço do canal

escolhido, conforme indicado no diagrama da Fig. 28. O dispositivo conversor digital-analógico utilizado no sistema é o TLC7225, da Texas Instruments, que possui internamente 4 conversores, independentes, com 8 bits de resolução e recurso de memória do valor disponibilizado em sua saída, conforme ilustrado na Fig. 29. O protótipo final, do sistema baseado em DSP implementado pode ser observado na foto apresentada na Fig. 30.

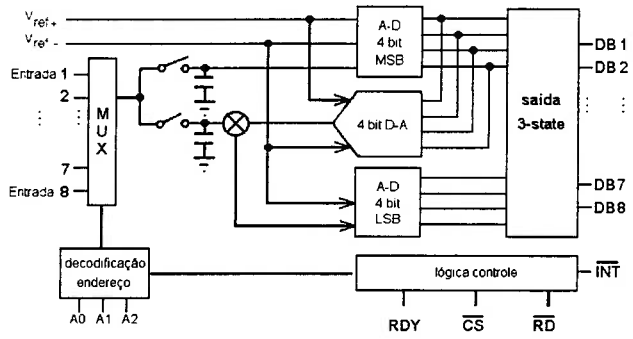


Fig. 28: Diagrama interno do dispositivo conversor analógico-digital

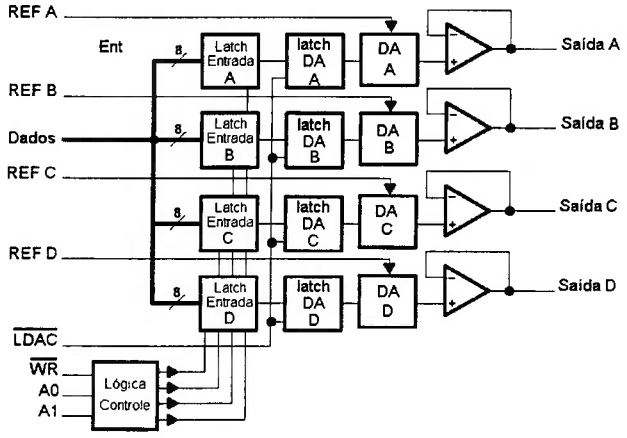


Fig. 29: Diagrama interno do dispositivo conversor digital-analógico

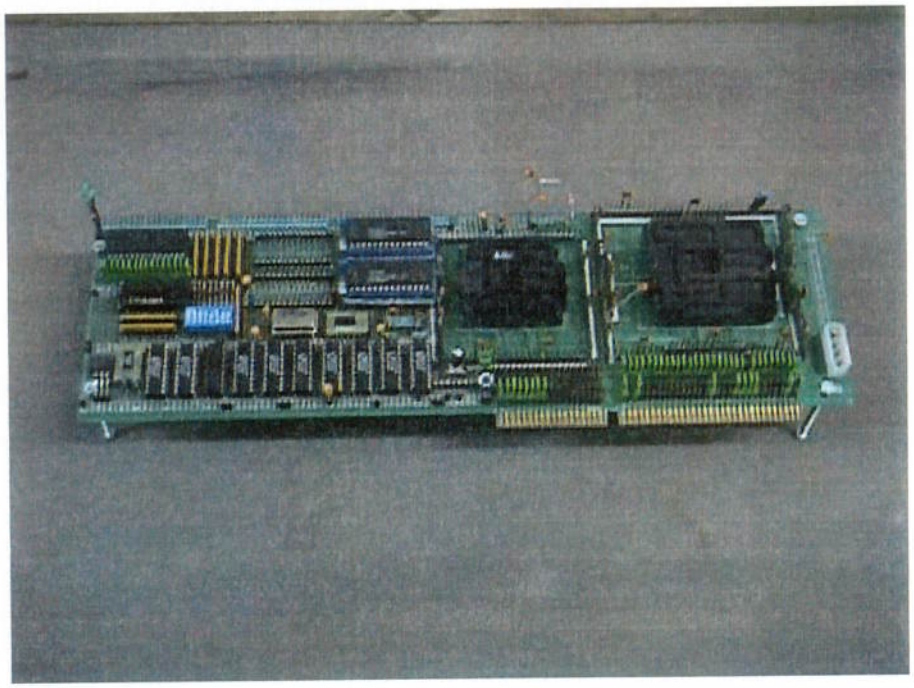


Fig. 30: Protótipo do sistema baseado em DSP

4.4 Sistema Monitor

Durante os anos 80 e o início dos anos 90, o grande desenvolvimento tecnológico na área de computação promoveu a popularização cada vez maior de aplicações de tempo real. Isto desencadeou uma produção enorme de sistemas operacionais de tempo real proprietários de pequeno porte cada vez mais sofisticados, mas que não respeitavam nenhum padrão pré-definido de comunicação de dados ou de conexão em rede ou mesmo de projeto do *firmware* de controle. Esta característica aumentava a flexibilidade das aplicações desenvolvidas, pois o projetista podia escolher, dentre os sistemas existentes àquele que possuísse as características mais importantes e adequadas à sua aplicação, mas diminuindo enormemente a portabilidade e limitando a aplicação à plataforma escolhida durante o projeto, e portanto dificultando uma posterior migração para outras plataformas de *hardware* e *software*. [33].

Foi enfatizado durante este projeto a obtenção de rotinas básicas do sistema baseado em DSP, permitindo ao usuário durante o desenvolvimento do seu projeto o teste e funcionamento de um sistema de controle digital a disponibilidade dos recursos do processador, de maneira agradável e simples, sem a necessidade de conhecer a arquitetura interna, familiarizando-se apenas com o sistema em si. Desta maneira, o objetivo é diminuir o tempo de um projeto de sistema de controle baseado em DSP e permitir ao projetista, focar-se apenas em seu projeto do sistema de controle.

4.5 Rotinas Básicas

Rotinas básicas são códigos implementados com o objetivo de permitir ao usuário o acesso aos recursos periféricos do DSP, recursos que podem ser ou de periféricos internos como o de periféricos externos ao DSP, conforme discutido no capítulo 3. A ênfase na criação destas rotinas foi a de disponibilizar funções (em linguagem *assembly*) que podem ser chamadas de códigos aplicativos escritos na linguagem C. Desta maneira, além de oferecer uma maior facilidade de programação, têm se o objetivo de deixar o código da aplicação do usuário portável para outros ambientes e altamente otimizado ao processador DSP em uso. Serão considerados nos periféricos internos do dispositivo DSP, recursos como as portas de I/O, os temporizadores/contadores e outros recursos de E/S serial, específicos. No caso dos periféricos externos ao DSP, estes são os recursos disponíveis no circuito do sistema baseado em DSP, como os conversores analógico-digital e digital-analógico. As rotinas implementadas no circuito, com chamadas a partir do aplicativo desenvolvido em linguagem C, incluem:

- Inicialização de configurações de barramentos e recursos específicos do DSP;
- Inicialização das portas B e C, do DSP;

- Leitura nas portas B e C;
- Escrita nas portas B e C;
- Acesso de leitura dos canais do conversor A/D;
- Acesso de escrita nos canais do conversor D/A;
- Acesso e programação dos temporizadores do DSP;
- Inicialização de Interrupções;
- Inicialização e atualização do sistema tolerante a falhas.

Para apresentar o trabalho que foi desenvolvido sobre o tema discutido no capítulo 3, será oferecido uma rápida análise das funções das rotinas básicas que foram implementadas no sistema monitor do circuito considerando as soluções encontradas durante sua implementação, sobre as necessidades e os parâmetros que *Sistemas de Controle em Tempo Real* devem possuir. O funcionamento destas rotinas, no sistema baseado em DSP, da forma como é acessível ao projetista do controlador digital, bem como sua forma de utilização é fornecido a seguir. Uma listagem mais detalhada, dos recursos discutidos e da forma de implementação destas rotinas básicas [8], está incluída no Apêndice 1.

4.5.1 Inicialização das portas B e C, do DSP

O dispositivo DSP permite configurar a porta B, de 16 bits, como 16 pinos de E/S para propósito geral (GPIO). Destes 16 pinos, 8 podem ser configurados como pinos de entrada de interrupção. Todos os pinos da porta B estão disponíveis no circuito do sistema, através de conector. A porta C, assim como a porta B, possui 16 bits. Seus 16 pinos podem ser configurados como pinos de GPIO ou pinos específicos de funções periféricas de E/S serial e temporizadores. Um outro conector também coloca a porta C

disponível ao usuário. Dessa maneira, o projetista tem disponível, o acesso ao ambiente, através da programação do DSP. Em ambas as portas, quando forem configuradas como GPIO, pode-se escolher entre o pino de E/S bidirecional, estar disponível como entrada ou como saída de dados, num dado instante, independentemente. Cada um destes recursos é programado em registradores específicos da porta B. Para permitir ao projetista configurar estas portas, inicializando-as, duas rotinas estão disponíveis: *inicia_b()* e *inicia_c()*. Nestas funções, nenhum parâmetro é passado ou retornado pela função, apenas os registradores das portas serão configurados. Estas funções consideram que a porta B e a porta C estão sendo inicializada pela primeira vez, antes da escolha de alguma opção específica que as portas possam oferecer.

No caso da inicialização da porta B, a função desabilita a opção de interrupções, configura todos os pinos como GPIO e a direção dos pinos como saída. Além disso, ela configura o registrador de controle do barramento, colocando na opção de estados de espera (*wait-states*) – para as operações de leitura e escrita no barramento de memória externa – o valor zero. Para a inicialização da porta C, a função *inicia_c()* desabilita os periféricos internos do dispositivo e coloca pinos da porta C como pinos de GPIO, sendo a direção de saída configurada em seus registradores.

4.5.2 Escrita nas portas B e C

Muitas vezes é necessário acionar dispositivos externos ao circuito, de maneira digital, ligando ou desligando seus acionamentos. A função *escreve_pb(<par1>)* e *escreve_pc(<par1>)* permite que seja escrito um valor de 16 bits, onde *<par1>* é um parâmetro passado para os pinos da porta B e ou, da porta C. Esta função normalmente é utilizada após a inicialização da porta B e/ou da porta C. Como trata-se de uma função de escrita na porta, a direção dos dados que foi configurada no registrador, não é

necessário alterá-la. O parâmetro passado é direcionado pela unidade de manipulação do barramento e de bit, para os pinos PB0..PB15 da porta, conforme foi mostrado na Fig. 23.

4.5.3 Leitura nas portas B e C

É comum, em projetos de sistemas de controle, o projetista necessitar manipular entradas digitais, que permitam tomar uma decisão baseado em uma ação externa. Um exemplo clássico é inserir um botão de parada, no caso de pane, no sistema de controle que está sendo executado, para não danificar o equipamento ou a planta do processo. Esta parada pode ser habilitada através de um *pooling* no programa, que faz a leitura de um pino externo ou através de uma interrupção específica. A função de leitura do conteudo das portas B e C permitem que seja retornado o conteúdo do registrador de dados dessas portas, contendo o estado instantâneo que os pinos se encontram. A função de leitura, implementada nas rotinas básicas do sistema baseado em DSP, possui a estrutura *<parret1> = le_pb()* ou *<parret1> = le_pc()*, para leitura da porta B e da porta C, respectivamente. Nesta função, *<parret1>*, é o parâmetro que a função retorna ao programa aplicativo, contendo o valor de 16 bits encontrado no registrador da porta acessada.

Para exemplificar as funções de inicialização, escrita e leitura das portas disponíveis através dos conectores do circuito do sistema baseado em DSP, é apresentado um pequeno código exemplo da aplicação destas três funções da rotina básica do sistema, demonstrado a seguir:

```
short dado;
void main(void)
{
      inicia_b();
      inicia_c();
      dado = le_pb();
```

```
        escreve_pc(dado);
}
```

Neste exemplo, ambas as portas são inicializadas, uma variável *dado* é alocada na memória do sistema baseado em DSP, que é utilizada para armazenar o parâmetro retornado pela função de leitura, com o conteúdo dos pinos da porta B, e passá-los aos pinos da porta C, através da função de escrita na porta. É importante lembrar que – como pode ser observado – em todo o código, desenvolvido em linguagem C, o projetista do sistema não necessita conhecer os detalhes de programação do dispositivo DSP ou do circuito da placa, conforme já discutido durante o desenvolver deste trabalho, bastanto ser instruído sobre o tipo da variável que é passada e que é retornada pelo sistema, além de familiarizar-se apenas com os pinos de E/S do circuito do sistema, e seus respectivos valores.

4.5.4 Acesso de leitura dos canais do conversor A/D

Infelizmente, os processadores digitais binários – como seu próprio nome evidencia – não permitem manipular diretamente as variáveis contínuas do 'mundo real', necessitando para isso, conversores que fazem a transformação da variável de um meio para outro. Conforme discutido na parte do circuito projetado para o sistema baseado em DSP, foi disponibilizado o acesso às variáveis continuas, aos sinais analógicos do meio externo, através do conversor analógico-digital. Contando com oito canais de entrada analógica, é disponibilizado ao projetista do sistema de controle, através das rotinas básicas, uma função que permite a leitura de cada um destes canais. A função *leitura()*, possui o seguinte formato: *<parret1> = leitura(int <par1>)*. Nesta função, *<par1>*, é o parâmetro que é passado pelo programa aplicativo, para a função de leitura, indicando qual dos oito canais do conversor deverá ser utilizado pelo sistema,

variando de 1 a 8. Feita a leitura do dispositivo externo ao DSP, *<parret1>* é o parâmetro retornado ao aplicativo, com o conteúdo já convertido do canal lido. Neste caso, o dispositivo utilizado possui 8 bits de resolução, mas retorna seu valor numa variável de 16 bits. Os bits mais significativos não são considerados.

4.5.5 Acesso de escrita nos canais do conversor D/A

Do mesmo modo que o controlador digital de um sistema de controle necessita, efetuar a leitura de sinais analógicos, na planta do processo, usualmente é necessário aplicar um sinal analógico no sistema. Este sinal pode ser um valor de referência, um distúrbio ou ruído necessário para teste de operação ou, como é mais comum, é necessário aplicar o valor de comando na planta. Este valor pode ser a quantidade de abertura de um atuador de uma válvula num tanque ou, no caso do controle de motores, a tensão ou corrente aplicada no sistema de acionamento do motor. Para permitir que o projetista tenha à disposição saídas de controle analógicas, foram disponibilizados no circuito do sistema baseado em DSP, quatro canais de saída em um dispositivo conversor digital-analógico. O acesso à este dispositivo, é efetuado de maneira simples, através da função incluída nas rotinas básicas, *escrita()*. Esta função necessita de dois parâmetros, que o aplicativo deve passar e seu formato é do tipo: *escrita(short <par1>, int <par2>)*. O primeiro parâmetro, *<par1>*, indicando o valor digital que é passado para o dispositivo conversor que é externo ao DSP e que será convertido – com resolução de 8 bits – para analógico, através do canal definido por *<par2>*, que pode variar de 1 a 4.

Um exemplo de aplicação, onde um sinal analógico é lido no canal 1 do dispositivo conversor analógico-digital, armazenado em uma variável localizada em uma posição de memória do DSP e depois escrito no dispositivo conversor digital-

analógico, é demonstrado no código exemplo, em linguagem C, a seguir. Novamente, neste exemplo, o projetista do sistema de controle não necessita conhecer os registradores e endereços utilizados pelos dispositivos conversores, bastanto incluir e utilizar as funções incluídas nas rotinas básicas, em linguagem de alto nível, em sua aplicação de controle.

```
short dado;
int canal;
void main(void)
{
      inicia_b();
      canal = 1;
      dado = leitura(canal); // leitura do canal 1 do ADC
      escrita(dado, canal);
}
```

4.5.6 Programação dos temporizadores

No capítulo 3 deste trabalho, sobre sistemas de controle em tempo real, foi discutida a possibilidade e vantagem da utilização dos temporizadores internos (quando disponível) do processador, no controle de um núcleo em tempo real. No texto, foi também demonstrada a vantagem, em desempenho, que este núcleo mínimo de um sistema operacional poderia obter fazendo uso do controle das interrupções de controle de execução de rotinas, por parte do projetista do sistema de controle, através do ajuste do tempo necessário que uma rotina necessita para executar seu código corretamente, como no caso da rotina de amostragem, como foi apresentado.

Uma maneira, na qual o controle da interrupção de uma rotina pode ser manipulado e alterado, foi incluído nas funções das rotinas básicas, através da possibilidade do projetista do sistema, configurar e habilitar uma função de acesso aos temporizadores internos do DSP56824, escolhido para o circuito do sistema baseado em DSP. A função *setaTimer0()*, *setaTimer1()* e *setaTimer2()* permitem o acesso e a alteração do valor utilizado pelos três temporizadores de 16 bits, disponíveis no sistema.

O formato desta função é *setaTimer (<par1>, <par2>)*, onde *<par1>* é o parâmetro que indica o valor de carga no registrador de Contagem do Timer escolhido e *<par2>* é o o valor passado para o registrador de Pré-Carga de Contagem do Timer escolhido.

Para que o projetista tenha disponível a interrupção gerada pelos temporizadores utilizados, é necessário associar a função de aplicação do sistema de controle por ele desenvolvido, ao vetor que inicia sua função a cada intervalo de tempo, regular. As funções, disponíveis no pacote de rotinas básicas, que permitem esta associação são: *intTimer0(<par3>)*, *intTimer1(<par3>)* e *intTimer2(<par3>)*, onde *<par3>*, é o parâmetro com o nome da função do projetista, que deverá ser acionada a cada intervalo de tempo. Este parâmetro é um ponteiro para a posição de memória, no segmento de memória de programa do sistema baseado em DSP, onde o código aplicativo se encontra instalado.

4.5.7 Inicialização de Interrupções

No capítulo 3, após a discussão de utilização do temporizador, para controlar e acionar rotinas em um intervalo de tempo pré-definido (determinístico), foi discutida a utilização de um recurso de programação – o semáforo – que permita informar ao núcleo e às demais rotinas do sistema, de que uma rotina encontra-se em execução. No caso do perfil mínimo de um núcleo, preferiu-se utilizar o recurso usual, incluído na forma de funções nas rotinas básicas, que desabilita, e posteriormente ao término da execução da rotina de interrupção, habilita o serviço de interrupções do sistema à sua operação normal.

As duas funções disponíveis ao projetista são *setaint()* e *resetaint()*. Estas funções, alteram o valor do registrador SR (Status Registrer) do processador DSP e do registrador de prioridade de interrupção (IPR, Interrupt Priority Register). Em ambos os

casos, o funcionamento da função é transparente ao projetista do sistema de controle. Para isso, faz-se a escolha da interrupção de interesse (entre os temporizadores disponíveis) e executa-se a função de inicialização do temporizador. O código desenvolvido pelo projetista do controlador digital, que deverá ser executado pelo sistema, está localizado na função referente ao parâmetro que é passado para o vetor da interrupção.

Da mesma forma que as outras funções incluídas nas rotinas básicas do núcleo do sistema, as funções de configuração do temporizador, configuração da rotina a ser executada e o controle de interrupções, são implementadas pelo projetista do sistema de controle, em linguagem C, como mostra o código exemplo a seguir.

```
typedef unsigned short WORD;
WORD *dataptr;

void T2int(void);

void main(void)
{
val = 0;                  // valor do contador
PLval = 500;              // valor da pre carga
setaTimer2(val, PLval); // configuração do Temporizador 2
intTimer2((WORD)T2int); // nome da funcao acionada

      while(1);
}



void T2int()                    // rotina executada em
{                               // intervalos de tempo, pré-definido
resetaint();
.
.
código de controle do sistema, desenvolvido pelo projetista
.
.
setaint();
}
```

4.5.8 Comunicação inter-processador

Outro recurso discutido no capítulo 3, sobre *Sistemas de Controle em Tempo Real*, e que foi implementado dentro das rotinas básicas, é o que permite a comunicação inter-processadores, através do recurso de E/S serial disponível no DSP escolhido para o circuito do sistema. Desse modo, duas placas distintas podem comunicar-se utilizando o recurso de Interface Serial Síncrona (SSI), que está tornando-se usual nos novos processadores dedicados. Além de permitir a expansão do sistema, através da conexão com dispositivos externos, como sensores de temperatura, CODECs e conversores que utilizam este padrão físico e protocolo de transmissão como recurso de E/S.

As funções implementadas e disponíveis ao projetista, através das rotinas básicas, são: *inicia_s()*, *recebe()* e *transmite()*. A função *inicia_s()* inicializa a porta C, quanto à utilização de seus pinos e registradores alocados para E/S serial, do tipo SSI. Através do formato *<parret1> = recebe()*, é possível passar o conteúdo do dado recebido, para uma variável do sistema, ou transmitir um dado através da passagem do parâmetro com o conteúdo do dado a ser transmitido pela função, que têm o formato *transmite(<par1>)*.

4.5.9 Tolerância à falhas

Outro recurso discutido no capítulo 3, e necessário em sistemas operacionais em tempo real, que foi disponibilizado para uso pelo projetista, no núcleo do sistema baseado em DSP, através das rotinas básicas, é a função que permite acionar e manter em funcionamento a unidade de temporização interna COP (*Computer Operanting Properly*). Esta função monitora a correta execução do processador incrementando um contador específico e dissociado dos temporizadores 1, 2 e 3 do processador, mantém-se

em funcionamento, mesmo em casos de pane do sistema, desde que a alimentação do circuito e o sistema do relógio, continuem operando normalmente.

Isto significa que em intervalos de tempo regulares caso o valor do contador de COP não seja limpo e atinja o limite de seu valor de contagem de 16 bits, uma interrupção de reinicialização do processador é gerada. Caso o sistema esteja funcionando corretamente e a rotina de controle desenvolvida pelo projetista, permitir atualizar (limpar) o conteúdo de COP, em intervalos regulares de tempo, o processamento é executado de forma natural. Caso ocorra alguma falha ou pane no sistema (por exemplo, devido ao ruído excessivo, gerado pelo dispositivo controlado), o sistema pode travar sua execução e ao ficar parado, não irá atualizar o valor de COP, levando esta unidade do processador à execução da reinicialização do sistema, recarregando a configuração e re-executando o código de controle. O acesso do recurso de COP é feito através da função *insCOP()*, e execução da função *copok()* pelo sistema, em intervalos de tempo regulares, que podem estar associados, p.ex., com o acesso de interrupção de uma rotina de controle, efetuada pela função de interrupções do temporizador.

Todas as funções discutidas neste capítulo, foram implementadas como rotinas básicas no sistema monitor e seus respectivos códigos encontram-se no Apêndice 1.

5 IMPLEMENTAÇÃO DO CONTROLE DO MOTOR DE CORRENTE CONTÍNUA

5.1 Introdução

Acionamentos de motores de corrente contínua são empregados em várias aplicações tais como em robôs manipuladores industriais. Quando o manipulador do robô muda sua posição ou pega objetos, o sistema de acionamento realimentado de cada junta é afetado por distúrbios de força. Estes distúrbios são: a força da gravidade, a carga, a força de Coriolis, a força centrifuga, a força de atrito e outras forças permanentes [25], além das não linearidades associadas aos sistemas mecânicos [1], resultando em erros de resposta frente aos comandos de controle.

A literatura apresenta algumas propostas de sistemas de controle robusto e de controle adaptativo aplicáveis ao acionamento de motores de corrente contínua com o objetivo de eliminar tais distúrbios [13][19][28][44]. Há uma proposta de solução mista, com o controlador implementado em processador digital e o circuito de supressão do distúrbio, implementado em eletrônica analógica [25], quando um controlador digital não consegue realizar a supressão de distúrbios em tempo real. Nesses casos, uma abordagem que permite a obtenção de uma estrutura simples para a supressão dos distúrbios de torque é a utilização de um Observador de Distúrbios de Torque (*Disturbance Torque Observer*, DTO) [24][26]. O diagrama de um controle implementado com o observador de torque é mostrado na Fig. 31, onde o controlador e o observador podem ser implementados em um sistema baseado em DSP, indicado pelos módulos envolvidos pela linha tracejada no diagrama de blocos.

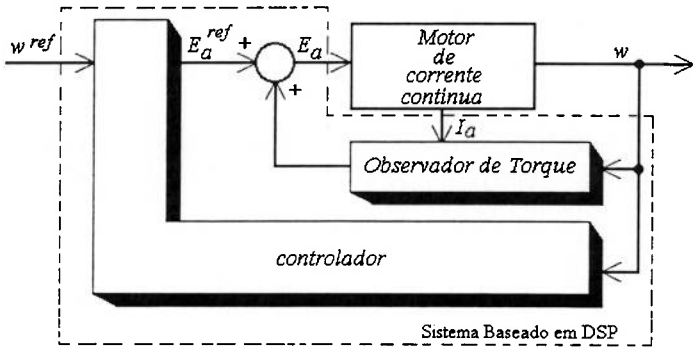


Fig. 31: Diagrama de controle implementado com o observador de torque

O sistema de acionamento para o motor de corrente contínua, é apresentado no diagrama de blocos da Fig. 32. Neste sistema, o motor é acionado por amplificador de tensão com ganho A_v. sendo o sinal de referência de velocidade, ω^{ref}, indicado por $V_{entrada}$, a constante de torque é indicada por k_t, e é numericamente igual à constante elétrica k_e, J é a constante de inércia mecânica do sistema.

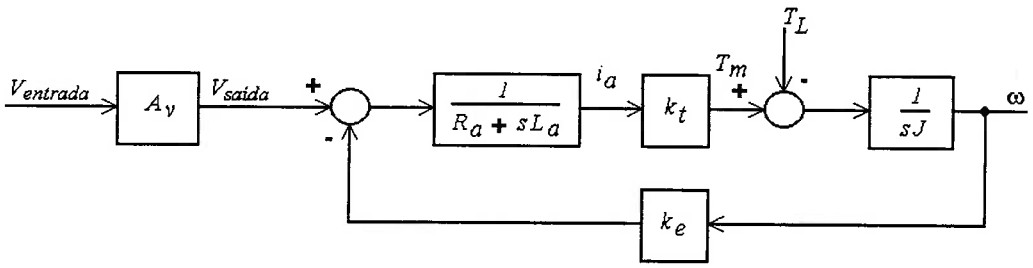


Fig. 32: Diagrama de um acionamento, de um motor de corrente contínua, utilizando um amplificador de tensão

Na Fig. 32 o torque gerado pelo motor (T_m) é obtido pelo produto de i_a e k_t, onde i_a é a corrente de comando do motor e T_L é o torque de carga, que representa a soma dos torques impostos ao sistema, como o torque de reação e os distúrbios de torque. Uma maneira para especificar os parâmetros do sistema é considerar que a velocidade do motor deve seguir o comando de entrada (velocidade de referência) ω^{ref}

assintoticamente, sem algum erro em regime permanente, isso pode ser obtido através de um controle PI convencional, apresentado na (16).

$$C(s) = k_1 \cdot \frac{T_1 s + 1}{T_1 s} \tag{16}$$

5.2 O Observador de Distúrbios de Torque

Segundo metodologia da teoria geral de controle alguns subsistemas de dois graus de liberdade possuem o recurso de que se pode projetar as características de resposta ao comando de entrada e as características de malha fechada independentemente. Apesar deste fato ser conhecido por mais de 45 anos, poucos trabalhos de aplicação em sistemas de controle foram realizados até aproximadamente o início da década de 90 [44].

Sob tal metodologia pode-se assumir que a corrente do motor de corrente contínua já esteja sendo controlada, então a função de transferência de um motor de corrente contínua pode ser obtida como:

$$\frac{\Omega(s)}{I(s)} = \frac{k_t}{Js} \tag{17}$$

Isso significa utilizar na implementação do observador de distúrbios de torque um acionamento do motor de corrente contínua através de um amplificador de corrente, A_i. Na Fig. 33, pode-se observar o modelo deste sistema, com a entrada T_L.

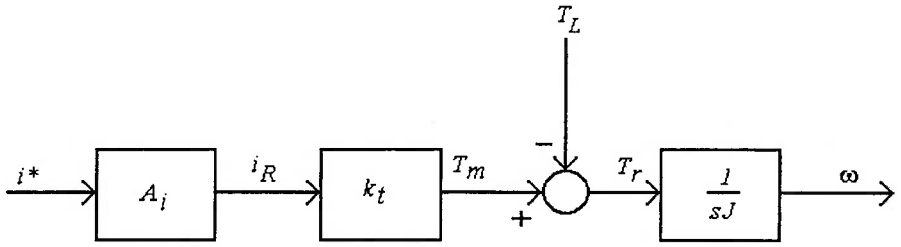


Fig. 33: Malha de corrente

Da representação da Fig. 33, podem-se obter as equações dinâmicas do sistema, apresentadas na (18) a seguir:

$$sJ\omega = T_m - T_L \tag{18}$$

Para realizar o cálculo da dinâmica do sistema, devem-se indicar as variações de inércia e do coeficiente de torque, onde Δ representa a diferença entre a dinâmica nominal do motor modelado $H_m(s)$ e a dinâmica do motor atual, $H(s)$. O que resultará na (19) e (20):

$$J = J_n + (J - J_n) = J_n + \Delta J \tag{19}$$

e,

$$k_t = k_{tn} + (k_t - k_{tn}) = k_{tn} + \Delta k_t \tag{20}$$

Onde J_n é a inércia nominal e k_{tn} é a constante de torque nominal do motor.

Normalmente o observador é considerado similar ao diagrama de blocos apresentado na Fig. 34, ou seja há uma equivalência física de um observador de torque, sendo implementado através da inserção de um filtro passa-baixas no sistema, uma vez que o observador de distúrbios de torque é um filtro passa-baixas. Esta também é a conclusão em relação ao DTO implementado no acionamento do robô Ariel [10], que será utilizado como referência para a implementação neste sistema baseado em DSP.

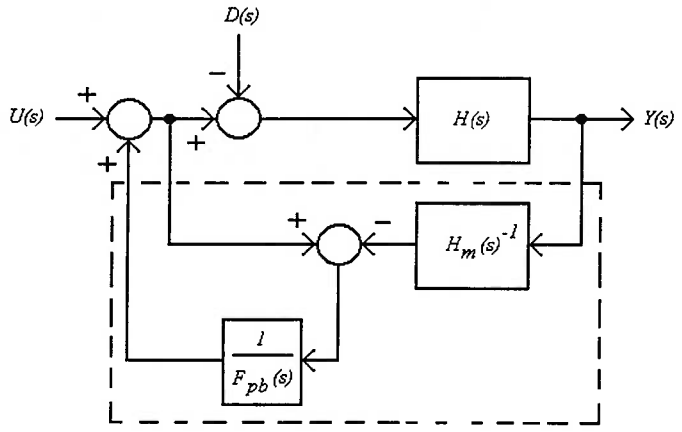


Fig. 34: Diagrama da equivalência física de um observador de torque

Em sua modelagem, o observador baseado no sinal de aceleração foi alterado ficando referenciado ao sinal da velocidade. Segundo o autor [10], o sinal de corrente nominal de armadura, i_a, não é afetado pela introdução do observador modificado do robô Ariel e fica sujeito apenas ao atraso introduzido pelo filtro passa-baixas, cuja frequência de corte é dada por τ_f, conforme apresenta o diagrama do sistema realimentado do observador na Fig. 35. A realimentação dos distúrbios de torque estimados tem como efeito a supressão dos distúrbios e o aumento da robustez da malha de controle do sistema. Quanto à robustez do controle do sistema esta é demonstrada e discutida em [19] para a variação dos valores do filtro no sistema.

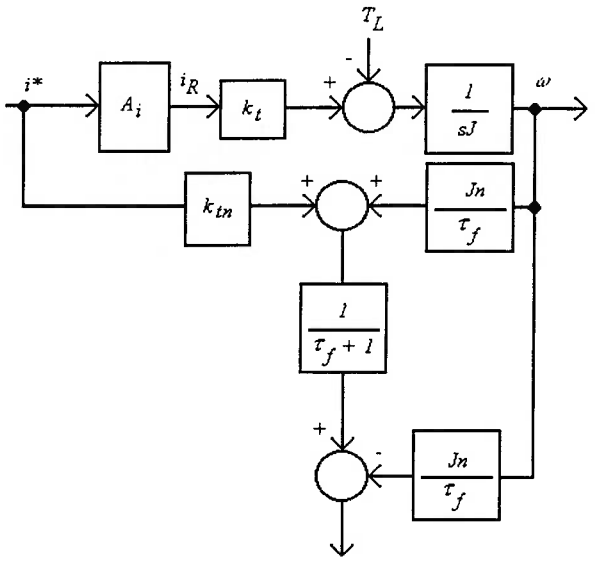


Fig. 35: Observador modificado baseado no sinal de velocidade

5.3 Implementação do Observador de Distúrbios de Torque

O sistema do presente projeto compreende o modelo do motor de corrente contínua associado ao compensador de distúrbios (DTO) devendo incluir um controlador de velocidade. Assim é necessário adaptar o modelo que foi apresentado, considerando o bloco do motor de corrente contínua apresentado anteriormente, onde o compensador foi aplicado, inserindo um controlador PI na malha externa do sistema realimentado.

Uma malha interna de corrente foi adicionada de forma a permitir que o sistema do motor implementado, que é controlado por um amplificador de tensão, possua as características de controle por um amplificador de corrente, sendo possível desta maneira a aplicação do DTO. Há na topologia de controle um ganho proporcional (A_i) fazendo com que o sinal de comando V_a seja diretamente proporcional ao sinal de erro obtido pela subtração de i_{ref} de i_m (corrente medida no motor de corrente contínua) e ao ganho do conversor tensão-frequência (A_i^*) utilizado no acionamento do robô Ariel

[10]. O sistema do observador de distúrbios de torque é o modelo do diagrama apresentado na Fig. 36.

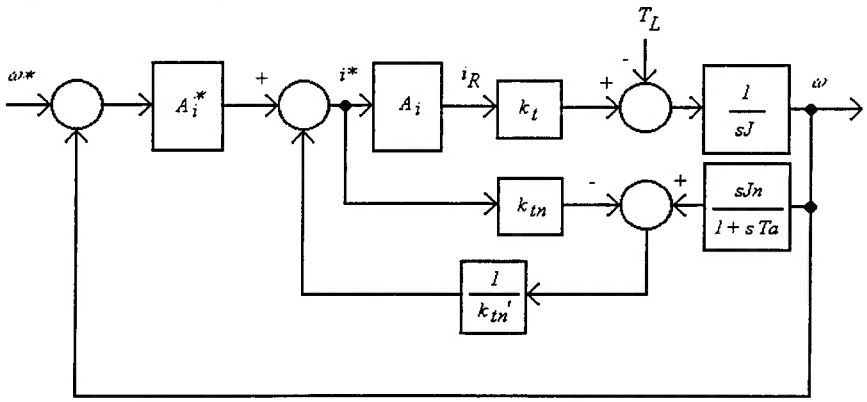


Fig. 36: Modelo aplicado no robô móvel Ariel

A partir deste diagrama de blocos do sistema modificado foi implementado um controlador digital para um motor com valor de comando como referência para o amplificador de tensão conforme mostra a Fig. 37. A área pontilhada do sistema representa o que foi implementado em um sistema baseado em DSP, apresentado no capítulo 4. O motor de corrente contínua utilizado para os ensaios e testes do sistema, foi implementado através de uma planta em um circuito analógico que simula o funcionamento deste motor, sendo composta de onze amplificadores operacionais que executam o modelo do motor de corrente contínua apresentado no capítulo 2. O protótipo implementado e seus pontos de entrada e saída são apresentadas na foto da Fig. 38. O esquema elétrico do circuito do simulador do motor de corrente contínua encontra-se no Apêndice 4.

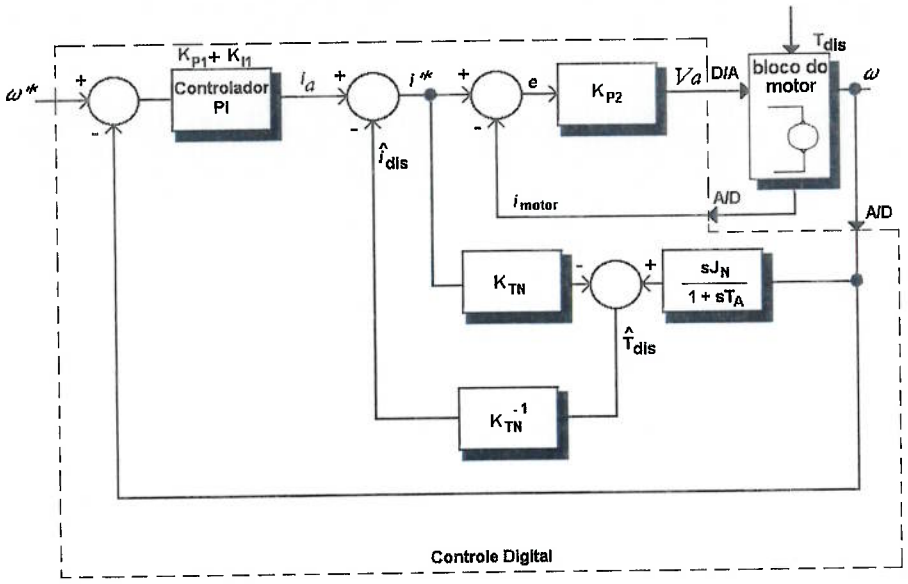


Fig. 37: Diagrama de blocos do modelo do DTO implementado no controlador digital

Fig. 38: Protótipo da planta analógica do motor de corrente contínua

As curvas de resposta da simulação do bloco do motor de corrente contínua através do programa Matlab/Simulink e a resposta da planta analógica do circuito implementado para uma entrada degrau são vistas na Fig. 39. A resposta do sistema, quando o algoritmo do controlador PI está ativo sobre a planta, pode ser observada na Fig. 40.

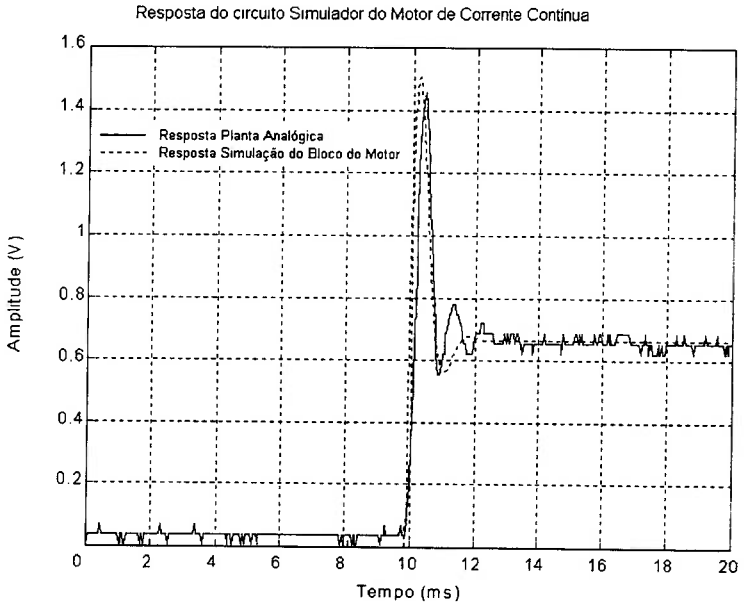


Fig. 39: Resposta da planta e da simulação do modelo do motor, para uma entrada degrau

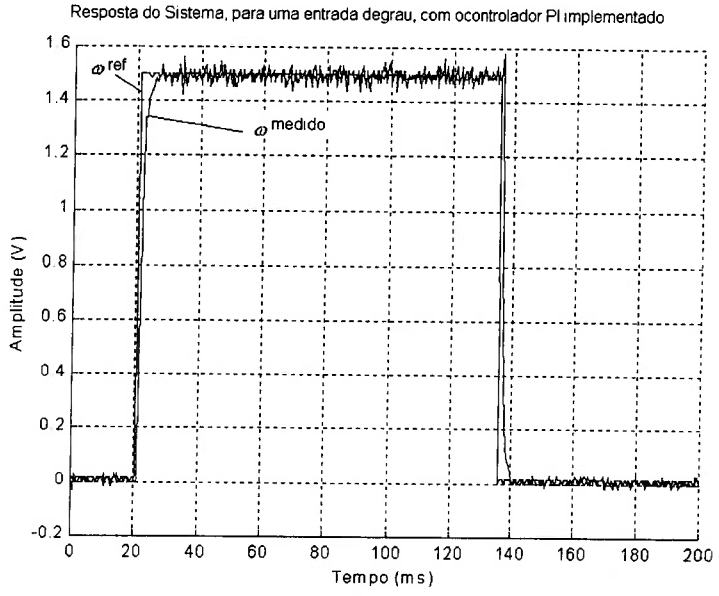


Fig. 40: Resposta do sistema para entrada degrau, com o controlador PI ativo

5.4 Implementação do Observador de Distúrbios no Sistema Baseado em DSP

A Fig. 41 apresenta as variáveis necessárias para desenvolver as equações que permitem a implementação do algoritmo do observador, em um sistema baseado em DSP.

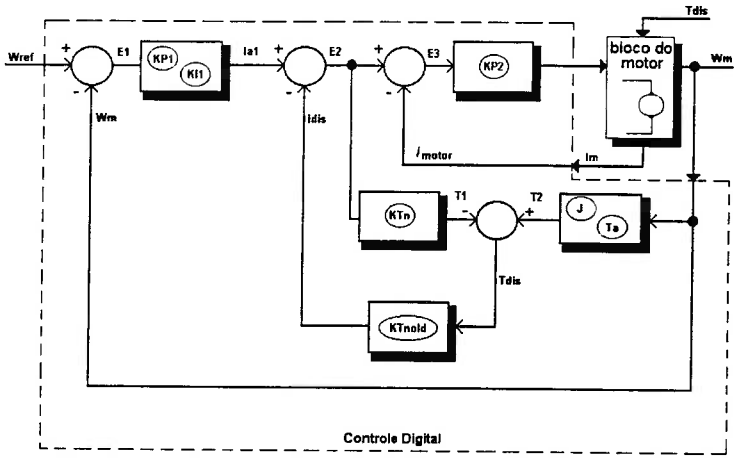


Fig. 41: Diagrama de blocos do observador implementado em algoritmo no controlador

As equações de diferença do controlador PI, são descritas em (21) e (22) para a corrente i_{a1}.

$$i_{a1} = k_{P1} \cdot E_1 + K_{I1} \int E_1 dt \tag{21}$$

$$\frac{di_{a1}}{dt} = k_{P1} \cdot \frac{dE_1}{dt} + K_{I1} \cdot E_1 \tag{22}$$

Considerando que o sistema pode ser discretizado pela regra de Euler, resulta:

$$\frac{i_a(k)-i_a(k-1)}{T_s}=k_{P1}\left[\frac{E_1(k)-E_1(k-1)}{T_s}\right]+K_{I1}\cdot E_1(k) \tag{23}$$

Substituíndo (21), resulta em i_a(k) da (24):

$$i_a(k)=i_a(k-1)+\left(k_{P1}+K_{I1}\cdot T_s\right)E_1(k)-k_{P1}\cdot E_1(k-1) \tag{24}$$

Agora, para desenvolver i_{dis}(k), temos:

$$i_{dis}(k)=k_m^{-1}\cdot T_{dis}=k_m^{-1}\cdot\left(T2-T1\right) \tag{25}$$

com,

$$T1=k_m\cdot E_2(k-1) \tag{26}$$

e,

$$T2(k)\cdot\left(T_s+T_A\right)=J_n\left(\frac{\omega_m(k)-\omega_m(k-1)}{T_s}\right)+T_A\cdot T_2(k-1) \tag{27}$$

Resultando em:

$$T2(k)=\left(\frac{J_n}{T_s+T_A}\right)\cdot\left(\frac{\omega_m(k)-\omega_m(k-1)}{T_s}\right)+\left(\frac{T_A}{T_s+T_A}\right)\cdot T_2(k-1) \tag{28}$$

Os componentes intermediários são:

$$E_1(k)=\omega_{ref}(k)\cdot\omega_m(k) \tag{29}$$

$$E_2(k)=i_{a1}(k)\cdot i_{dis}(k) \tag{30}$$

$$E_3(k)=E_2(k)\cdot i_m(k) \tag{31}$$

$$V_a(k)=k_{P2}\cdot E_3(k) \tag{32}$$

O sistema implementado para realização dos experimentos e obtenção dos resultados do controlador com o observador de distúrbios de torque, é mostrado na Fig. 42. Foi utilizado um Gerador de Sinais como fonte do torque de carga T_L (distúrbios de torque). O circuito do Sistema Baseado em DSP foi conectado à planta do simulador do motor de corrente contínua através de três pontos. A aquisição das tensões proporcionais aos sinais de corrente do motor (I_a) e da velocidade (ω) foram adquiridas através de dois canais do conversor analógico-digital e o sinal de comando do motor (V_a) foi imposto à planta através de um canal do conversor digital-analógico. Os códigos implementados, com os algoritmos do controlador PI e do observador de distúrbios de torque, encontram-se no Apêndice 6.

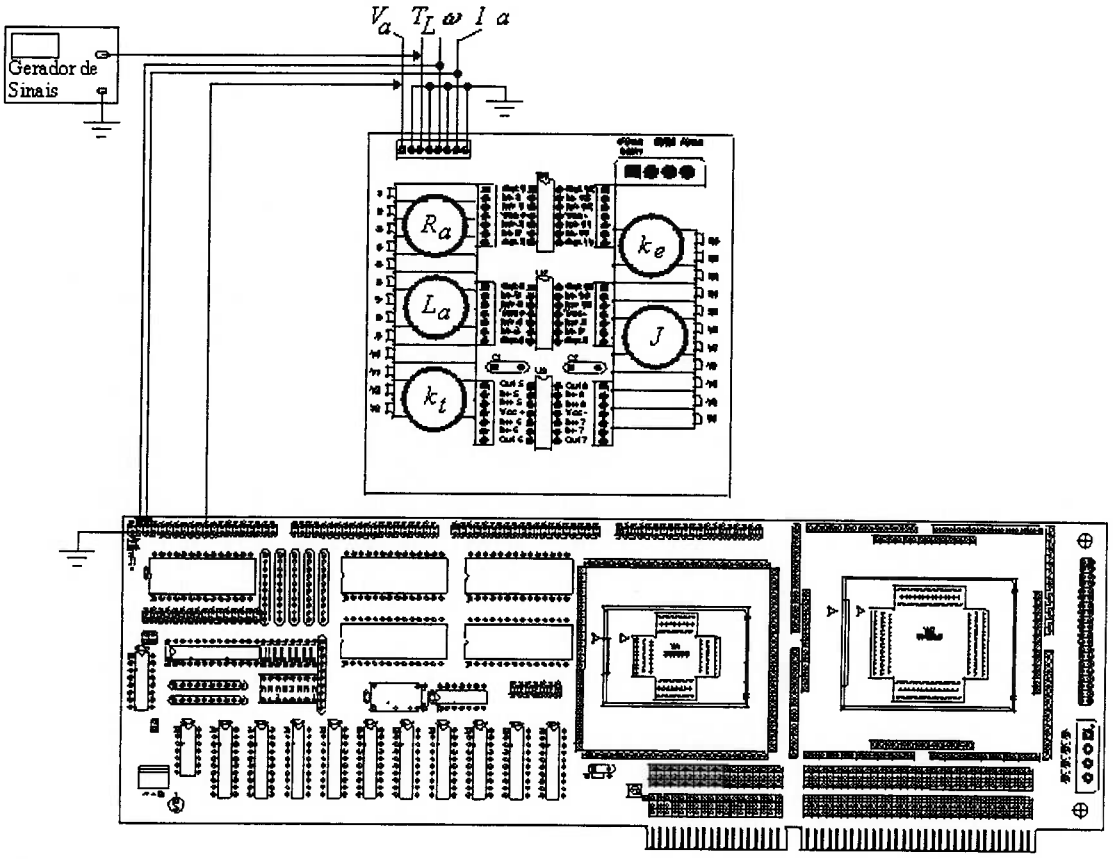


Fig. 42: Esquema de conexões do sistema completo utilizado para obtenção dos resultados experimentais

Da Fig. 43 à Fig. 46, pode-se observar a resposta do sistema frente uma entrada degrau aplicada à velocidade de referência (ω^{ref}) do sistema de controle, com amplitude de 1,5V. Os resultados demonstram a implementação de algoritmos de controle PI, em linguagem C, com e sem a utilização do observador de distúrbios de torque. Na Fig. 43 a entrada degrau é aplicada no sistema, que está sem Observador de Distúrbios de Torque (DTO). A resposta do sistema aponta a sobreelevação do sistema sem o controle implementado. Na Fig. 45 é possível observar a resposta do sistema, onde junto ao algoritmo de controle PI, no controlador digital, ocorre a supressão do torque de carga TL imposto ao motor de corrente contínua, que neste caso é de 200mV, para o degrau de entrada de velocidade de 1,5V. Já na Fig. 44, para o mesmo degrau de entrada (ω^{ref}) e torque de carga (T_L), a resposta do sistema para o algoritmo do controle PI sem o DTO implementado. Na Fig. 46, com o sistema de controle atuando com os dois algoritmos, de controle PI e DTO, é possível observar a supressão do ruído, como no caso anterior, porém com um torque de carga T_L aplicado ao motor de corrente contínua, de quase duas vezes sua amplitude, ou seja, 388 mV. Os experimentos foram realizados para uma frequência de acionamento do motor de corrente contínua de 4 Hz e uma frequência, para o torque de carga imposto, de aproximadamente 36 Hz. As figuras deixam claro que, o o circuito do sistema foi implementado apenas para valores positivos na planta, sem a presença de um circuito conversor do nível das entradas analógicas do conversor analógico-digital, que possui a excursão de suas entradas entre 0 e 5V.

O resultado obtido, com o DTO implementado em sistema baseado em DSP, mostra que o mesmo algoritmo quando implementado com um circuito de ajuste para os valores positivos e negativos da planta, conseguirá suprimir todos os distúrbios de torque para acionamento do motor elétrico, em ambos os sentidos.

Tabelas e referências com valores utilizados no simulador do motor – a relação de tensão na entrada da planta analógica do motor, proporcional à tensão de armadura do motor, e a tensão de saída, proporcional à velocidade – baseados nos dados de um motor de corrente contínua comercial (WEG DNF132.130S) encontram-se no Apêndice 4.

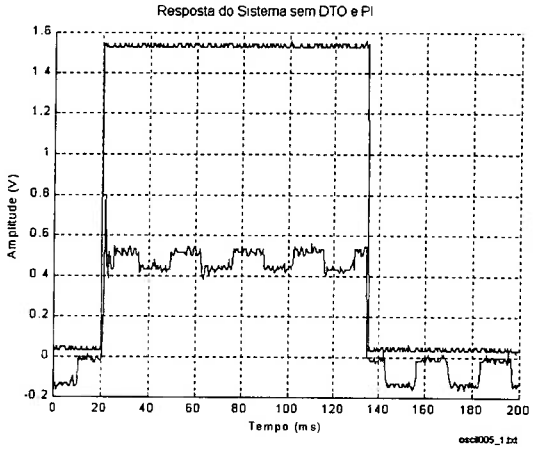


Fig. 43: Resposta do sistema sem observador de disturbios de torque e regulador PI

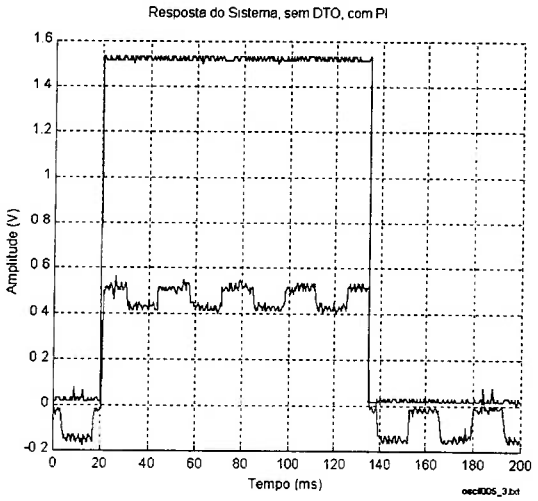


Fig. 44: Resposta do sistema sem observador de disturbios de torque, com regulador PI

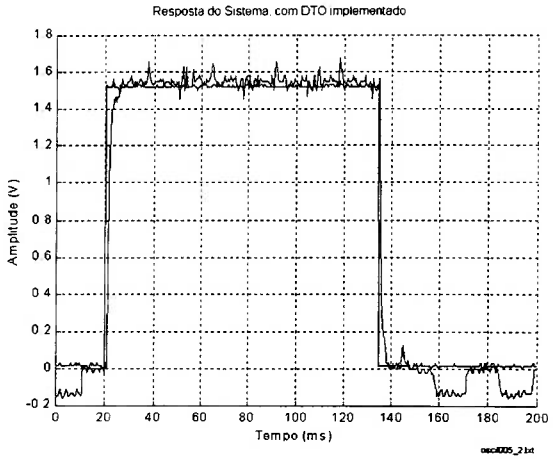


Fig. 45: Resposta do sistema com observador de disturbios de torque e regulador PI

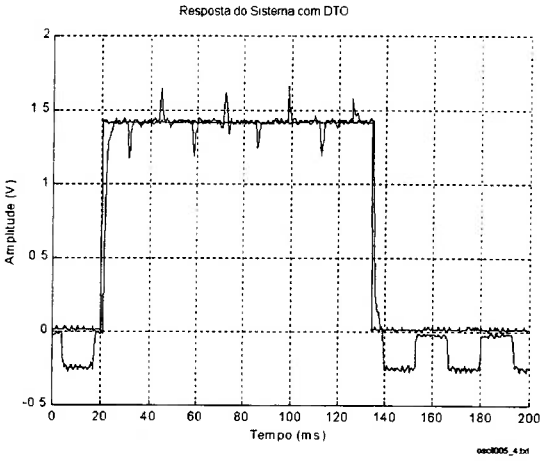


Fig. 46: Resposta do sistema com observador de disturbios de torque e regulador PI

A foto do sistema utilizado para realizar os experimentos e comprovar os resultados é apresentada na Fig. 47.

Fig. 47: Estrutura utilizada no desenvolvimento do sistema baseado em DSP

6 CONSIDERAÇÕES FINAIS

A motivação para o desenvolvimento deste trabalho, conforme apresentado no capítulo 1, foi a de obter um sistema baseado em DSP permitindo a implementação e o ensaio de várias estratégias de controle de motores elétricos, em tempo real. O capítulo 2, apresentou e discutiu as características do motor de corrente contínua, tendo sido abordado o modelo deste motor, sua construção e seu acionamento e controle. No capítulo 3, foram discutidas as características de um sistema de controle em tempo real, através do estudo das características de sistemas operacionais padronizados. No capítulo 4 foram apresentadas as características de um sistema baseado em DSP, mostrando o desenvolvimento, composto de um protótipo implementado e seu sistema monitor, bem como suas funções disponíveis para o projeto de controladores digitais. No capítulo 5 foi apresentada uma aplicação envolvendo o controle e acionamento de um motor de corrente contínua, utilizando uma planta simuladora especialmente desenvolvida para esta finalidade, a qual foi desenvolvida no sistema DSP contendo uma estratégia de controle baseada em um controlador PI digital, com a adição de um observador de distúrbios de torque (DTO). Resultados da implementação deste controlador digital foram apresentados, onde pode ser observada a eficiência do DTO na supressão dos distúrbios de torque do motor elétrico. Foi demonstrada a viabilidade de implementação através do protótipo do sistema baseado em DSP, permitindo que mudanças de valores nos parâmetros do sistema comprovassem a robustez da estratégia de controle.

Estes fatos, demonstram que:

- O sistema digital realizado permite a obtenção dos mesmos resultados encontrados em sistemas de controle analógico, como o DTO do acionamento do robô Ariel, todavia com muito maior flexibilidade;
- O sistema baseado em DSP permite a realização do controle em tempo real de motor elétrico, além de permitir que novas estratégias de controle sejam implementadas;
- A utilização de funções existentes nas rotinas básicas do sistema flexibiliza o projeto, facilitando sua implementação e diminuindo o tempo necessário para a sua realização, e
- Novas estratégias de controle, para motores elétricos, podem ser desenvolvidas utilizando a estrutura que foi criada.

A utilização de uma planta simuladora do motor de corrente contínua, permitiu configurar as constantes do motor para um caso de "sistema real" e facilitou a implementação do controle do motor, sem utilizar um módulo de potência para o acionamento. Com os resultados encontrados, pode-se transferir o controle da planta para um motor 'real', utilizando este sistema para o desenvolvimento de estratégias de controle mais complexas, a partir da estrutura desenvolvida ao longo deste trabalho.

Devido à grande diversidade de temas, plantas e processos controlados por sistemas digitais, as seguintes sugestões de trabalhos futuros restringem-se apenas ao controle de motores elétricos em tempo real:

- Implementação de estratégias de controle para motores elétricos especiais como o motor pentafásico do projeto do motor embutido em roda além de outros motores, tais como motor de indução e motor síncrono;

- A utilização de um microcomputador hospedeiro, conforme discutido no decorrer deste trabalho também permite que sejam explorados a construção de um ambiente de desenvolvimento, que esteja conectado ao sistema baseado em DSP, permitindo ao projetista desenvolver o controle através de uma interface gráfica a exemplo do que já ocorre com os sistema de desenvolvimento na área de telecomunicações, sendo utilizados ambientes como o Simulink/Matlab e a partir dele, gerando código para o circuito do controlador para controle em tempo real;
- A existência de uma pilha (*stack*) baseada em memória utilizada no sistema que foi desenvolvido permite o chaveamento de tarefas, com tarefas e interrupções reentrantes, levando ao desenvolvimento e à construção de um sistema operacional robusto e multi-tarefa em tempo real;
- O desenvolvimento de trabalhos que permitam o multi-processamento, inter-sistemas, de estratégias de controle em mais de um processador DSP.

A expectativa, após a conclusão deste sistema baseado em DSP, flexível, de baixo custo e de excelente desempenho, para controle de motores elétricos, permite afirmar que o esforço de pesquisadores ao gerar ferramentas de desenvolvimento e apoio ao projetista de sistemas de controle, continua válido e necessário, apesar da grande variedade de sistemas comerciais atualmente disponíveis.

REFERÊNCIA BIBLIOGRÁFICA

[1] Dote, Y., "Servo Motor and Motion Control Using Digital Signal Processors", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1990.

[2] Keuchel, U. and Stephan, R. M., "Microcomputer-based adaptive control applied to thyristor-driven DC-motors", Springer-Verlag London Limited, 1994.

[3] Aström, K. and Hägglund T., PID Controllers: Theory, Design and Tuning, Instrument Society of America, 2nd. Ed., 1995.

[4] Tang, J., "Laboratory Development for a Digital Control Systems Course", Journal of Engineering Technology, Fall 1997, ASEE, pp. 8-13

[5] Tamaki, K., Ohishi, K., Ohnishi, K. and Miyachi K., "Microprocessor based robust control of a dc servo motor", IEEE Contr. Sys. Mag. 1986, Vol. 6, No. 5, pp. 30-36

[6] Ohmae, T.; Matsuda, T.; Kanno, M., Saitoh, K. and Sukegawa, T., "A microprocessor based speed regulator using Fast Response state observer for motor drives", IEEE Industry App. Society Annual Meeting, 1986, pp. 271-276

[7] Simões M. G., "Acionamento e Controle de motor embutido em roda "wheel-motor" para veículos elétricos". Proposta de trabalho aprovada pela FAPESP, projeto FAPESP No. 98/11351-7. Julho, 1998

[8] Szafir, S., "Sistema DSP-PC: Manual do Usuário", Relatório interno do Depto. de Eng. Mecatrônica e de Sistemas Mecânicos da EPUSP, Universidade de São Paulo, Agosto, 2000.

[9] Szafir S. e Simões M. G., "Proposta de Sistema Simulador da Dinâmica de Veículo Elétrico Baseado em Processador Digital de Sinais", 4o. SBAI, 08-10 setembro 1999.

[10] Chiaramonte, M. J., "Aplicação do Observador de Distúrbios de Torque no Robô Móvel Ariel, Dissertação de Mestrado apresentada à Escola Politécnica da USP, São Paulo, 1993.

[11] Benchaib, A.; Rachid, A., Audrezet, E. and Tadjine, M. "Real-Time Sliding-Mode Observer and Control of an Induction Motor", IEEE Trans. on Industrial Electronics, February 1999, Vol. 46, No. 1, pp. 128-138.

[12] Naitoh, H. and Tadakuma, S., "Microprocessor-Based Adjustable-Speed DC Motor Drives Using Model Reference Adaptative Control", IEEE Trans. on Industry Applications, march/april 1987, Vol. IA-23, No. 2, pp. 313-318.

[13] Matsui, N. and Ohashi, H., "DSP-Based Adaptative Control of a Brushless Motor", IEEE Trans. on Industry Applications, march/april 1992, Vol. 28, No. 2, pp. 448-454.

[14] Guo, Y.; Lee, H. C. and Ooi, B., "Extensible Digital-Signal-Processing Modules for Real-Time Control and Simulation", IECON 93 Vol. III, 1993, pp. 2229-2234

[15] Samudio, R. e Pillay, P., "Design of a Flexible DSP-Based Drive Development System", IEEE, IAS Meeting 95 Vol. III, 1995, pp. 1854-1861

[16] Huh, U.; Lee, J. and Lee, T., "A torque control strategy of brushless DC motor with low resolution encoder", Power Electronics and Drive Systems, 1995, proceedings, 1995, pages 496-501 vol. 1.

[17] du Toit, J.A.; Enslin, J.H.R. and Spée, R., "Experimental evaluation of digital control options for high power electronics", IECON 21st, 1995 Industrial Electronics, Control & Instrumentation proceedings, 1995, vol. 1, pages 674-679.

[18] Marchesoni, M.; Sciutto, G.; Segarich, P; Soressi, E. and Taffone, A, "High computational power and great interfacing capability for electric drives control: a new surface-mount DSP based system", MELECON'96 Electrotechnical Conference, 1996 8th Mediterranean, 1996, vol. 1, pages 346-349.

[19] Tomita, M.; Senjyu, T.; Doki, S. and Okuma, S., "New Sensorless Control for Brushless DC motors Using Disturbance Observers and Adaptative Velocity Estimations", IEEE Trans. on Industrial Electronics, April 1998, Vol. 45, No. 2, pp. 274-282.

[20] Hong, K. and Nam, K., "A Load Torque Compensation Scheme Under the Speed Measurement Delay", IEEE Trans. on Industrial Electronics, April 1998, Vol. 45, No. 2, pp. 282-290.

[21] Mehta, S. and Chiasson, J., "Nonlinear Control of a Series DC Motor: Theory and Experiment", IEEE Trans. on Industrial Electronics, February 1998, Vol. 45, No. 1, pp. 134-141.

[22] Stern, M., "Universal Digital Motion Controller", EPE 89, Aachen 1989, pp. 1-6

[23] Dessaint, L. A., Hebert, B. J., Le-Huy, H. and Cavuoti, G., "A DSP-Based Adaptative Controller for a Smooth Positioning System", IEEE Transactions on Industrial Electronics, October 1990, Vol. 37, No. 5, pp. 372-377

[24] Chattopadhyay, A. K. and Meher, N., "Microprocessor Implemantation of a State Feedback Control Strategy for a Current Source Inverter-Fed Induction Motor Drive", IEEE Trans. on Power Applications, april 1989, Vol. 4, No. 2, pp. 279-288.

[25] Ohishi, K.; Ohnishi, K. and Miyachi, K., "Adaptative DC servo drive control taking force disturbance suppression into accountt", IEEE Trans. Industry Applications, January/February 1988, Vol. 24, No. 1, pp. 171-176.

[26] Miyagi, P. E.; Mizutani, T.; Chiaramonte, M. J. e Cozman, F. G., "Insensibilidade a Variações de Carga Através de Observador de Distúrbio de Torque", 11° Congresso Brasileiro de Eng. Mecânica, Dezembro, 1991, São Paulo, SP – Brasil, pp. 73-76.

[27] Ji, J. and Sul, S., "DSP-Based Self-Tuning IP Speed Controller with Load Torque Compensation for Rolling Mill DC Drive", IEEE Transactions on Industrial Electronics, August 1995, Vol. 42, No. 4, pp. 382-386.

[28] Cerruto, E.; Consoli, A.; Raciti, A. and Testa, A., "A Robust Adaptative Controller for PM Motors Drives in Robotic Applications", IEEE Trans. on Power Electronics, January 1995, Vol. 10, No. 1, pp. 62-71.

[29] Chiappa, C.; Hapiot, J. C. and Grandpierre, M., "Structure of a Torque Control Module for DC Motors", EPE, Aachen 1989, pp. 277-281.

[30] Joos, G.; Sicard, P. and Goodman, E. D., "A Comparison of Microcomputer-Based Implementations of Cascaded and Parallel Speed and Current Loops in DC Motor Drives", IEEE Industry Applications Society Annual Meeting, Atlanta, 1987, Part I, pp. 413-419.

[31] Gayakwad, R. and Sokoloff, L., "Analog and Digital Control Systems", Prentice-Hall, 1988.

[32] Buja, G. S., Menis, R. and Valla, M. I., "Disturbance Torque Estimation in a Sensorless DC Drive", IEEE Transactions on Industrial Electronics, August 1995, Vol. 42, No. 4, pp. 351-357.

[33] Almeida, P. E. M., "Implementação de um Ambiente de Desenvolvimento de Controladores Difusos e suas Aplicações em Controle de Processos Reais", Dissertação de Mestrado apresentada à PPGEE Escola de Engenharia U.F.M.G., 1996.

[34] Schiebe, M. Pferrer, S. (editors), "Real-Time Systems Engineering and Applications", Kluwer Academic Publishers, Boston, 1992.

[35] IEEE Std 610.12, "Glossary of Software Engineering Terminology", IEEE, New York, 1990.

[36] Laplante, P., "Real-Time Systems Design and Analysis, An Engineer's Handbook", IEEE Press, New York, 1993.

[37] Lawson, H. W., "Parallel Processing in Industrial Real-Time Applications", Prentice-Hall, Englewwod Cliffs, NJ, 1992.

[38] Dezza, F. C.; Cristaldi, L.; Ferrero, A. and Monti A., "Real time virtual system for electric drive testing: Basic Concepts and Implementation", MELECON'96 1996 8th Mediterranean Electrotechnical Conference, 1996, pages 513-516, vol. 1

[39] Wisnosky, D. E., "SoftLogic: Overcoming Funnel Vision", Wizdom Controls Inc., Naperville, IL, October, 1996.

[40] Lu, C. W., "Torque Controller for Brushless DC Motors", IEEE Trans. on IE, April 1999, Vol. 46, No. 2, pp.471-473.

[41] Sasaki T. M., "Controlador baseado em processador digital de sinais para o ensino de controle de robôs", Dissertação de Mestrado apresentada à Escola Politécnica da USP, São Paulo, 1997.

[42] Bose, B. K. and Szczesny, P. M., "Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System for Electric

Vehicle Propulsion", IEEE Transactions on Industrial Electronics, November 1988, Vol. 35, No. 4, pp. 547.

[43] Asano, K., Okada, S. and Iwama, N., "Vibration Suppression of Induction-Motor-Driven Hybrid Vehicle Using Wheel Torque Observer", IEEE Trans. on Ind. App., March/April 1992, Vol. 28, No. 2, pp. 441-447.

[44] Umeno, T. and Hori, Y., "Robust Speed Control of DC Servomotors Using Modern Two Degrees-of-Freedom Controller Design", IEEE Trans. Ind. Electron., October 1991, Vol. 38, pp. 363-368.

[45] Akbarzadeh-T., M., Kumbla, K. K., Medina E. and Kim, Y. T., "Evolutionary Fuzzy Control In Real-Time DSP Controlled Systems", Soft Computing with Industrial Applications: Recent Trends in Research and Development, Proceedings of the World Automation Congress (WAC'96), Montpellier, France, May 1996, Vol. 5, pp.7-12.

[46] Goureau, P., Sirier, D. and Bigand, A., "A Dynamic Fuzzy Controller for a D.C. Drive" , Soft Computing with Industrial Applications: Recent Trends in Research and Development, Proceedings of the World Automation Congress (WAC'96), Montpellier, France, May 1996, Vol. 5, pp.257-264.

[47] Dawson, F. P. and Klaffke, L., "Variable-Sample-Rate Delayless Frequency-Adaptative Digital Filter for Synchronized Signal Acquisition and Sampling", IEEE Trans. on IE, October 1999, Vol. 46, No. 5, pp.889-896.

[48] Berkeley Design Technology, Inc., "Buyer's Guide to DSP Processors, 1999 Edition", Berkeley Design Technology, Inc., 1999, Berkeley, CA.

[49] Ohishi, K.; Nakao, M., Ohnishi, K., Miyachi, K., "Microprocessor-controlled dc motor for load-insensitive Position Servo System", IEEE Trans. Ind. Electron., Feb. 1987, vol. IE-34, No. 1, pp. 44-49.

[50] Gurubasavaraj, K. H., "Implementation of a self-tuning controller using DSP chips", IEEE Control Systems Magazine, June 1989, pp.38-42.

BIBLIOGRAFIA RECOMENDADA

[1] Motorola Inc, "DSP56L811 User's Manual", Motorola Semiconductor Products Sector, DSP Division, Austin, TX, 1996.

[2] Motorola Inc, "DSP56L800 Family Manual", Motorola Semiconductor Products Sector, DSP Division, Austin, TX, 1996.

[3] Motorola Inc, "DSP56824 16-bit Digital Signal Processor Chip Specifications" , Motorola Semiconductor Products Sector, DSP Division, Austin, TX, Rev. 0.1 Release date: 4/24/97.

[4] Metrowerks Inc., "CodeWarrior Targeting DSP56800", Release 3.5, Metrowerks Codewarrior Documentation, Austin, TX, 2000.

[5] Anderson, D. and Shanley T., "ISA System Architeture", Third edition, Addison-Wesley Pub Co, MindShare, Inc., 1995.

[6] Zelenovsky, R. e Mendonça, A., "PC, Um Guia Prático de Hardware e Interfaceamento", Interciência, Rio de Janeiro, RJ, 1996.

[7] Labrosse, J. J., "MicroC/OS-II The Real-Time Kernel", R&D Books, Lawrence, KS, 1999.

[8] Peatman, J. B., "Design with Microcontrollers", McGrawHill Inc., 1988.

[9] Kavi, K. M., "Real Time Systems: Abstractions, Languages, and design", IEEE Computer Society, 1992.

[10] Dorf, Richard C. and Bishop Robert H., "Modern Control Systems", Eighth edition, Addison-Wesley Longman Inc, Menlo Park, CA, 1998.

[11] Cogdell, J. R., "Foundations of electric power", Prentice Hall, Upper Saddle River, N.J., 1999.

APÊNDICE 1: ROTINAS BÁSICAS

Rotinas básicas da placa DSP-PC, escritas em linguagem *assembly* do 56800.

Arquivo: rotinas.asm

```
SECTION rotinas

ORG     P:$0000                     ; deixar para GNU Assembler e
                                    ; para Codewarrior 2.3-
                                    ; retirar para Codewarrior 3.0+

GLOBAL Finicia_b                    ; funcao inicia_b()

Finicia_b:
move #$0000,X:$FFF9          ; ws = 0
move #$0000,X:$FFEA
move #$FFFF,X:$FFEC          ; PBD = 1
move #$0000,X:$FFEB          ; PB0..PB15 = saida
RTS

GLOBAL Fleitura                     ; funcao leitura(<par1>)
                                            ; retorna <parret1>

Fleitura:
move  #$3F00,X:$FFEB   ; PB0..PB6  = entrada
; PB7..PB15 = saida
move #$3F00,X:$FFEC          ; PB8..PB15 = 1
brset #$0008,Y0,CH8          ; .
brset #$0007,Y0,CH7          ; .
brset #$0006,Y0,CH6          ; .
brset #$0005,Y0,CH5          ; vai p/ config. canal
brset #$0004,Y0,CH4          ; .
brset #$0003,Y0,CH3          ; .
brset #$0002,Y0,CH2          ; .
brset #$0001,Y0,CH1          ; .
RTS
                                                        CH1:
bfclr #$0700,X:$FFEC   ; canal = 1
bra   LE
CH2:
bfclr #$0600,X:$FFEC   ; canal 2
bra   LE
CH3:
bfclr #$0500,X:$FFEC   ; canal 3
bra   LE
CH4:
bfclr #$0400,X:$FFEC   ; canal 4
bra   LE
CH5:
bfclr #$0300,X:$FFEC   ; canal 5
bra   LE
CH6:
bfclr #$0200,X:$FFEC   ; canal 6
bra   LE
CH7:
bfclr #$0100,X:$FFEC   ; canal 7
bra   LE
CH8:
bfclr #$0000,X:$FFEC   ; canal 8
bra   LE

LE:
bfclr #$0800,X:$FFEC   ; /CS = 0
bfclr #$1000,X:$FFEC   ; /RD = 0
nop                                 ; => que tRD (80 ns)
```

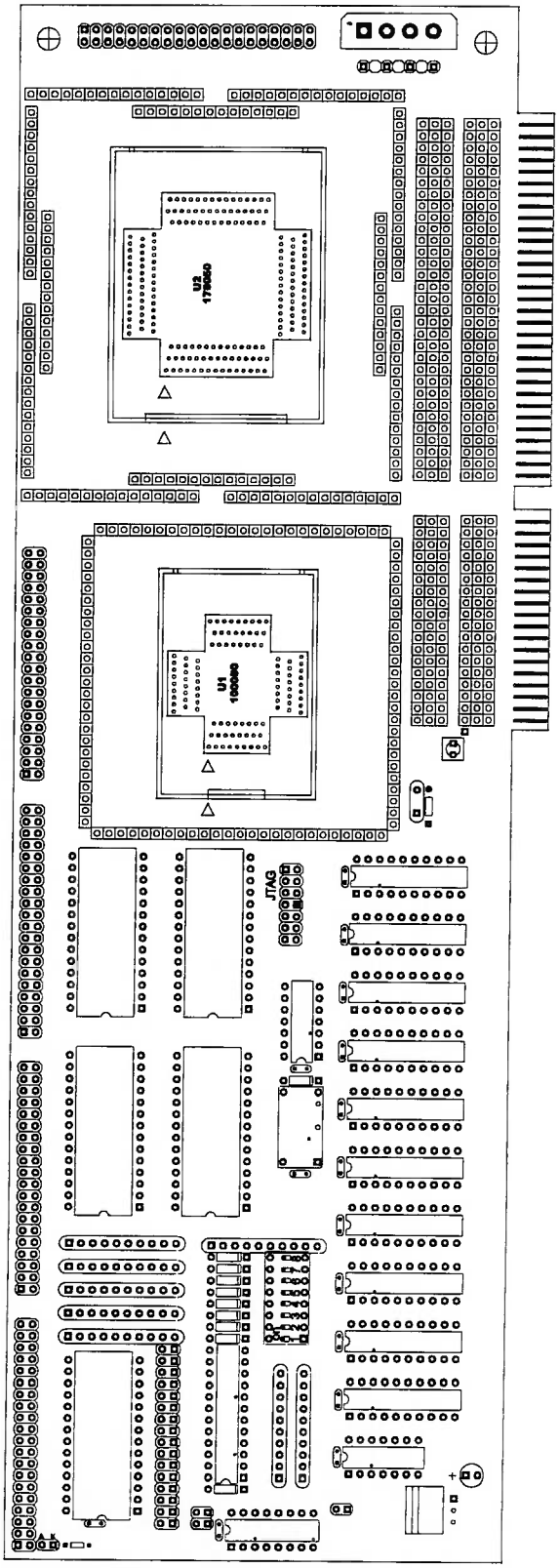
```
                                                ; + tACC1 (110 ns)
bfset #$1000,X:$FFEC   ; /RD = 1
move  #15,x0                    ; loop
; tP (500 ns, tip.)
; + tCRD (2,4 us, tip.)
CONT:
decw  x0
bgt   CONT                      ; loop total (2,9 us)
bfclr #$1000,X:$FFEC   ; /RD = 0
nop                                     ; delay
move  X:$FFEC,Y0                ; <parret1> = PBD
bfset #$1000,X:$FFEC   ; /RD = 1
bfset #$0700,X:$FFEC   ; A0 = A1 = A2 = 1
bfset #$0800,X:$FFEC   ; /CS = 1
andc  #$00FF,Y0                 ; formatacao <parret1>
RTS


GLOBAL Fescrita                 ; funcao escrita(<par1>,<par2>)

Fescrita:
move    #$3FFF,X:$FFEB          ; pinos PB0..PB15 = saida
bfset #$3F00,X:$FFEC   ;
brset #$0004,Y0,CHN4   ; vai p/ config. Canal 4
brset #$0003,Y0,CHN3   ; vai p/ config. Canal 3
brset #$0002,Y0,CHN2   ; vai p/ config. Canal 2
brset #$0001,Y0,CHN1   ; vai p/ config. Canal 1
bfclr #$0800,X:$FFEC   ; /LDAC = 0
RTS

CHN1:
bfclr #$0300,X:$FFEC    ; config. canal 1
     bra   ESCREVE
CHN2:
bfclr #$0200,X:$FFEC    ; config. canal 2
bra   ESCREVE
CHN3:
bfclr #$0100,X:$FFEC    ; config. canal 3
bra   ESCREVE
CHN4:
bfclr #$0000,X:$FFEC   ; config. canal 4
bra   ESCREVE
ESCREVE:
bfclr #$2000,X:$FFEC   ; /WR = 0
bfclr #$FF00,Y1                 ; formatacao do <par2>
move  Y1,X:$FFEC                ; PBD = <par2>
bfset #$2000,X:$FFEC   ; /WR = 1
bfset #$2F00,X:$FFEC   ; sinais = 1
RTS

GLOBAL Finicia_c                        ; funcao inicia_c()

Finicia_c:
move #$0000,X:$FFED             ; pinos porta C = GPIO
RTS

GLOBAL Finicia_s                        ; funcao inicia_s()

Finicia_s:
move #$0000,X:$FFF9             ; ws = 0
move #$3F00,X:$FFED             ; Config. Porta C = SSI
move #$6313,X:$FFD3             ; Tx = 16 B/W (7.2 kHz)
move #$6313,X:$FFD4             ; Rx = 16 B/W
RTS

GLOBAL Ftransmite                       ; funcao transmite(<par1>)

Ftransmite:
move #$1190,X:$FFD2             ; SCR2 = Tx
;move X:dado,Y0                 ; Tx frac. (desabilitado)
move Y0,X:$FFD0                 ; envia <par1>
RTS

GLOBAL Frecebe                          ; funcao recebe()
                                                ; retorna <parret1>
```

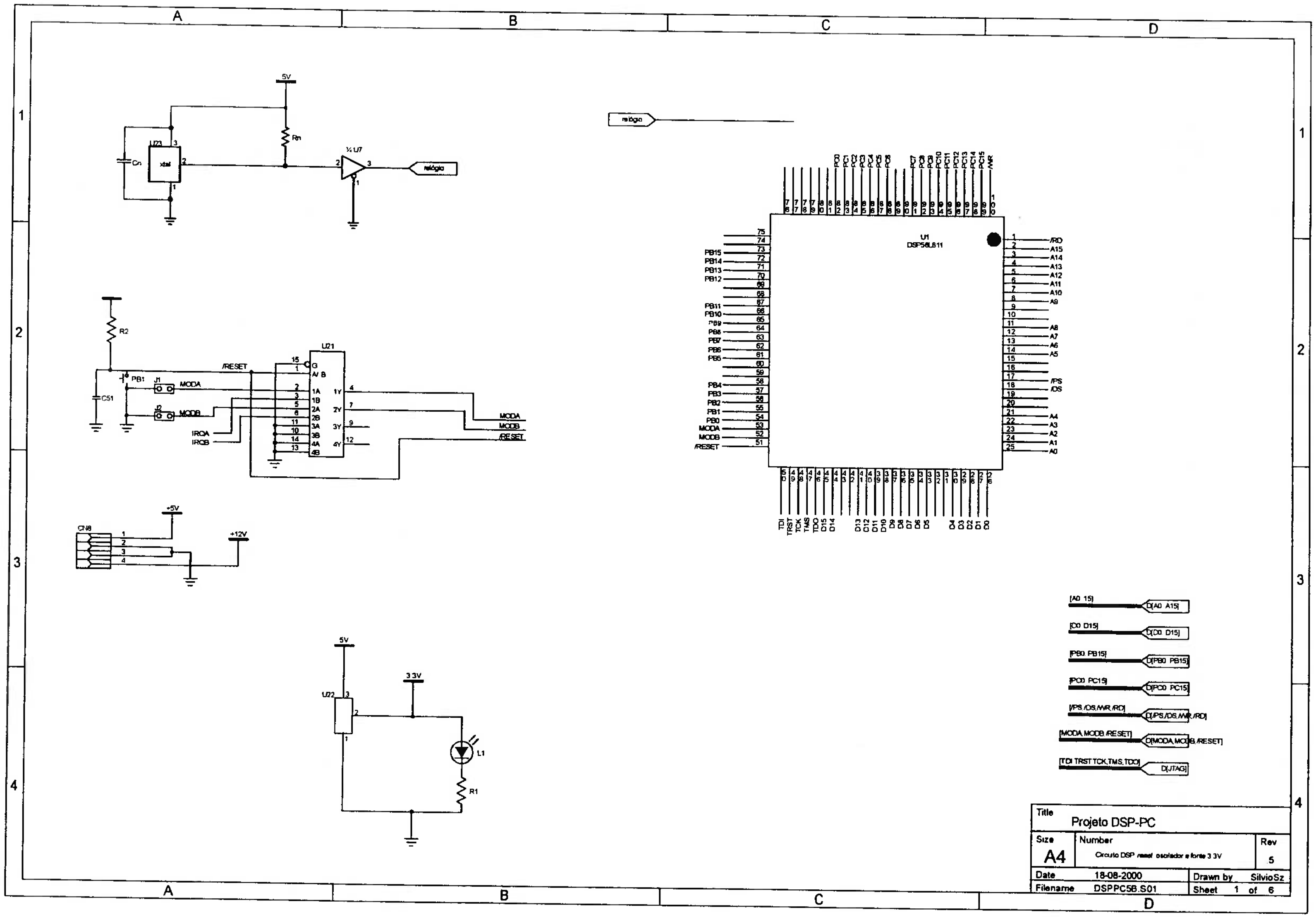
```
Frecebe:
move #$A090,X:$FFD2          ; SCR2 = Rx
move X:$FFD0,Y0              ; recebe <parret1>
RTS

GLOBAL Fle_pc                        ; funcao le_pc()
                                            ; retorna <parret1>

Fle_pc:
move #$0000,X:$FFEE          ; PC0-PC15 = entrada
move X:$FFEF,Y0              ; <parret1> = PCD
RTS

GLOBAL Fescreve_pc                   ; funcao escreve_pc(<par1>)

Fescreve_pc:
move #$FFFF,X:$FFEE          ; PC0-PC15 = saida
move Y0,X:$FFEF              ; PCD = <par1>
RTS

GLOBAL Fle_pb                        ; funcao le_pb()
                                            ; retorna <parret1>

Fle_pb:
move #$0000,X:$FFEB          ; PC0-PC15 = entrada
move X:$FFEC,Y0              ; <parret1> = PCD
RTS

GLOBAL Fescreve_pb                   ; funcao escreve_pb(<par1>)

Fescreve_pb:
MOVE   #$FFFF,X:$FFEB  ; seleciona PB0..PB15 como
; saidas, no PBDDR
MOVE   Y0,X:$FFEC            ; move passagem <par1> para
; PBD, via Y0
RTS

GLOBAL Fsetaint              ; funcao setaint()

Fsetaint:
bfclr #$0200,SR
bfset #$0800,X:$FFFB
RTS

GLOBAL Fresetaint            ; funcao resetaint()

Fdisableints:
bfset #$0200,SR
bfclr #$0800,X:$FFFB
RTS

GLOBAL Fpwrite               ; funcao pwrite(<par1>, <par2>)

Fpwrite:
move  Y1,R0
nop                                  ; trava 1 ciclo (pipeline)
move  Y0,P:(R0)+
RTS

GLOBAL FsetaTimer0     ; funcao setaTimer0(<par1>, <par2>)

FsetaTimer0:
move  #$C000,x0
move  x0,X:$FFED
move  #$0,x0
move  x0,X:$FFF3
move  #$0,x0
move  x0,X:$FFF2
move  Y0,X:$FFDD             ; "timer count"
move  Y1,X:$FFDE             ; "timer PL"
bfclr #$0200,SR
bfset #$0800,X:$FFFB
move  #$009C,x0
move  x0,X:$FFDF
```

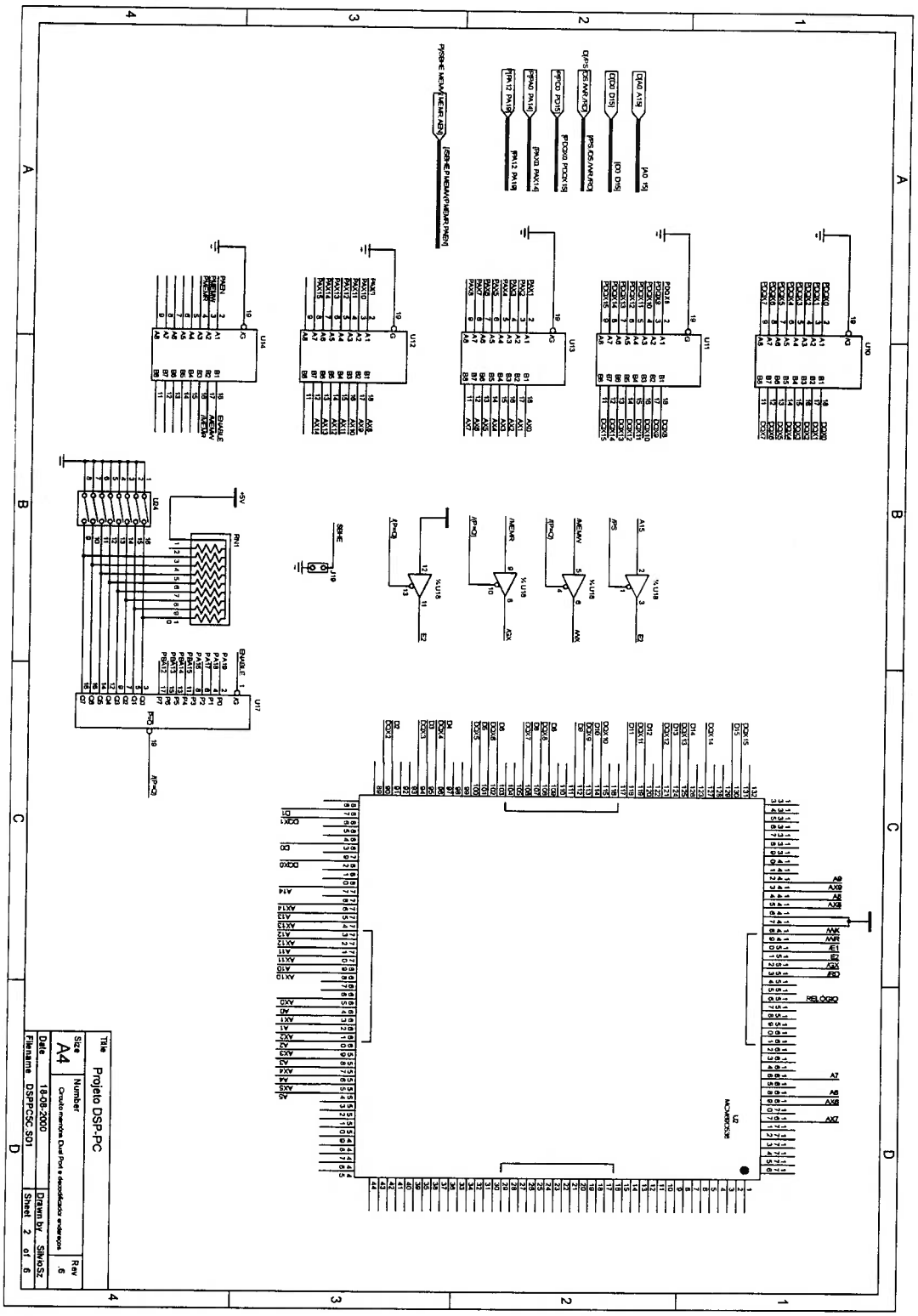
```
rts

GLOBAL FintTimer0                    ; funcao intTimer0(<par1>)

FintTimer0:
move  #$0019,R0
nop
move  Y0,P:(R0)+
nop
move     #$E9C8,x0
move     x0,X:$0018
rts

GLOBAL FsetaTimer1     ; funcao setaTimer1(<par1>, <par2>)

FsetaTimer1:
move  #$C000,x0
move  x0,X:$FFED
move  #$0,x0
move  x0,X:$FFF3
move  #$0,x0
move  x0,X:$FFF2
move  Y0,X:$FFDB              ; "timer count"
move  Y1,X:$FFDC              ; "timer PL"
bfclr #$0200,SR
bfset #$0800,X:$FFFB
move  #$9C00,x0
move  x0,X:$FFDF
RTS

GLOBAL FintTimer1                    ; funcao intTimer1(<par1>)

FintTimer1:
move  #$001B,R0
nop
move  Y0,P:(R0)+
nop
move     #$E9C8,x0
move     x0,X:$001A
RTS

GLOBAL FsetaTimer2     ; funcao setaTimer2(<par1>, <par2>)

FsetaTimer2:
move  #$C000,x0
move  x0,X:$FFED
move  #$0,x0
move  x0,X:$FFF3
move  #$0,x0
move  x0,X:$FFF2
move  Y0,X:$FFD8              ; "timer count"
move  Y1,X:$FFD9              ; "timer PL"
bfclr #$0200,SR
bfset #$0800,X:$FFFB
move  #$009C,x0
move  x0,X:$FFDA
rts

GLOBAL FintTimer2                    ; funcao intTimer2(<par1>)

FintTimer2:
move  #$001D,R0
nop
move  Y0,P:(R0)+
nop
move     #$E9C8,x0
move     x0,X:$001C
rts

GLOBAL FinsCOP                ; funcao insCOP()

FinsCOP:
move #$0003,R0
nop
move  Y0,P:(R0)+
```

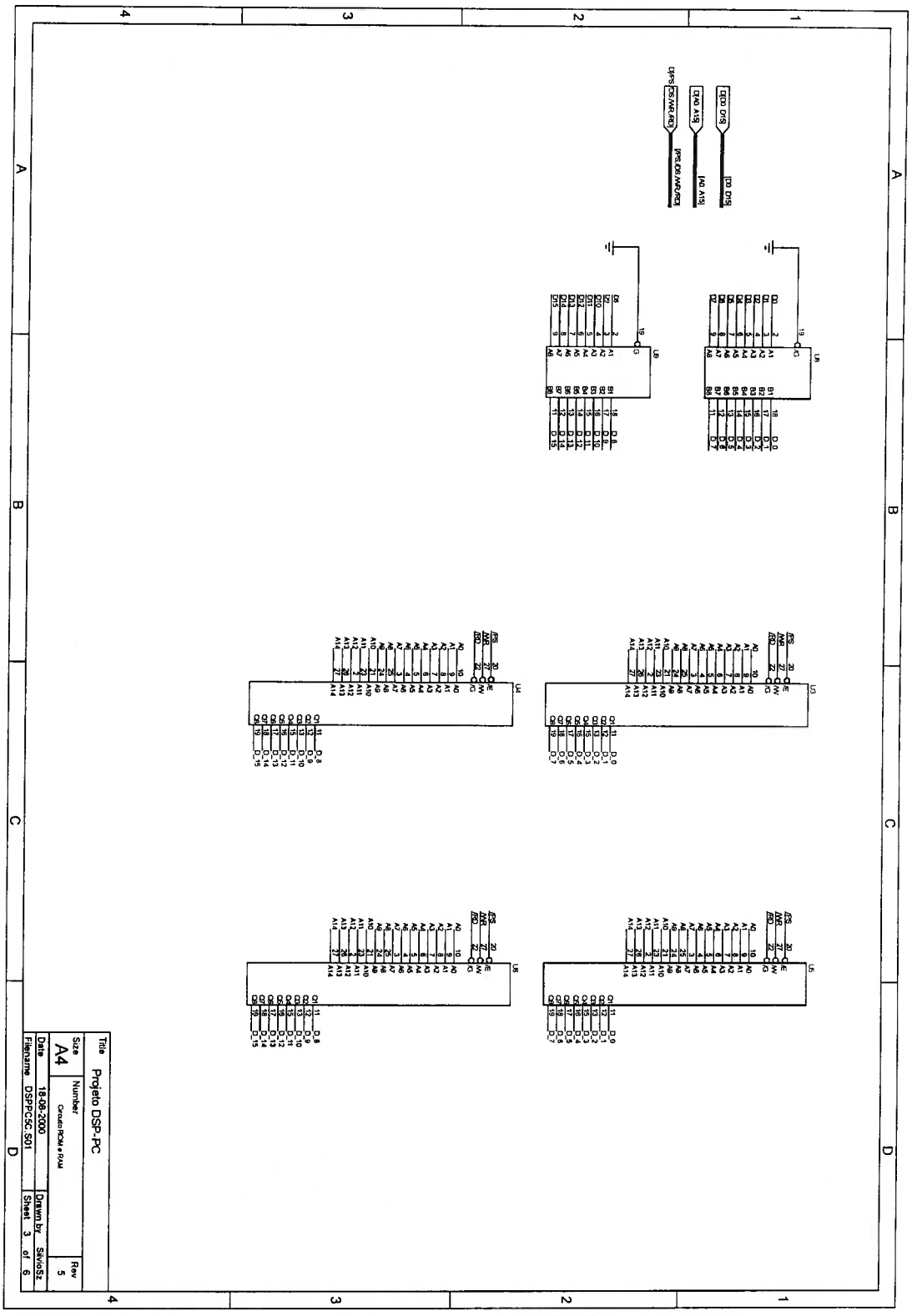
```
nop
move #$E9C8,x0
move x0,X:$0002
nop
movep #$06C0,X:$FFF3
movep #$5555,X:$FFF0
nop
movep #$AAAA,X:$FFF0
nop
movep #$5555,X:$FFF0
nop
bfclr #$0800,X:$FFF1
movep #$C1FF,X:$FFF1
bfset #$0800,X:$FFF1
movep #$AAAA,X:$FFF0
nop
RTS

GLOBAL Fcopok                   ; funcao copok()

FcopOK:
movep #$5555,X:$FFF0            // inicio sequencia reset COP
nop
movep #$AAAA,X:$FFF0
nop
movep #$5555,X:$FFF0
nop
movep #$AAAA,X:$FFF0
nop
RTS


ENDSEC
```

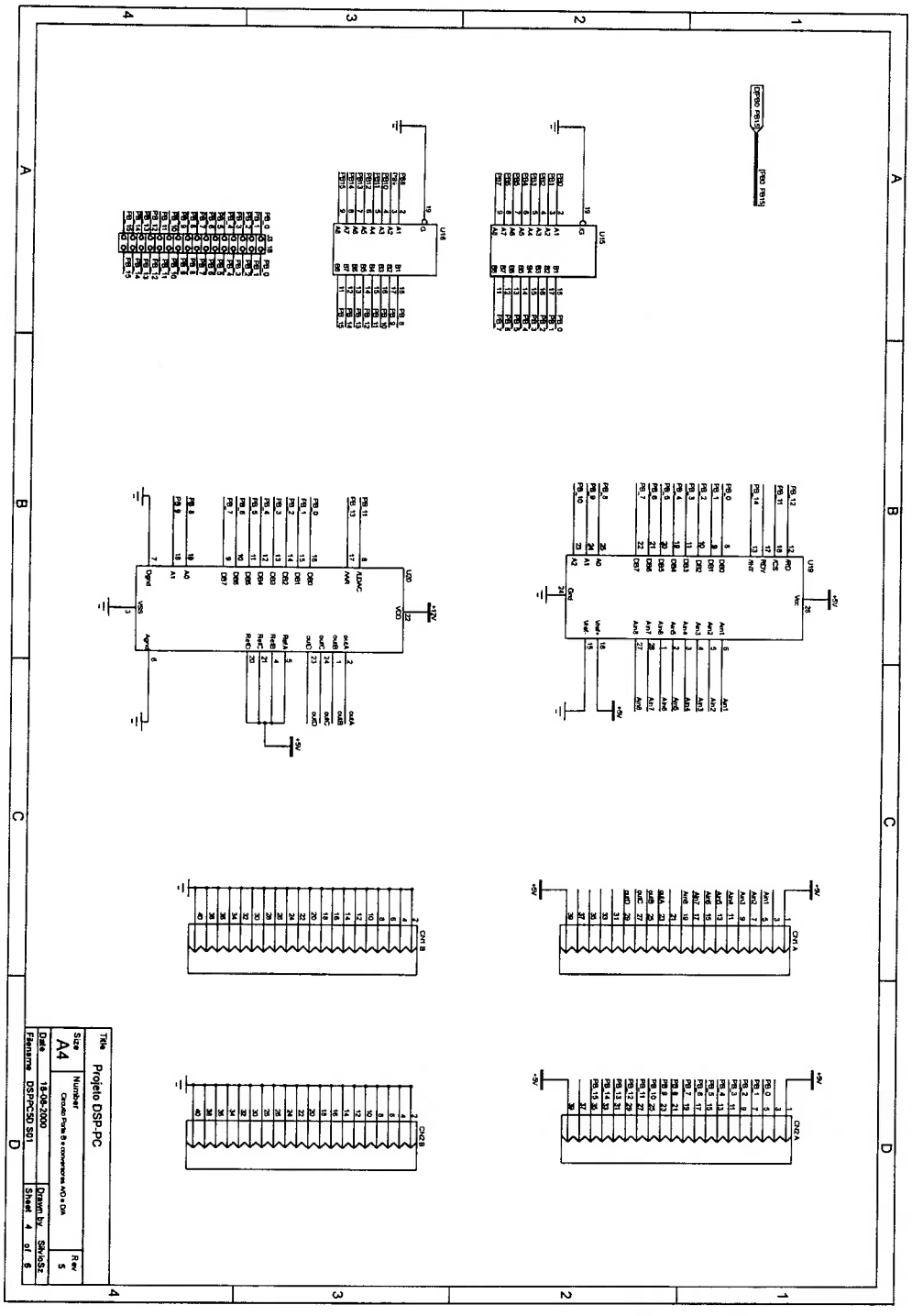
APÊNDICE 2: ESQUEMÁTICOS SISTEMA BASEADO EM DSP

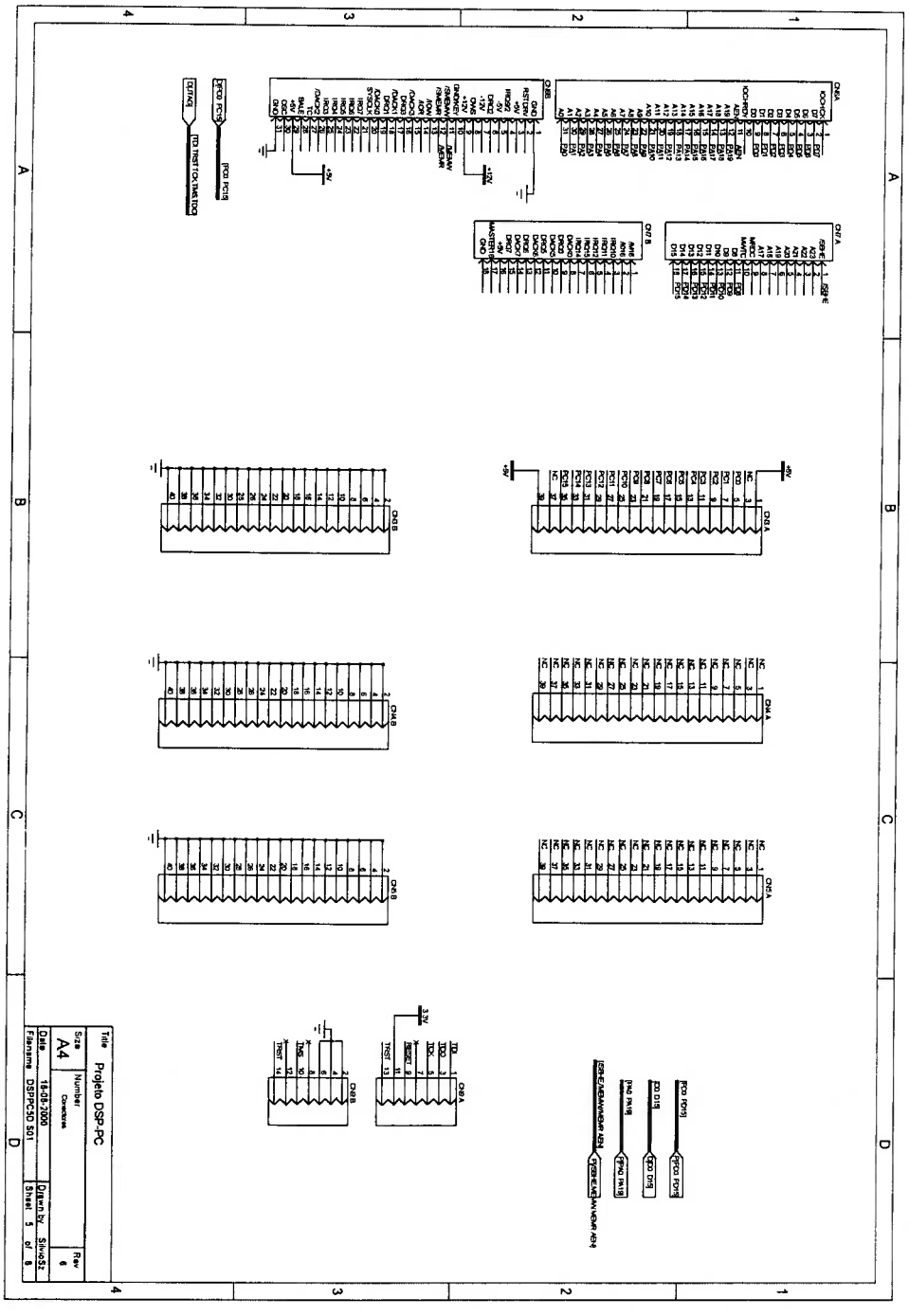


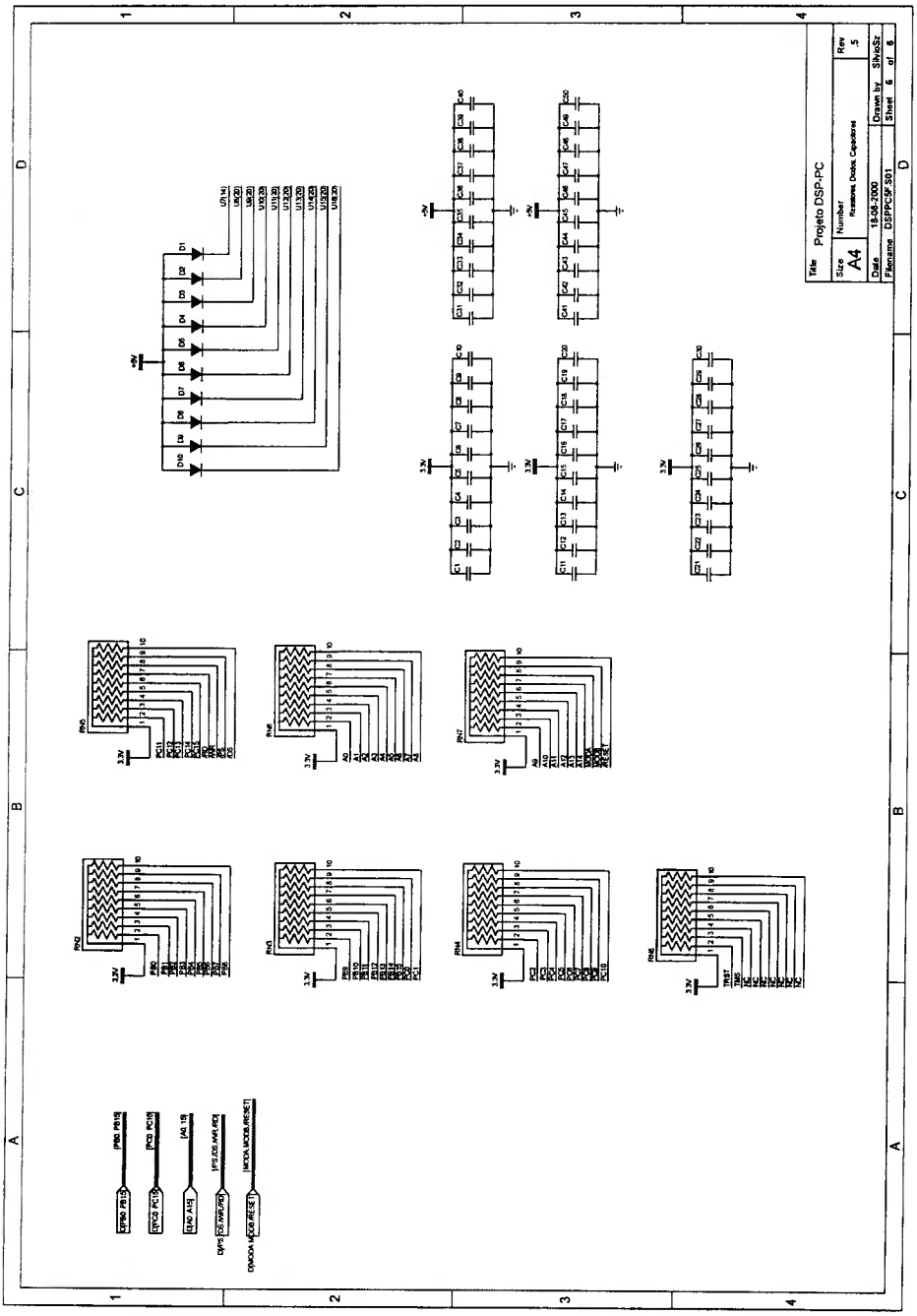

Title
Projeto DSP-PC
Size
A4
Number
Circuito DSP reset oscilador e fonte 3.3V
Rev
5
Date
18-08-2000
Drawn by
SilvioSz
Filename
DSPPC56.S01
Sheet 1 of 6
U1
DSP56L811
U21
U22
U23
xtal
½ U7
relógio
/RESET
MODA
MODB
IRQA
IRQB
PB1
J1
J2
C51
R2
Rn
Cn
CN8
+5V
+12V
5V
3.3V
L1
R1
[A0 15]
D[A0 A15]
[D0 D15]
D[D0 D15]
[PB0 PB15]
D[PB0 PB15]
[PC0 PC15]
D[PC0 PC15]
[/PS,/DS,/WR,/RD]
D[/PS,/DS,/WR,/RD]
[MODA,MODB,/RESET]
D[MODA,MODB,/RESET]
[TDI,TRST,TCK,TMS,TDO]
D[JTAG]


Projeto DSP-PC
A4
Circuito memórias Dual Port e descodificador endereços
18-09-2000
DSPPCSC.SCI
Sheet 2 of 6


Title
Projeto DSP-PC
Size
A4
Number
Circuito RAM e ROM
Rev
5
Date
18-08-2000
Drawn by
SilvioSz
Filename
DSPPCSC.S01
Sheet 3 of 6


Projeto DSP-PC
A4
Number
Rev
5
Date
15-09-2000
Sheet 4 of 8
DSPPC50.SCH
Drawn by:
Circuito Prova B e conversores A/D e D/A


Projeto DSP-PC
A4
15-05-2000
DSPPCSD.SCI
Sheet 5 of 6
Rev 6


Title Projeto DSP-PC
Size A4
Number
Revision
Rev .5
Date 18-08-2000
Drawn by SilvaSz
Filename DSPPC3F.SCH
Sheet 6 of 6

APÊNDICE 3: PROGRAMA DP.C

```
/*
 / ------------------------------------------------------------------
 / programa:DP.c
 / Obs.:    Le arquivo dados SREC, manipula dados  e
 /          escreve na Dual Port do DSP-PC
 /
 /      le arquivo SREC e coloca na Dual Port a partir de POS
 /
 / ------------------------------------------------------------------
*/

#include <dos.h>
#include <stdio.h>
#define  POS    0xD800   // endereco DP

char MARCA_DADOS='1';
char DEBUG = 1;
char DEBUG_ = 1;
char BYTES_ENDERECO = 2;


unsigned char AsciiToHex(char c1, char c2)
{
  char DEBUG=0;
  char b1;
  char b2;
  if (c1>57) {b1=c1-55;} else {b1=c1-48;}
  if (c2>57) {b2=c2-55;} else {b2=c2-48;}
  b1=b1*16;
  if (DEBUG) printf("%2x %2x %2x\n", b1, b2, b1+b2);
  return(b1+b2);
}


void main(void)
{
  FILE *fp;
  unsigned int far *pt;

  char c;
  int i;
  int i1;
  int i2;
  char LP; // line pointer;

  unsigned int BUFFER[16000];
  unsigned int BUFPTR;
  unsigned long Temp;


  char LB[100]; // line buffer
  unsigned int  DADO;
  unsigned int  NBL; // numero de bytes por linha
  unsigned long ADDR;
  unsigned long ULT_ADDR = 0;
```

```
  unsigned char CRC;


// ponteiro para DP
   pt = (unsigned int *) MK_FP(POS, 0);


  for (i2=0; i2<1000; i2++)
    BUFFER[i2]=0;


  if (DEBUG_) printf("\n\n\n\n\n\n\n\n\n\n");


  if ((fp = fopen("<NOME DO ARQUIVO SREC>", "r")) == NULL)
  {
    printf("1 - Erro abrindo o arquivo\n");
  }
  else
  {
    do
    {
      fscanf(fp, "%s", &LB);
      printf("%s\n", LB);
    }
    while (!feof(fp));
    fclose(fp);
  }

  if (DEBUG_) printf("!\n");


  if ((fp = fopen("<NOME DO ARQUIVO SREC>", "r")) == NULL)
  {
    printf("2 - Erro abrindo o arquivo\n");
  }
  else
  {
    do
    {
      fscanf(fp, "%s", &LB);
      if (LB[1]==MARCA_DADOS)
      {
    // Pega numero de bytes na linha
    if (DEBUG) printf("%s\n", LB);
    NBL = AsciiToHex(LB[2], LB[3]);

    // Pega endereco destino
    if (BYTES_ENDERECO==2)
    {
      ADDR = AsciiToHex(LB[4], LB[5]);
      ADDR = ADDR*256 + AsciiToHex(LB[6], LB[7]);
      LP=8; // Posicao dos dados no LB
      NBL=(NBL-3)/2; // desconta endereco(2B) e crc(1B)
    }

    if (BYTES_ENDERECO==4)
    {
      ADDR = AsciiToHex(LB[4], LB[5]);
      ADDR = ADDR*256 + AsciiToHex(LB[6], LB[7]);
```

```
        ADDR = ADDR*256 + AsciiToHex(LB[8], LB[9]);
        ADDR = ADDR*256 + AsciiToHex(LB[10], LB[11]);
        LP=12; // Posicao dos dados no LB
        NBL=(NBL-5)/2; // desconta endereco(4B) e crc(1B)
      }

      if (ULT_ADDR<(ADDR+NBL-1)) ULT_ADDR=ADDR+NBL-1;
      if (DEBUG_) printf("bytes na linha: %d\n", NBL);
      if (DEBUG_) printf("endereco da linha: %x\n", ADDR);
      // Pega os dados
      BUFPTR=ADDR;
      for (i=1; i<=NBL; i++) // ((NBL-2)/2)
      {
        DADO=AsciiToHex(LB[LP], LB[LP+1]);
        LP=LP+2;
        DADO=DADO + AsciiToHex(LB[LP], LB[LP+1])*256;
        LP=LP+2;
        if (DEBUG_) printf("%x ", DADO);

        // Calculo do CRC
        BUFFER[BUFPTR]=DADO;
        BUFPTR++;
      }
      if (DEBUG) printf("\n");

      // Pega o CRC
      CRC=AsciiToHex(LB[LP], LB[LP+1]);
      if (DEBUG_) printf("CRC=%x\n", CRC);
        }
      }
      while (!feof(fp));
      fclose(fp);
    }
    // DUMP
    if (DEBUG_) printf("Ultimo endereco usado = %04X\n", ULT_ADDR);
    for (i1=0; i1<(ULT_ADDR/8+1); i1++)
    {
      if (DEBUG) printf("%04X = ", i1*8);
      for (i2=0; i2<8; i2++)
      {
          if (DEBUG) printf("%04X ", BUFFER[i2+i1*8]);
      }
      printf("\n");
    }
    if (DEBUG) printf("*\n");
    // escreve na DP
    for (i=0; i<128; i++)
    {
      if (DEBUG) printf("%04X ", BUFFER[i]);
      Temp=BUFFER[i] + BUFFER[i]*65536;
      *(pt+i) = BUFFER[i];
      delay(15);
    }
    if (DEBUG) printf("**\n");
    for (i=0; i<128; i++)
    {
      if (DEBUG) printf("%04X ", *(pt+i));
      delay(15);
    }
}
```

APÊNDICE 4: CIRCUITO ANALÓGICO DO MOTOR DE CORRENTE CONTÍNUA

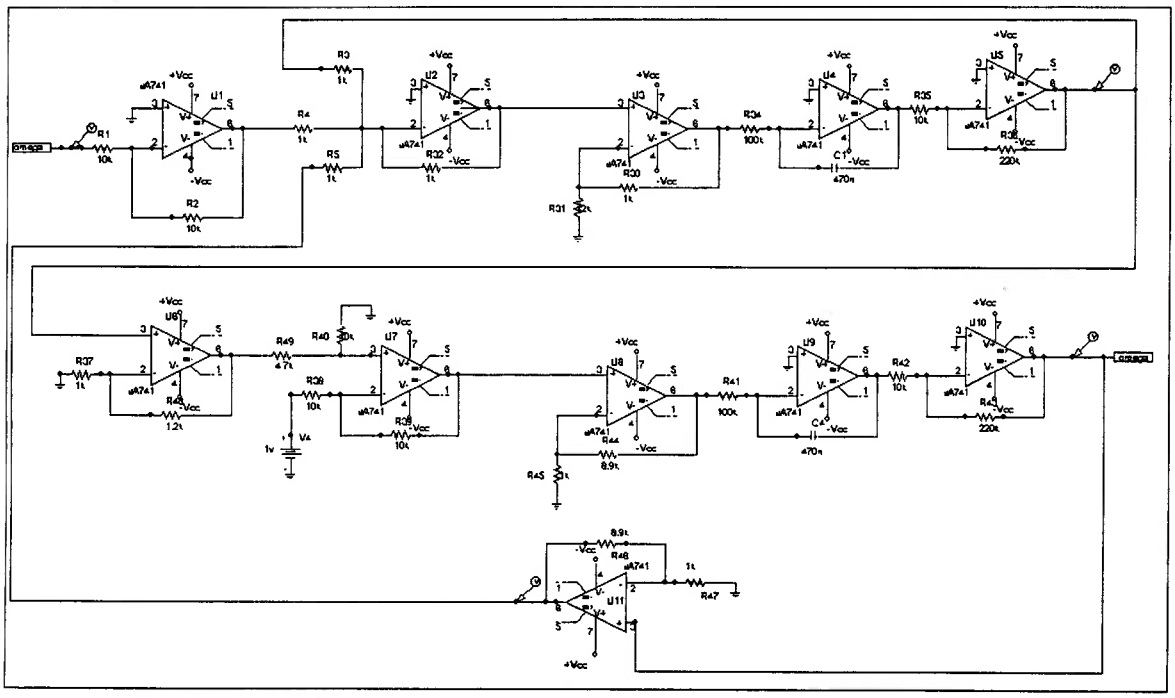


Fig. 1: Circuito analógico do motor de corrente continua

Tabela A4-1: Especificações do motor de corrente contínua

Potência	7,5 kW
Rotação mínima	20 rpm
Rotação nominal	1000 rpm
Rotação máxima	1800 rpm
Tensão de armadura	240 V
Corrente de armadura	37,4 A
Tensão de campo	190 V
Momento de Inércia	0,8 $Kg.m^2$
Conjugado nominal	71,60 N.m
Conjugado rotação máxima	39,79 N.m
Conjugado de partida	143,20 N.m

Tabela A4-2: Parâmetros da simulação do modelo do motor de corrente contínua.

J	0.10
K_e	1.8960
K_T	1.8960
k_{TL}	0.0062
R_a	1
L_a	0.12
R_f	120
L_f	0.12
V_a	240
V_f	190
T_L	50

Tabela A4-3:. Ponto de operação nominal do motor

ed.	Tq. carga		Vel. (ω)		Corrente	
	Volts	Nm	mV	rpm	mV	Amp
01	5,77	161,1	0	0	1960	83,97
02	5,45	152,2	69	131,9	1850	79,26
03	5,13	143,2	107,5	205,5	1740	74,54
04	4,81	134,3	157	300,2	1630	69,83
05	4,49	125,3	207	395,8	1520	65,12
06	4,17	116,4	257,5	492,3	1410	60,41
07	3,85	107,4	307	586,9	1310	56,12
08	3,53	98,5	357	682,5	1190	50,98
09	3,21	89,5	411	785,8	1070	45,84
10	2,89	80,6	473	904,3	973	41,68

Med.	Tq. carga		Vel. (ω)		Corrente	
	Volts	Nm	m V	rpm	mV	Amp
11	2,56	71,6	526	1005,6	873	37,40
12	2,24	62,7	573	1095,5	766	32,82
13	1,92	53,7	626	1196,8	663	28,40
14	1,60	44,8	680	1300,1	546	23,39
15	1,28	35,8	733	1401,4	440	18,85
16	0,96	26,9	786	1502,7	323	13,84
17	0,64	17,9	840	1605,9	217	9,30
18	0,32	9,0	886	1693,9	107	4,58
19	0,00	0,0	941,5	1800	0	0,00
20						

Tabela A4-4: Variação da velocidade com a tensão de armadura Va

Medição	Va		Velocidade (ω)	
	miliVolts	Volts	miliVolts	rpm
01	2030	240,00	526	1005,60
02	1917,8	226,73	469	896,62
03	1805,6	213,47	420	715,93
04	1693,4	200,20	373	507,69
05	1581,2	186,94	320	308,86
06	1469	173,67	260	152,66
07	1356,8	160,41	206	59,79
08	1244,6	147,14	146	16,59
09	1132,4	133,88	106	3,34
10	1020,2	120,61	46	0,29
11	908,0	107,35	0	0

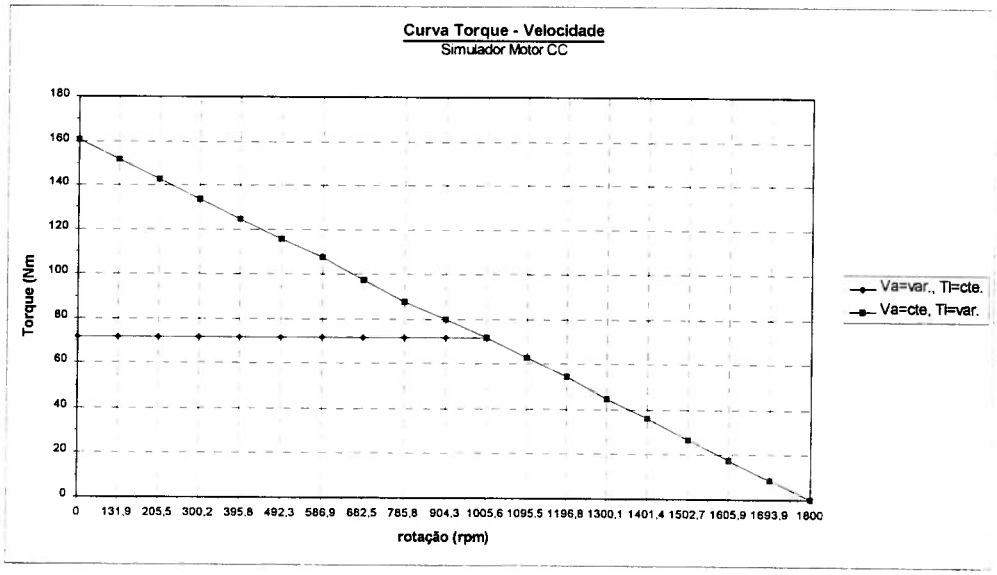


Fig. A4-2: Curva característica de torque e velocidade obtida experimentalmente

APÊNDICE 5: SIMULINK / MATLAB

Modelos utilizados nas simulações

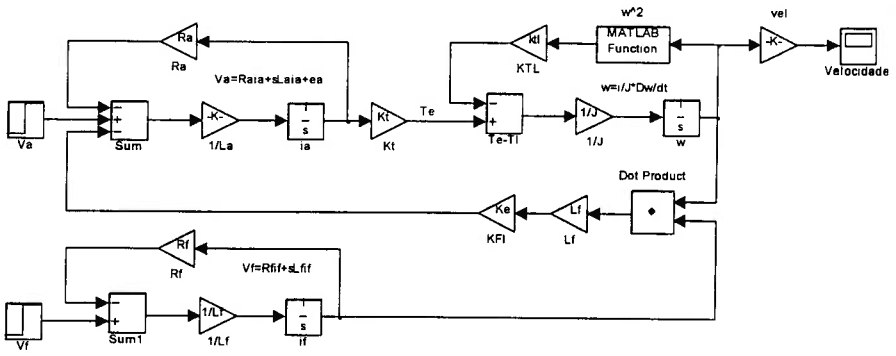


Fig. A5-1: Modelagem do motor de corrente contínua, do capítulo 2.

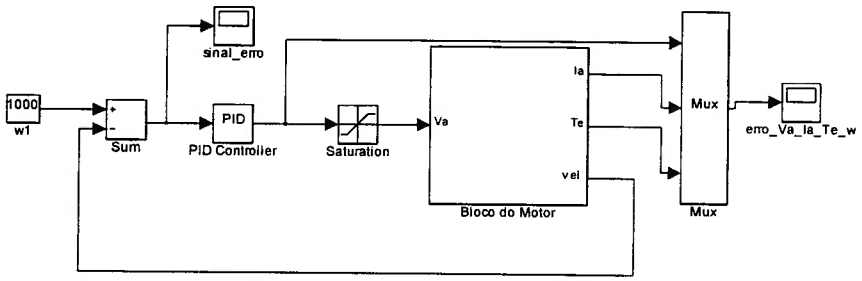


Fig. A5-2: Diagrama de blocos da malha de velocidade com o regulador PI.

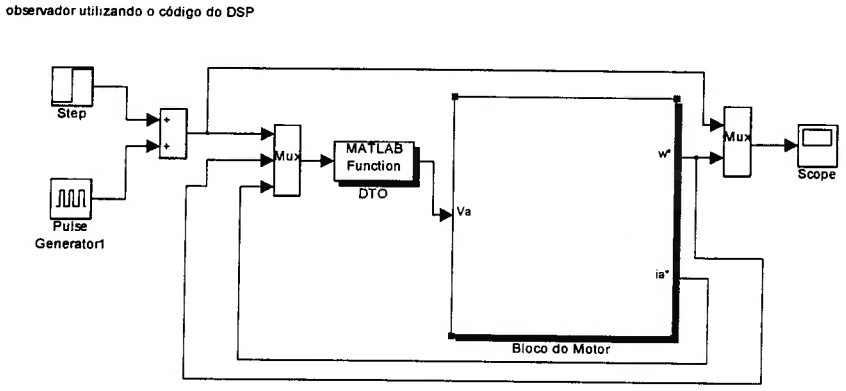


Fig. A5-3: Simulação em MatLab do DTO implementado no DSP

APÊNDICE 6: CÓDIGO PI E DTO

Código utilizado para gerar o controlador PI e o Observador de Distúrbios de Torque (DTO) no sistema baseado em DSP.

Arquivo:me02.c (Código do controlador PI)

```
/*
+-------------------------------------------------------------------------------------
| programa:            me02.c
|
| obs.:
|
+-------------------------------------------------------------------------------------
*/

// PI defines

#define Ts      0.001          //      Time Sampling
#define Kp1     1              //      Kp for velocity PI control loop
#define Ki      15             //      Ki for velocity PI control loop

#define Va_max 200             //      Higher Output Limit (200 ~ 2 V DAC output)
#define Va_min  0              //      Lower Output Limit

#define Kt      1.120          //      Torque Motor Constant
#define J       0.100          //      J


typedef unsigned short WORD;
WORD *dataptr;

int AlwaysTrue=1;
int j;

short   Wref = 0,              //      SetPoint
        Wm = 0;                //      w(k)

short   e1 = 0,                // e(k)
        e1old = 0;             // e(k-1)

short   P = 0,
        I = 0,
        Iold = 0;

short   Vout = 0,              //Va(k)
        Voutold = 0,
        u = 0;

// Prototypes

void    inicia_b(void);
void    inicia_c(void);
void    escrita(int,short);
short   leitura(int);
void    setaint(void);
void    resetaint(void);
void    setaTimer2(int,int);
short   intTimer2((WORD));

//      ******************************************************

void main(void)
{
        inicia_b();
        inicia_c();
        SetaTimer2(0, 400)
```

```
        IntTimer2((WORD)pi);                    // ISR pi

        escrita(1,0);

        setaint();

        while(AlwaysTrue)
        {
        }       // while

}       // main


// ******************************

void pi(void)
{

resetaint();

Wref =  leitura(1);

Wm = leitura(2);

e1 = Wref - Wm;

P = Kp1 * e1;

Vout = Voutold + (Ki / 2 * Ts * (e1old)) + (Ki / 2 * Ts + Kp1) * e1;

u = Vout;

if(u > Va_max)
        u = Va_max;
if(u < Va_min)
        u = Va_min;

escrita(1, u);

e1old = e1;
Voutold = Vout;

setaint();

}       //      pi
```

Arquivo:me06_com_DTO.c (Código com o observador de distúrbios de torque)

```
/*
+-------------------------------------------------------------------------------------
| programa:     me06_com_DTO.c
|
| obs.:         Controle PI + DTO, sem Timer2 !
|
+-------------------------------------------------------------------------------------
*/

#define Ts              0.0005          //      Time Sampling
#define Ta              0.10            //      Ra / La
#define Kp1             1               //      Kp for velocity PI control loop
#define Ki              200             //      Ki for velocity PI control loop
#define Kp2             .7              //      Kp for inner current P control loop

#define Va_max          200                     //      200 ~ 2 V DAC output
#define Va_min          0                       //      Lower Output Limit

#define Kt              1.120                   //      Torque Motor Constant
#define Ktlinha         1.20                    //      Torque Motor Constant for DTO
#define J               0.0100                  //      J

typedef unsigned short WORD;
WORD *dataptr;
```

```
int     AlwaysTrue=1;
int     j;
short   Wref = 0,                                       //      SetPoint
        Wm = 0,                                         //      w(k), measured velocity
        Wmold = 0;
short   e1 = 0,                                         // e(k)
        e1old = 0,                                      // e(k-1)
        e2,
        e3;
short   P = 0,
        I = 0,
        Iold = 0;
short   Tdis = 0,
        Tdisold = 0,
        T1 = 0,
        T2 = 0,
        T2old = 0;
short   i = 0,                          //      Va(k) output command to motor simulator
        iold = 0,
        im = 0,                                                 //      i motor
        Va = 0,
        u = 0;

// Prototypes

void    inicia_b(void);
void    inicia_c(void);
void    escrita(int,short);
short   leitura(int);

//      ******************************************************

void main(void)
{
        inicia_b();
        inicia_c();

        escrita(1,0);                                           // turn DAC outputs = 0 !
        escrita(2,0);
        escrita(3,0);
        escrita(4,0);

        while(AlwaysTrue)
        {
                Wref =  leitura(1);

                Wm = leitura(2);

                e1 = Wref - Wm;
                P = Kp1 * e1;
                i = iold + (Ki / 2 * Ts * (e1old)) + (Ki / 2 * Ts + Kp1) * e1;
                T2 = ( J * ( Wm - Wmold ) + T2old ) / ( Ts + Ta );
                e2 = i - Tdisold * Ktlinha;
                T1 = e2 * Kt;
                Tdis = T2 + T1;

                im = leitura(3);

                e3 = e2 - im;
                Va = e3 * Kp2;
                u = Va;
                if(u > Va_max)
                        u = Va_max;
                if(u < Va_min)
                        u = Va_min;

                escrita(1,u);

                for(j=0; j<100; j++);
                e1old = e1;
                iold = i;

        }       // while
}       // main
```